Создай пользовательский интерфейс в стиле Web 2.0 с помощью ASP.NET AJAX

Рассмотрены пакеты Control Toolkit, Core, Futures CTP и другие

Программирование в

ASP.NET AJAX

СИМВОЛ®

O'REILLY®

Кристиан Венц

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-122-3, название «Программирование в ASP.NET AJAX» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

Programming ASP.NET AJAX

Christian Wenz

O'REILLY®

Программирование в ASP.NET AJAX

Кристиан Венц

Санкт-Петербург — Москва
2008

Кристиан Венц

Программирование в ASP.NET AJAX

Перевод А. Киселева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*Б. Попов*
Редактор	*Ю. Бочина*
Корректор	*С. Минин*
Верстка	*Д. Орлова*

Венц К.

Программирование в ASP.NET AJAX. – Пер. с англ. – СПб: Символ-Плюс, 2008. – 512 с., ил.

ISBN-13: 978-5-93286-122-6
ISBN-10: 5-93286-122-3

Книга Кристиана Венца «Программирование в ASP.NET AJAX» – полное практическое введение в новую платформу Microsoft ASP.NET AJAX 1.0, которая предоставляет большие преимущества при разработке Ajax-приложений, схожие с теми, которые дает ASP.NET для разработки серверных сценариев. Вы узнаете, как с помощью технологий Ajax без особых затрат времени создавать профессионально выполненные динамические веб-страницы.

После общего обзора платформы и основ JavaScript и Ajax рассмотрена организация составляющих частей .NET, включая пакеты Extensions, Control Toolkit, Futures CTP и Microsoft AJAX Library. Также читателю предлагается исчерпывающая глава об элементе управления UpdatePanel, который позволяет организовать независимое обновление отдельных частей веб-страницы.

Насыщенная примерами, демонстрирующими ключевые аспекты платформы, эта книга идеально подходит не только разработчикам ASP.NET, стремящимся расширить свои возможности за счет использования Ajax, но и всем специалистам, кто интересуется данной платформой, независимо от того, какие технологии они используют в настоящее время.

ISBN–13: 978–5–93286–122–6
ISBN–10: 5–93286–122–3
ISBN 0–596–51424–7 (англ)

© Издательство Символ-Плюс, 2008
Authorized translation of the English edition © 2007 O'Reilly Media, Inc. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 31.07.2008. Формат 70×100 1/16. Печать офсетная.
Объем 32 печ. л. Тираж 2000 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие 11

I. Основы 21

1. ASP.NET AJAX, Ajax и ASP.NET 23

ASP.NET AJAX и Ajax 23
ASP.NET AJAX и ASP.NET 25
Подготовка и установка ASP.NET AJAX 28
Структура и архитектура ASP.NET AJAX 34
Первый пример ASP.NET AJAX: Hello User 35
Элемент управления ScriptManager 39
Подведение итогов 42
Для дополнительного чтения 42

2. JavaScript 43

Язык программирования JavaScript 46
Объектно-ориентированное программирование (ООП) 57
Доступ к элементам страницы 61
Методы DOM 65
Подведение итогов 66
Для дополнительного чтения 67

3. Ajax 68

Объект XMLHttpRequest 69
Объект XMLDocument 81
JSON 86
Подведение итогов 89
Для дополнительного чтения 90

II. Расширения ASP.NET AJAX 91

4. Использование расширений JavaScript в ASP.NET AJAX 93

Псевдонимы и вспомогательные функции ASP.NET AJAX 93
Расширения к существующим объектам JavaScript 96

Объектно-ориентированные возможности JavaScript в ASP.NET AJAX . . . 97
Клиентские версии классов .NET . . . 111
Подведение итогов . . . 115
Для дополнительного чтения . . . 115

5. Веб-службы . . . 116

Обработка ошибок . . . 116
Методы страницы . . . 121
Управление состоянием сеанса . . . 125
Обмен данными со сложной структурой . . . 130
Доступ к веб-службам из JavaScript . . . 134
Подведение итогов . . . 145
Для дополнительного чтения . . . 145

6. UpdatePanel: обновление только части страницы . . . 146

Создание обновляемой области страницы . . . 147
Подведение итогов . . . 162
Для дополнительного чтения . . . 163

7. Использование службы управления профилями в ASP.NET AJAX . . . 164

Подготовка веб-сайта . . . 165
Доступ к данным в профиле . . . 166
Доступ к сгруппированным данным в профиле . . . 171
Подведение итогов . . . 176
Для дополнительного чтения . . . 176

8. Использование службы аутентификации в ASP.NET AJAX . . . 177

Подготовка приложения . . . 177
Вход и выход . . . 180
Подведение итогов . . . 185
Для дополнительного чтения . . . 186

9. Локализация и глобализация приложений . . . 187

Локализация . . . 188
Глобализация и интернационализация . . . 203
Подведение итогов . . . 207
Для дополнительного чтения . . . 207

III. ASP.NET AJAX Control Toolkit . . . 209

10. Использование Control Toolkit . . . 211

Установка пакета Control Toolkit . . . 212
Использование пакета Control Toolkit . . . 216

Подведение итогов 218
Для дополнительного чтения 219

11. Добавление анимационных эффектов в веб-страницу 220

Платформа создания анимационных эффектов 220
Перетащить-и-отпустить 229
Подведение итогов 232
Для дополнительного чтения 232

12. Автодополнение ввода пользователя, борьба со спамом и не только 233

Создание раздвижных панелей 233
Управление относительным положением элемента 236
Добавление функции автодополнения к элементу управления TextBox 237
Присоединение календаря к текстовому полю 246
Динамическая свертка единственной панели 247
Отображение всплывающего окна на странице 248
Борьба со спамом в блогах и других формах ввода информации 252
Создание интерфейса с вкладками 254
Подведение итогов 256
Для дополнительного чтения 257

13. Создание собственных элементов управления и помощь сообществу 258

Создание собственных элементов управления ASP.NET AJAX 258
Содействие проекту Control Toolkit 268
Подведение итогов 277
Для дополнительного чтения 278

IV. ASP.NET AJAX Futures 279

14. Клиентские элементы управления 281

Введение в клиентские элементы управления ASP.NET AJAX 281
Использование элементов управления ASP.NET AJAX 282
Обработка событий в элементах управления 300
Подведение итогов 304
Для дополнительного чтения 304

15. Привязка и проверка данных 305

Привязка данных 305
Проверка данных 323
Подведение итогов 340
Для дополнительного чтения 340

16. Использование клиентских аспектов поведения и компонентов . . . 341

Использование аспектов поведения . . . 341
Использование компонентов . . . 355
Подведение итогов . . . 358
Для дополнительного чтения . . . 358

17. Использование данных, размещенных на стороне сервера . . . 359

Использование элемента управления ListView . . . 359
Создание собственного источника данных . . . 377
Подведение итогов . . . 383
Для дополнительного чтения . . . 383

18. Использование внешних веб-служб . . . 384

Использование веб-службы Google . . . 385
Использование веб-службы Amazon . . . 393
Преобразование результатов веб-службы с помощью XSLT . . . 398
Использование веб-службы Yahoo! (а также REST и XPath) . . . 407
Подведение итогов . . . 416
Для дополнительного чтения . . . 416

19. Использование анимационных эффектов . . . 417

Использование анимационных эффектов . . . 417
Использование анимации для создания эффекта плавного исчезновения . . . 419
Подведение итогов . . . 430
Для дополнительного чтения . . . 430

20. Кнопки Назад/Вперед и закладки . . . 431

Исправление недостатков программным способом . . . 432
Исправление недостатков в поддержке механизма закладок и кнопок Назад/Вперед с помощью элемента управления UpdateHistory . . . 435
Исправление недостатков в поддержке механизма закладок и кнопок Назад/Вперед с помощью элементов управления из пакета ASP.NET AJAX Futures . . . 440
Подведение итогов . . . 447
Для дополнительного чтения . . . 447

21. Веб-части . . . 448

ASP.NET AJAX и ASP.NET Web Parts . . . 448
Подведение итогов . . . 454
Для дополнительного чтения . . . 454

V. Microsoft AJAX Library 455

22. ASP.NET AJAX в комбинации с другими серверными технологиями 457

Использование ASP.NET AJAX вместе с PHP 458
Подведение итогов 462
Для дополнительного чтения 462

VI. Приложения 463

A. Отладка приложений ASP.NET AJAX 465

B. Справочник по XMLHttpRequest 479

C. Справочник DOM 481

D. Справочник по ASP.NET AJAX 485

E. Справочник по ScriptManager, UpdatePanel, UpdateProgress и Timer 489

Алфавитный указатель 493

Предисловие

В Википедии (на странице *http://en.wikipedia.org/wiki/Ajax*) дается более 40 толкований слова Ajax – имена персонажей из поэмы Гомера «Илиада» (Аякс Великий и Аякс Малый), название футбольной команды из Амстердама, пара моделей автомобилей, кличка лошади и даже мое любимое чистящее средство, выпускаемое компанией Colgate. Но кроме всего прочего, под термином Ajax подразумевается целый набор технологий, которые, по мнению многих, стали революционными во Всемирной паутине. Если верить различным публикациям в Интернете, Ajax – это будущее веб-разработки, основа Web 2.0 и действенное средство против множества проблем.

Многие веб-разработчики стремятся обеспечить своих клиентов приложениями, обладающими богатыми возможностями, не создавая при этом обычные приложения Windows (или, по определенным причинам, не имея возможности их создавать). Технология Ajax предоставляет все необходимое для этих целей. Она позволяет наделять их возможностями, которые раньше были характерны только для обычных настольных приложений, такими как горячие комбинации клавиш и техника «перетащить-и-отпустить» (drag-and-drop).

Платформа ASP.NET под кодовым названием «Atlas», разрабатываемая компанией Microsoft, обеспечивает набор средств, реализующих функциональность Ajax, для разработчиков ASP.NET. Она предлагает большую часть функциональности, доступной для разработки ASP.NET-приложений на стороне сервера. Осенью 2006 года было объявлено о выходе окончательной версии продукта под названием ASP.NET AJAX. (Впрочем, название Atlas более удобно для произношения.)

Я долго воздерживался от написания книги об Ajax. В течение многих лет я использовал на практике и писал о технологиях, являющихся составными частями Ajax, но сам термин появился лишь в начале 2005 года, еще до того, как данная технология стала популярной. На мой взгляд, лучше всего по этому поводу выразился Клеменс Вастерс (Clemens Vasters) в своем блоге, озаглавив эту запись «Web 2.0 бла-бла-бла AJAX бла-бла-бла преимущества!(?)» (*http://vasters.com/clemensv/PermaLink,guid,d88c1112-d8da-496e-9fd0-8cf03cf55c32.aspx*).

Крикливая реклама, которая сейчас развернулась вокруг Ajax, напоминает мне шумиху вокруг XML и веб-служб, поднятую несколько лет тому назад: все говорили о них, но мало кто читал соответствующие

спецификации. После того как шумиха улеглась и появились реальные приложения, оказалось, что объединение этих технологий дало существенный эффект.

Я уверен, что развитие Ajax пойдет тем же путем, только более высокими темпами. Во Всемирной паутине уже сейчас можно обнаружить удобные Ajax-приложения.

Но вернемся к моему нежеланию писать об Ajax.

Я всегда говорил, что описание самой технологии Ajax займет от силы страниц 20–30. Добавив немного вводной информации и примеров, этот объем можно довести до 75 страниц, может быть, даже до 100. Но чем можно было бы заполнить остальную часть книги? Авторам многих книг, посвященных Ajax, которые сейчас можно найти в продаже, пришлось пойти на определенные ухищрения, чтобы достигнуть желаемого количества страниц.

Мое мнение изменилось, когда в сентябре 2005 года я посетил конференцию профессиональных разработчиков Microsoft (Microsoft Professional Developers Conference) в Лос-Анджелесе и впервые увидел Atlas. На этой конференции компания Microsoft объявила о платформе, которая не просто обеспечивает функциональные возможности Ajax, но и предоставляет инструментальные средства, облегчающие разработку современных веб-приложений. Тогда я подумал, что это действительно стоит того, чтобы написать книгу. Я приступил к работе над рукописью, основываясь на предварительной версии Atlas. Мне пришлось несколько раз переписывать ее по мере появления промежуточных версий платформы Atlas, которые попадали мне в руки. Нехватка документации к промежуточным версиям вынуждала меня выполнять самостоятельные исследования внутренних механизмов Atlas. Как результат этих усилий, в этой книге можно найти несколько неофициальных способов решения некоторых задач.

В сентябре 2006 года вышла книга «Programming Atlas». Как одна из первых книг, посвященных этой теме, она содержала подробные сведения о платформе, которая еще продолжала изменяться. В конце января 2007 года вышла окончательная версия ASP.NET AJAX 1.0. Кроме изменения названия, существенные изменения претерпела и внутренняя организация платформы, что потребовало создания новой редакции книги, так как все существующие приложения необходимо было адаптировать под новую версию платформы.

В данной книге рассказывается о том, как создавать профессиональные динамические веб-страницы на основе платформы ASP.NET AJAX. В самом начале книги приводятся некоторые вводные сведения о JavaScript и ASP.NET, поскольку они будут необходимы для понимания темы.

Я стараюсь придерживаться принципа «больше дела, меньше слов». Поэтому книга содержит большое количество примеров, иллюстрирующих ключевые аспекты использования платформы ASP.NET AJAX.

Кроме того, стараясь фокусировать внимание на конкретных фактах, я стремился создавать небольшие примеры, объясняющие один или два вопроса, благодаря чему мне удалось избежать в книге длинных листингов, каждый из которых объяснял бы сразу множество вопросов. На своем опыте автора книг и преподавателя я уже убедился, что более короткие примеры приводят к лучшим результатам и существенно упрощают изучение материала.

Кроме того, следует отметить, что все примеры являются достаточно универсальными. Это позволит вам добавлять их непосредственно в свои собственные проекты, видоизменять и извлекать из них отдельные фрагменты по мере потребностей. Все примеры являются совершенно самостоятельными, что облегчает их использование.

Для кого написана книга

Данная книга адресована двум группам веб-разработчиков: тем, кто уже использует ASP.NET и хотел бы привнести в свои приложения возможности, которые предоставляет технология Ajax, и тем, кто в настоящее время использует в своей работе другие технологии, но заинтересован в освоении платформы ASP.NET AJAX. Кроме того, книга будет интересна программистам на JavaScript, которые хотели бы избежать головной боли при разработке программного кода, совместимого с разными типами броузеров. В этой книге используются языки программирования C# и JavaScript. Если вам необходимы начальные сведения об этих языках, можно порекомендовать обратиться к книгам «Learning C# 2005» (авторы: Джесс Либерти (Jesse Liberty) и Брайан Макдональд (Brian MacDonald)) и «Learning JavaScript» (автор Шелли Пауэрс (Shelley Powers)).

Как устроена книга

Глава 1 «ASP.NET AJAX, Ajax и ASP.NET» содержит обзор технологии Ajax и платформы ASP.NET AJAX, а также охватывает процедуру установки ASP.NET AJAX, описывает ее структуру и содержит первый пример.

Глава 2 «JavaScript» представляет собой краткое введение в JavaScript. Несмотря на то, что платформа ASP.NET AJAX стремится избавить программиста ASP.NET от ненужных функциональных подробностей, тем не менее некоторое знание JavaScript будет действительно необходимо настоящему специалисту в ASP.NET AJAX.

Глава 3 «Ajax» описывает технологии без лишней рекламы. Здесь вы узнаете о том, как работают внутренние механизмы Ajax; и все, что действительно необходимо знать, описывается менее чем на 20 страницах.

Глава 4 «Использование расширений JavaScript в ASP.NET AJAX» рассказывает о том, как ASP.NET AJAX обогащает функциональность

клиентского JavaScript за счет добавления объектно-ориентированных возможностей и переопределения некоторых классов платформы .NET для обеспечения возможности их использования на стороне клиента.

Глава 5 «Веб-службы» рассказывает о веб-службах, использующих XML. Хотя в платформе ASP.NET AJAX основной упор сделан на разработке клиентских приложений, в ней также имеются средства разработки веб-служб, исполняемых на стороне сервера. Сюда включаются возможности обработки ошибок и поддержка сеансов.

Глава 6 «UpdatePanel: обновление только части страницы» представит вашему вниманию элемент управления `UpdatePanel`, который позволяет обновлять отдельные части страницы без необходимости выполнять обновление всей страницы целиком. Это один из наиболее важных элементов платформы ASP.NET AJAX.

Глава 7 «Использование службы управления профилями в ASP.NET AJAX» описывает порядок организации доступа из JavaScript к прикладному интерфейсу ASP.NET 2.0 Profile API в ASP.NET AJAX.

Глава 8 «Использование службы аутентификации в ASP.NET AJAX» описывает функции JavaScript доступа к прикладному интерфейсу ASP.NET 2.0 Forms Authentication API.

Глава 9 «Локализация и глобализация приложений» охватывает тему создания веб-приложений, которые могут работать с различными языками и региональными настройками.

Глава 10 «Использование Control Toolkit» представляет собой введение в ASP.NET Control Toolkit – комплект серверных элементов управления, дополненных функциональными возможностями Ajax.

Глава 11 «Добавление анимационных эффектов в веб-страницу» описывает платформу для создания анимационных эффектов, входящую в состав ASP.NET AJAX Control Toolkit.

Глава 12 «Автодополнение ввода пользователя, борьба со спамом и не только» описывает особенности (возможно, спорные) ASP.NET AJAX Control Toolkit, демонстрирует разнообразные инструментальные средства, содержит некоторые советы и рассказывает о наиболее удачных приемах использования.

Глава 13 «Создание собственных элементов управления и помощь сообществу» рассказывает о том, как с помощью платформы Control Toolkit писать собственные элементы управления и как интегрировать их с существующими элементами в проекте.

Глава 14 «Клиентские элементы управления» описывает элементы управления, используемые на стороне клиента, которые поставляются в комплекте с ASP.NET AJAX Futures CTP. Они позволяют упростить обращение к HTML-элементам из сценариев JavaScript с использованием непротиворечивого прикладного интерфейса.

Глава 15 «Привязка и проверка данных» демонстрирует, как реализовать привязку данных к элементам управления на стороне клиента с помощью ASP.NET AJAX Futures CTP.

Глава 16 «Использование аспектов клиентского поведения и компонентов» демонстрирует аспекты поведения (behaviors), встроенные в ASP.NET AJAX, и порядок связывания их функциональности с клиентскими компонентами и элементами управления.

Глава 17 «Использование данных, размещенных на стороне сервера» объясняет, как организовать работу с базами данных. Платформа ASP.NET AJAX может связываться с источниками данных посредством специализированных веб-служб, облегчая доступ к данным без необходимости обновлять всю страницу. Кроме того, в ASP.NET AJAX имеются специализированные элементы управления, предназначенные для отображения данных на стороне клиента.

Глава 18 «Использование внешних веб-служб» поможет преодолеть ограничения общности происхождения, которые накладывает политика безопасности JavaScript, и обеспечить возможность обращения к удаленным веб-службам посредством моста на стороне сервера.

Глава 19 «Использование анимационных эффектов» демонстрирует некоторые возможности ASP.NET AJAX Futures CTP по созданию анимационных эффектов.

Глава 20 «Кнопки Назад/Вперед и закладки» рассказывает о возможных решениях двух наиболее раздражающих проблем, которые присущи Ajax-приложениям (связанных с нарушением стандартного поведения броузера).

Глава 21 «Веб-части» демонстрирует, что можно реализовать с помощью веб-частей (web parts) ASP.NET AJAX и что нельзя, включая, к примеру, реализацию механизма «перетащить-и-отпустить» (drag-and-drop), который будет работать в любом броузере.

Глава 22 «ASP.NET AJAX в комбинации с другими серверными технологиями» наглядно демонстрирует, что некоторые части Microsoft Ajax Library никак не связаны с ASP.NET 2.0. Приводится пример приложения на языке PHP, которое демонстрирует, как можно соединить эти два мира.

Приложение A «Отладка приложений ASP.NET AJAX» рассказывает, как выполняется поиск ошибок в приложениях ASP.NET AJAX и описывает некоторые инструментальные средства броузеров, которые желательно иметь каждому разработчику.

Приложение B «Справочник по XMLHttpRequest» содержит перечень методов и свойств объекта `XMLHttpRequest`.

Приложение C «Справочник DOM» описывает наиболее важные методы JavaScript объектной модели документа (DOM).

Приложение D «Справочник по ASP.NET AJAX» описывает наиболее важные методы, предоставляемые платформой ASP.NET AJAX.

Приложение E «Справочник по ScriptManager, UpdatePanel, UpdateProgress и Timer» документирует свойства этих четырех ключевых серверных элементов управления платформы ASP.NET AJAX.

Технические требования

Для опробования примеров из этой книги необходимо и достаточно иметь платформу ASP.NET 2.0, которая включена в состав свободно распространяемой платформы .NET. Однако, чтобы получить максимум от использования ASP.NET и ASP.NET AJAX, вам необходимо иметь какую-либо интегрированную среду разработки из предлагаемых компанией Microsoft. Среда Visual Web Developer 2005 Express Edition (VWD) распространяется бесплатно. Visual Studio 2005 (любые ее редакции) – пакет программного обеспечения, который обладает богатыми возможностями, распространяется на коммерческой основе. Любой из упомянутых инструментов прекрасно подойдет для опробования примеров из этой книги.

Типографские соглашения

В этой книге приняты следующие соглашения:

Обычный шрифт

Применяется для обозначения пунктов меню, кнопок и комбинаций клавиш (таких как Alt и Ctrl).

Курсив

Обозначает новые термины, адреса электронной почты и веб-сайтов, имена файлов и каталогов, расширений имен файлов, а также названия утилит Unix.

Моноширинный шрифт

Применяется для выделения команд, параметров, ключей, переменных, атрибутов, имен функций, типов, классов, пространств имен, методов, модулей, свойств, аргументов, значений, объектов, событий, обработчиков событий, тегов XML, тегов HTML, макроопределений, содержимого файлов и результатов работы команд.

Моноширинный жирный

Используется для выделения участков программного кода.

Моноширинный курсив

Используется для выделения элементов, которые необходимо заменить на реальные значения.

Этим значком обозначаются советы, предложения и примечания общего характера.

Этим значком обозначаются предупреждения и предостережения.

Использование программного кода примеров

Данная книга призвана оказать вам помощь в решении ваших задач. Вообще вы можете свободно использовать примеры программного кода из этой книги в своих приложениях и в документации. Вам не нужно обращаться в издательство за разрешением, если вы не собираетесь воспроизводить существенные части программного кода. Например, если вы разрабатываете программу и используете в ней несколько отрывков программного кода из книги, вам не нужно обращаться за разрешением. Однако в случае продажи или распространения компакт-дисков с примерами из этой книги вам *необходимо* получить разрешение от издательства O'Reilly. Для цитирования данной книги или примеров из нее, при ответе на вопросы не требуется получение разрешения. При включении существенных объемов программного кода примеров из этой книги в вашу документацию вам *необходимо* будет получить разрешение издательства.

Мы приветствуем, но не требуем добавлять ссылку на первоисточник при цитировании. Под ссылкой на первоисточник мы подразумеваем указание авторов, издательства и ISBN. Например: «Programming ASP.NET AJAX, by Christian Wenz. Copyright 2007 Christian Wenz, 978-0-596-51424-2».

При необходимости использовать значительный объем программного кода примеров из этой книги, за получением разрешения обращайтесь по адресу *permissions@oreilly.com*.

Отзывы и предложения

С вопросами и предложениями, касающимися этой книги, обращайтесь в издательство:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в Соединенных Штатах Америки или в Канаде)
707-829-0515 (международный)
707-829-0104 (факс)

Список опечаток, файлы с примерами и другую дополнительную информацию вы найдете на сайте книги:

http://www.oreilly.com/catalog/9780596514242

Свои комментарии и вопросы технического характера отправляйте по адресу:

bookquestions@oreilly.com

Дополнительную информацию о книгах, обсуждения, Центр ресурсов издательства O'Reilly вы найдете на сайте:

http://www.oreilly.com

Safari® Books Online

Если на обложке технической книги есть пиктограмма «Safari® Books Online», это означает, что книга доступна в Сети через O'Reilly Network Safari Bookshelf.

Safari предлагает намного лучшее решение, чем электронные книги. Это виртуальная библиотека, позволяющая без труда находить тысячи лучших технических книг, вырезать и вставлять примеры кода, загружать главы и находить быстрые ответы, когда требуется наиболее верная и свежая информация. Она свободно доступна по адресу *http://safari.oreilly.com*.

Благодарности (к книге «Programming Atlas»)

Работа над этой книгой оказалась непростой задачей. Нехватка документации, которая описывала бы изменения от версии к версии, и сложности в отладке программного кода JavaScript обусловили появление большого числа ошибок и необходимость проведения экспериментов. При том, что я работал с ASP.NET и JavaScript в течение длительного времени, Atlas мне пришлось изучать с самого начала. К счастью, разработчики Atlas оказались готовы оказать поддержку и открыты для контактов, в частности, на общедоступном форуме по адресу: *http://forums.asp.net/default.aspx?GroupID=34*.

Я благодарен большому коллективу технических редакторов, которые помогли мне оформить эту книгу и с которыми мы работали в режиме обратной связи. Далее в алфавитном порядке перечислены имена тех, кто помог мне спасти свою репутацию в нескольких случаях: Адонис Битар (Adonis Bitar), Арсен Еремин (Arsen Yeremin), Бертран Ле Рой (Bertrand Le Roy), Кристоф Уилл (Christoph Wille), Майк Поуп (Mike Pope) и Тобиас Хаузер (Tobias Hauser).

Кроме того, я весьма признателен своему редактору Джону Осборну (John Osborn), который руководил этим проектом. Он – единственный известный мне редактор, который постоянно проявлял неудовольствие, когда я отправлял подготовленный материал *до* условленного

крайнего срока. Но именно его превосходное руководство проектом позволило мне сосредоточиться на работе и оставаться при этом в графике (а иногда и опережать его).

Должен признать, что я не большой приверженец выражения личной благодарности членам семьи, мужьям/женам/невестам/партнерам, а также кошкам/собакам. (Единственное исключение – Ричард Хандхаузен (Richard Hundhausen), который когда-то выразил благодарность за то, что в районе, где он жил, не оказалось круглосуточной службы, выполняющей расторжение брака.) Тем не менее я хотел бы воспользоваться такой возможностью и поблагодарить моих родителей. Они очень поддерживали меня в период, когда я работал над своей первой книгой, и теперь, когда у меня за плечами уже более 50 книг, я хочу выразить им свою признательность. Мне неловко признаться, но они иногда умудряются находить ошибки, даже не зная описываемых технологий: однажды мой отец заметил, что в листинге содержится неравное число открывающих и закрывающих скобок. Спасибо вам, Мама и Папа. И еще – спасибо моим друзьям и семье, которые, кажется, относились ко мне с терпением, когда я подолгу работал над книгами или уезжал на еще какую-нибудь конференцию.

Благодарности (к книге «Programming ASP.NET AJAX»)

Иногда оказывается так, что вы просто неудачно выбрали время. Спустя приблизительно две недели после выхода в свет первого издания книги под названием «Programming Atlas» компания Microsoft изменила название платформы на ASP.NET AJAX. Неудачность выбора времени обусловлена не только изменением названия – в Microsoft также весьма радикально изменили внутреннюю архитектуру платформы. Как следствие этого – все примеры программного кода для Atlas оказались неработоспособными в ASP.NET AJAX. Следует признать, что переписать большую часть примеров не составило большого труда, но некоторые функциональные возможности платформы были утрачены или изменены до неузнаваемости.

Вследствие этого данное издание книги полностью отличается от предыдущего. Структура книги претерпела существенные изменения: в нее было добавлено большое число новых глав, какой-то материал был добавлен, какой-то пришлось выбросить, а некоторые главы были полностью переписаны. Таким образом, хотя технически это второе издание – это более или менее новая книга. Однако если у вас имеется программный код, основанный на особенностях платформы Atlas, не волнуйтесь: в нескольких главах этой книги вы найдете рекомендации по переносу устаревшего программного кода на новую версию платформы.

Я выражаю благодарность моему редактору Джону Осборну (John Osborn), который руководил проектом, постоянно высказывая новые

идеи, касающиеся книги. Главным техническим редактором был Майк Поуп (Mike Pope) – эту роль он взял на себя еще при подготовке первого издания книги. Он не только удалял мои остроты в адрес Microsoft (вздох), но и поставлял бесконечное число предложений, комментариев и идей для этого издания. Это был тяжелый труд как для него (поиск ошибок), так и для меня (их исправление), но, на мой взгляд, получившийся результат стоил затраченных усилий. Спасибо вам обоим за ваши усилия, направленные на то, чтобы сделать второе издание книги еще лучше.

Кроме того, спасибо всем читателям первого издания, приславшим огромное число отзывов и предложений. Спасибо всем разработчикам, выучившимся с помощью этой книги, кто также высказывал мне свои замечания.

I

Основы

Глава 1. ASP.NET AJAX, Ajax и ASP.NET

Глава 2. JavaScript

Глава 3. Ajax

1

ASP.NET AJAX, Ajax и ASP.NET

В этой книге рассказывается о платформе ASP.NET AJAX (в предварительных версиях известной как «Atlas») – комплекте новых технологий компании Microsoft, которые дают возможность веб-разработчикам, и разработчикам на ASP.NET 2.0 в частности, упростить создание веб-сайтов со страницами, использующими технологии Ajax. Страницы, написанные в стиле Ajax, обеспечивают более богатый пользовательский интерфейс. Такие страницы более отзывчивы, потому что в состоянии немедленно откликаться на действия пользователя и достаточно быстро взаимодействовать с сервером. Кроме того, платформа ASP.NET AJAX включает в себя инструментальные средства для создания *гибридных* (*mashups*) веб-приложений, которые могут отображать содержимое, собранное с нескольких сайтов, – как правило, с использованием прикладных интерфейсов веб-служб, реализованных третьими фирмами. На протяжении всей книги мы будем исследовать эти возможности. Данная глава представляет введение в платформу ASP.NET AJAX, дает краткий обзор технологии, лежащей в ее основе, и архитектурное представление о том, как она функционирует.

ASP.NET AJAX и Ajax

Платформа ASP.NET AJAX расширяет набор технологий, поддерживаемых броузером, включая асинхронный JavaScript и XML (Asynchronous JavaScript and XML – Ajax). Технология Ajax произвела довольно много шума в последнее время (в предисловии приводятся некоторые соображения по этому поводу), поскольку она позволяет приблизить веб-приложения по функциональности и пользовательскому интерфейсу к обычным настольным приложениям.

Главная концепция, которая лежит в основе Ajax, заключается в том, чтобы дать веб-страницам возможность производить HTTP-запросы

в фоновом режиме, или *асинхронно*, без необходимости обновлять содержимое страницы целиком (или, в терминологии ASP.NET, без отправки заполненной формы и получения в ответ новой страницы). Кроме того, применение технологий Ajax позволяет создавать более отзывчивый пользовательский интерфейс с привлечением всей мощи функциональных возможностей, поддерживаемых броузером, таких как JavaScript, объектная модель документа (Document Object Model – DOM) и каскадные таблицы стилей (Cascading Style Sheets – CSS). Приложение Google Suggest (*http://www.google.com/webhp?complete=1&hl=en*) наглядно демонстрирует, как страница с поддержкой Ajax может предоставлять пользователю подсказки по мере ввода текста (этот прием известен также как *автодополнение*). Еще один пример подобных приложений – Microsoft Virtual Earth (*http://www.virtualearth.com/*).

Платформа ASP.NET AJAX поможет вам в создании подобных приложений, обладающих поддержкой Ajax, исполняющихся броузером (клиентом). Для работы с технологиями Ajax и ASP.NET AJAX на стороне клиента вам потребуется четкое понимание базовых технологий Ajax. Для создания веб-страниц с поддержкой Ajax, в которых используются сценарии, исполняемые броузером, необходимо знание и понимание языка программирования JavaScript, DOM и объекта `XMLHttpRequest`, который выполняет запросы от клиента к серверу. Знание языка разметки XML и языка описания преобразований XSLT желательно, но не обязательно. (В книге они подробно не рассматриваются.)

Введение в основы языка программирования JavaScript дается в главе 2. В главе 3 более подробно обсуждаются прочие технологии Ajax. Для понимания примера, который приводится в конце этой главы (раздел «Первый пример ASP.NET AJAX: Hello User»), достаточно обладать базовыми представлениями о технологии Ajax. Ваши знания будут углубляться по мере продвижения вперед.

Создание приложений на основе технологии Ajax без использования какой-либо оболочки, такой как ASP.NET AJAX, может оказаться делом достаточно сложным. В какой-то момент вы вдруг обнаружите, что вам снова и снова приходится писать один и тот же программный код, выполняющий такие действия, как отображение данных, полученных от сервера по запросу, привязка данных к элементам управления или взаимодействие с веб-службами. Вам также придется столкнуться с необходимостью писать код, который учитывал бы различия реализации DOM в разных броузерах. Одна из целей, которые преследует ASP.NET AJAX, состоит в том, чтобы уменьшить или вообще ликвидировать необходимость написания избыточного программного кода и предоставить в распоряжение разработчика, занимающегося разработкой клиентских приложений, весь опыт, накопленный разработчиками платформы ASP.NET 2.0. Еще одна цель состоит в том, чтобы усилить JavaScript некоторыми преимуществами, которые дают объектно-ориентированное программирование (ООП) и использование таких платформ, как .NET. Платформа ASP.NET AJAX включает в себя

библиотеки клиентских сценариев, которые дают программисту JavaScript/DOM/CSS следующие преимущества:

Совместимость с разными типами броузеров

Обеспечивает возможность исполнения сценариев ASP.NET AJAX в большинстве броузеров и ликвидирует необходимость создавать вручную сценарии для каждого броузера, поддерживаемого приложением. (Тем не менее создание сценариев, учитывающих особенности отдельных броузеров, неизбежно, в чем вы убедитесь в главе 3.)

Основные службы

Предоставляют расширения JavaScript, которые позволяют использовать объектно-ориентированный стиль программирования, включая поддержку классов, пространств имен, обработку событий, наследование и сериализацию (serialization) объектов с учетом форматов JSON (JavaScript Object Notation – формат представления объектов JavaScript) и XML. Наиболее важные из этих расширений будут рассматриваться в главе 4.

Библиотека базовых классов

Эта библиотека содержит ряд .NET-подобных компонентов, таких как построители строк и таймеры. О них будет говориться в главе 4.

Сценарные компоненты и элементы управления

Представляют собой версии ASP.NET AJAX стандартных элементов управления HTML, которые дополнены такими возможностями, как привязка данных, предопределенное поведение (например, поддержка функциональности «перетащить-и-отпустить») и тесная интеграция с клиентскими библиотеками ASP.NET AJAX. Вы можете использовать эти элементы управления и компоненты непосредственно или воспользоваться новой декларативной разметкой, получившей название *xml-script*, которая будет рассматриваться в нескольких главах этой книги. Если вы уже знакомы с синтаксисом разметки ASP.NET, значит, вы понимаете (в общих чертах) взаимосвязь между элементами управления HTML, абстрактными программируемыми версиями этих элементов управления и декларативным синтаксисом.

ASP.NET AJAX и ASP.NET

Платформа ASP.NET AJAX оказывает неоценимую помощь разработчику клиентских сценариев при создании Ajax-приложений, но ее возможности не ограничиваются одним только JavaScript и способностью производить асинхронные обращения к серверу. Эта платформа создавалась командой разработчиков ASP.NET, поэтому совершенно неудивительно, что одна из отличительных ее особенностей – это серверная инфраструктура, интегрированная с (и требующая наличия) ASP.NET 2.0.

Как и ASP.NET, платформа ASP.NET AJAX проектировалась для предоставления функциональных возможностей (в данном случае – в сочетании с преимуществами Ajax) без необходимости владения технологиями Ajax. Платформа ASP.NET AJAX в состоянии обеспечивать функциональность Ajax без участия программиста, точно так же, как платформа ASP.NET обеспечивает такие функциональные возможности HTTP, как отправка данных формы на сервер (postback), управление состоянием, а клиентский сценарий только требует, чтобы в ASP.NET это «просто работало».

Кроме всего прочего на стороне сервера платформа ASP.NET AJAX работает как часть ASP.NET, получая в свое распоряжение преимущества от наследования функциональных возможностей. Элементы управления ASP.NET AJAX могут взаимодействовать с элементами управления и компонентами ASP.NET и участвовать в жизненном цикле страницы. Она имеет возможность обращаться к таким службам ASP.NET 2.0, как механизмы управления сеансами, аутентификацией и профилями, позволяя вам использовать эти функциональные возможности на стороне клиента. Наконец, с помощью ASP.NET AJAX и ASP.NET можно выполнять обращения к специализированным веб-службам.

Ниже перечислены некоторые из ключевых элементов серверной платформы ASP.NET AJAX:

Серверные элементы управления ASP.NET AJAX

ASP.NET AJAX предоставляет серверные элементы управления, напоминающие аналогичные элементы, имеющиеся в ASP.NET 2.0, но в отличие от последних их функциональность доступна из клиентской платформы ASP.NET AJAX. Наиболее фундаментальными для приложений ASP.NET AJAX являются следующие два элемента управления: `ScriptManager`, который будет обсуждаться далее в этой главе (раздел «Элемент управления ScriptManager»), и `UpdatePanel`, который рассматривается в главе 6.

Службы ASP.NET AJAX и ASP.NET

Предоставляет некоторые прикладные службы ASP.NET 2.0, доступные непосредственно клиентским сценариям ASP.NET AJAX, включая службы управления профилями, персонализацией, аутентификацией, членством в группах и региональными настройками по умолчанию. Вполне оправданно ожидать, что число служб ASP.NET, доступных приложениям ASP.NET AJAX, будет увеличиваться с выходом новых версий ASP.NET AJAX.

Библиотека Microsoft Ajax Library

Эта библиотека содержит исключительно программный код JavaScript, который никак не связан с ASP.NET. Благодаря этому данная библиотека может использоваться и без ASP.NET, о чем мы поговорим в главе 22.

Несомненно, ASP.NET AJAX займет свое законное место ключевого компонента в следующих версиях ASP.NET и получит полную поддержку дизайнеров, технологии IntelliSense и инструментальных средств отладки в будущих версиях Visual Studio.

Пакеты ASP.NET AJAX

На домашней странице проекта ASP.NET AJAX (*http://ajax.asp.net/*) представлен перечень пакетов, каждый из которых предназначен для решения определенных задач:

ASP.NET AJAX Extensions

Известен также под названием «ASP.NET AJAX Core». Это «главный» пакет ASP.NET AJAX. Он целиком поддерживается компанией Microsoft и содержит инфраструктуру ASP.NET AJAX (обсуждается во второй части книги).

ASP.NET AJAX Control Toolkit

Этот пакет содержит обширную коллекцию серверных компонентов, предоставляющих легкий и удобный доступ к потрясающим функциональным возможностям Ajax. Пакет Control Toolkit – проект с открытыми исходными текстами, хотя компания Microsoft и остается руководителем проекта, что гарантирует его высокое качество. Тем не менее официально Microsoft не осуществляет поддержку элементов пакета. Возможности пакета Control Toolkit будут исследованы третьей части книги.

ASP.NET AJAX Futures Release

Этот пакет содержит экспериментальные реализации функциональных возможностей, которые могут войти (или не войти) в состав ASP.NET и ASP.NET AJAX. Кроме того, в пакет Futures Release помещаются редко используемые элементы, которые первоначально входили в состав предварительных версий ASP.NET AJAX. Версия CTP (Community Technology Preview – предварительная версия, доступная для загрузки) обновляется чаще, чем основной пакет. Но эта версия пакета официально не поддерживается, поэтому вы можете использовать его только на свой страх и риск. В четвертой части книги рассматривается пакет Futures Release, наиболее свежий на момент работы над книгой, а именно версия ASP.NET Futures от июля 2007 года. Кроме того, в мае 2007 года ASP.NET AJAX Futures CTP входил в состав пакета ASP.NET Futures CTP, который включает в себя новые интересные (и не поддерживаемые) возможности для работы с классической версией ASP.NET.

Microsoft Ajax Library

Упоминавшаяся ранее библиотека JavaScript, которая будет рассматриваться в пятой части книги.

Что самое удивительное – компания Microsoft предоставляет полный комплект исходных текстов ASP.NET AJAX. Загрузить их можно по адресу: *http://ajax.asp.net/*.

В этой главе будет рассказано и показано, как установить основной пакет ASP.NET AJAX Extensions (а также мы вкратце коснемся пакета Futures CTP), а затем во второй части книги этот пакет будет рассмотрен более подробно. Остальные пакеты будут рассматриваться далее в соответствующих частях.

Подготовка и установка ASP.NET AJAX

Лучший способ осознать мощь ASP.NET AJAX – начать использовать его. Все, что вам потребуется для разработки приложений, – это броузер с поддержкой JavaScript на стороне клиента и веб-сервер с поддержкой ASP.NET 2.0. Для начала вполне будет достаточно обычного текстового редактора. Однако, по мере роста сложности приложений, наличие интегрированной среды разработки с такими функциональными возможностями, как IntelliSense, автодополнение кода, управление проектами, отладчик и WYSIWYG (What You See Is What You Get – что видишь, то и получишь) поможет сэкономить массу времени. В мире ASP.NET 2.0 для этих целей обычно используется редактор, входящий в состав Visual Studio 2005.

Установка интегрированной среды разработки

Следует иметь в виду, что хотя в общем случае наилучшим выбором является приобретение полной версии Visual Studio 2005, тем не менее бесплатно распространяемая Express-версия Visual Studio 2005, предназначенная для разработки веб-приложений – Microsoft Visual Web Developer 2005 Express Edition, – также полностью поддерживает ASP.NET AJAX.

Для упрощения в этой книге иногда вместо полного названия Visual Web Developer будет использоваться аббревиатура VWD. Под этой аббревиатурой будут подразумеваться как Visual Web Developer 2005 Express Edition, так и полная версия Visual Studio 2005. Компонент разработки веб-приложений, входящий в состав Visual Studio 2005 также называется Visual Web Developer (вы можете увидеть его в процессе установки Visual Studio), поэтому название VWD можно считать универсальным обозначением инструментальных средств разработки приложений для ASP.NET 2.0 в интегрированной среде разработки компании Microsoft.

Если у вас еще нет интегрированной среды разработки, установите Visual Studio 2005 или Visual Web Developer Express Edition. Если вы выбрали второй вариант, откройте в броузере страницу *http://msdn.microsoft.com/vstudio/express/vwd/download*, где вы найдете мастер установ-

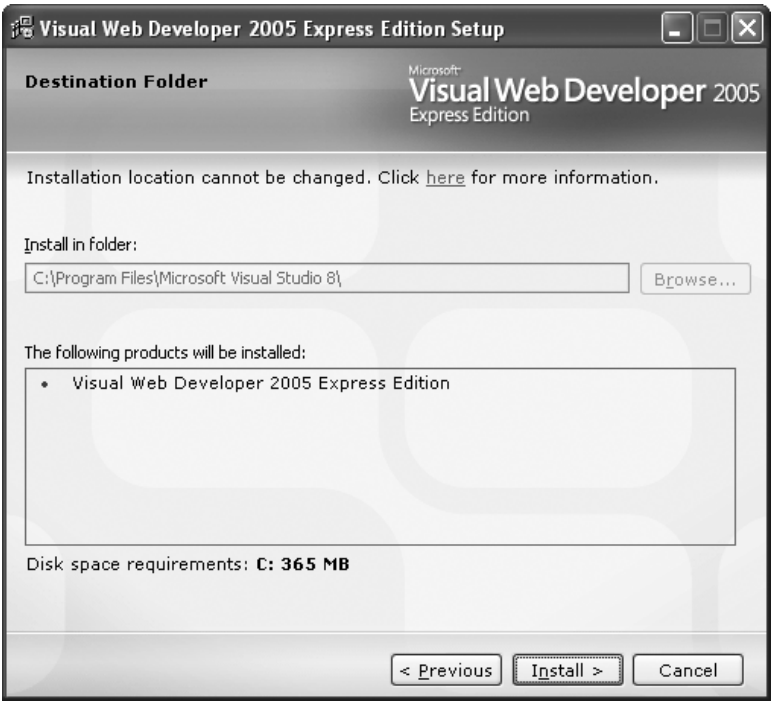


Рис. 1.1. *Установка Visual Web Developer Express Edition*

ки по сети, который не только загрузит и установит VWD (рис. 1.1), но и установит при необходимости .NET Framework 2.0.

Если мастер установки по сети не сможет работать на вашей машине (например, из-за ограничений на соединение с узлами Интернета из корпоративной сети или из-за очень медленного соединения), на этом же сайте вы найдете образы компакт-дисков в формате ISO и IMG, содержащие все необходимое для установки Visual Web Developer (*http://msdn.microsoft.com/vstudio/express/support/install*), которые можно загрузить на любой другой компьютер, на котором имеется лучшее соединение с Интернетом, а затем записать их на компакт-диски.

Установка ASP.NET AJAX

Независимо от того, какая версия VWD используется, ASP.NET AJAX интегрируется непосредственно в среду разработки. На домашней странице ASP.NET AJAX (*http://ajax.asp.net*) вы найдете ссылку на дистрибутив ASP.NET AJAX в формате MSI с именем *ASPAJAXExtSetup.msi*. Ищите ссылку с текстом «Microsoft ASP.NET 2.0 AJAX Extensions 1.0».

Перед запуском пакета установки необходимо удалить все предыдущие версии ASP.NET AJAX, которые могли быть установлены в системе.

Мастер установки *.msi* задаст всего несколько вопросов. После того как будут приняты условия лицензионного соглашения ASP.NET AJAX, начнется процесс установки, как показано на рис. 1.2.

После установки в VWD или в Visual Studio появится новый шаблон приложений – ASP.NET AJAX Web Site. Описанный способ установки ASP.NET AJAX является предпочтительным, так как в этом случае все необходимые файлы копируются в соответствующие каталоги (рис. 1.3). Кроме того, при таком способе установки сборка ASP.NET

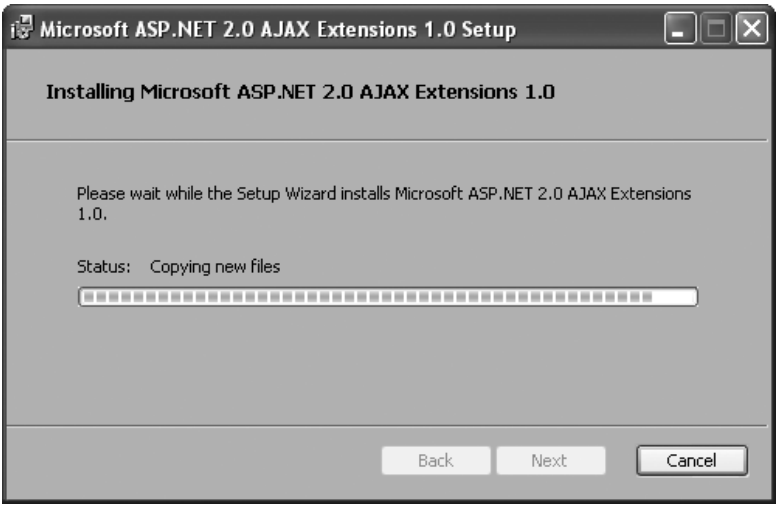


Рис. 1.2. *Установка шаблона*

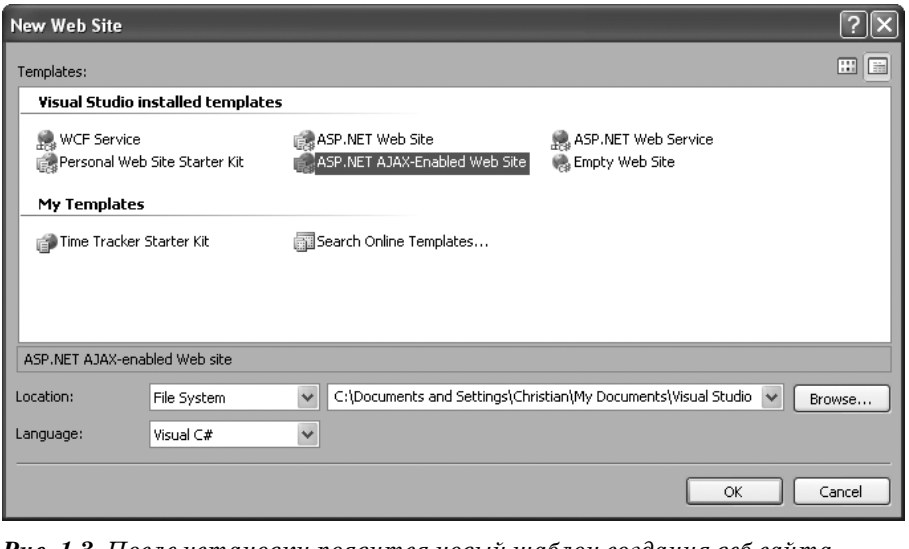


Рис. 1.3. *После установки появится новый шаблон создания веб-сайта*

AJAX будет помещена в глобальный кэш сборок (Global Assembly Cache – GAC), что автоматически сделает ее доступной для всех приложений ASP.NET.

ASP.NET AJAX также работает в готовящихся к выходу Visual Studio 2008 и Visual Web Developer 2008 Express Edition. В этих средах разработки при создании нового веб-сайта просто укажите, что должна использоваться платформа .NET Framework 3.5 (рис. 1.4); вам не придется устанавливать ASP.NET AJAX, поскольку она уже включена в состав .NET Framework 3.5. Для версий .NET ниже 3.5 придется отдельно устанавливать ASP.NET AJAX. Если вы работаете и с Visual Studio 2005, и с Visual Studio 2008 или вы хотите открыть в Visual Studio 2008 проект ASP.NET AJAX, который был создан в Visual Studio 2005, перейдите к разделу «Для дополнительного чтения» в конце этой главы; там вы найдете несколько важных советов по данному вопросу.

На сайте проекта ASP.NET AJAX можно найти дополнительную информацию и программное обеспечение. Ниже перечислена лишь часть того, что можно найти:

- Документация, которая ознакомит вас с некоторыми аспектами ASP.NET AJAX (*http://ajax.asp.net/docs*). Кроме того, имеется возможность загрузить и установить документацию на локальном компьютере (*AspNet_AJAX_Documentation.zip*).
- Примеры ASP.NET AJAX
- Ссылки на другие пакеты ASP.NET AJAX: ASP.NET AJAX Control Toolkit, Microsoft Ajax Library, ASP.NET AJAX Futures Release и исходные тексты платформы ASP.NET AJAX.

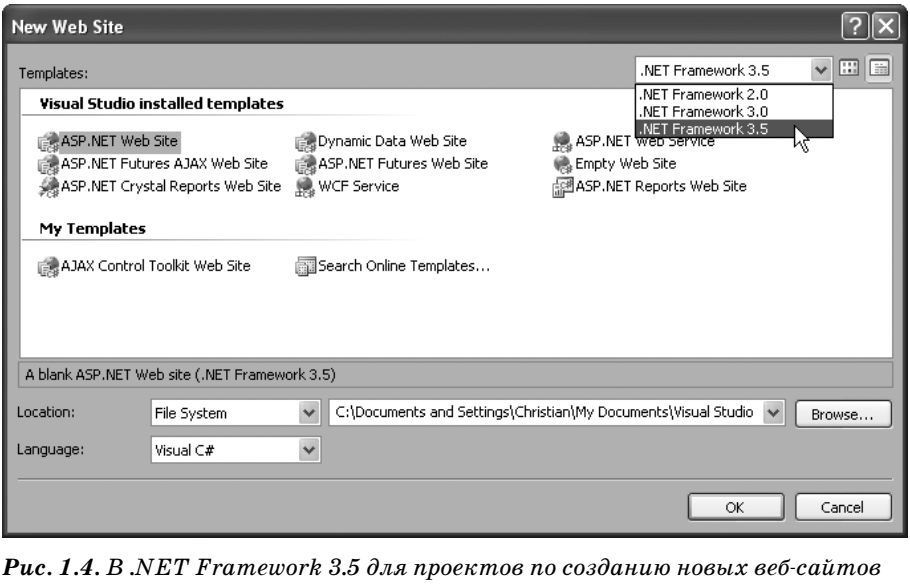


Рис. 1.4. *В .NET Framework 3.5 для проектов по созданию новых веб-сайтов автоматически используется ASP.NET AJAX*

Установка испытательной базы данных

Примеры в этой книге написаны в предположении, что в качестве сервера баз данных используется SQL Server 2005 Express Edition (хотя примеры легко могут быть адаптированы для использования любых других источников данных). Если у вас на компьютере установлен Microsoft SQL Server, то он также может быть использован. В противном случае загрузите и установите SQL Server Express (его можно загрузить бесплатно).

Чтобы максимально упростить процесс установки и развертывания, во всех примерах из этой книги, работающих с базой данных, используется типовая база данных *AdventureWorks* компании Microsoft. Также предполагается, что *AdventureWorks* была установлена в локальный сервер SQL Server 205 Express Edition и доступна с параметрами аутентификации Windows в *(local)\SQLEXPRESS*.

Возможно, вам придется изменить в примерах путь к SQL Express, если он отличается от указанного.

В зависимости от используемой версии SQL Server требуемая версия базы данных AdventureWorks доступна для загрузки по следующему адресу:

SQL Server 2005

http://www.microsoft.com/downloads/details.aspx?familyid=E719ECF7-9F46-4312-AF89-6AD8702E4E6E&displaylang=en

SQL Server 2005 Express Edition

http://www.microsoft.com/downloads/details.aspx?familyid=9697AAAA-AD4B-416E-87A4-A8B154F92787&displaylang=en

Выберите нужную версию, загрузите дистрибутив и запустите мастер установки. По окончании установки вам будет необходимо подключить файл *AdventureWorks_Data.mdf* (в подкаталоге *Data*) к вашему серверу баз данных SQL Server 2005. Проще всего это сделать с помощью Microsoft SQL Server Management Studio Express (SSMSE) – бесплатного инструмента с графическим интерфейсом для администрирования SQL Server 2005 Express Edition. SSMSE доступен для загрузки в двух версиях, 32-битной и 64-битной, по адресу: *http://www.microsoft.com/downloads/details.aspx?FamilyID=c243a5ae-4bd1-4e3d-94b8-5a0f62bf7796&DisplayLang=en*.

По окончании установки и настройки базы данных *AdventureWorks* откройте меню Пуск (Start) и запустите SQL Server. Введите в диалоговом окне информацию для вашего сервера SQL Server 2005 Express Edition, как показано на рис. 1.5. По умолчанию сервер доступен по имени *(local)\SQLEXPRESS* или *ИмяКомпьютера\SQLEXPRESS* с использованием типа аутентификации Windows Authentication.

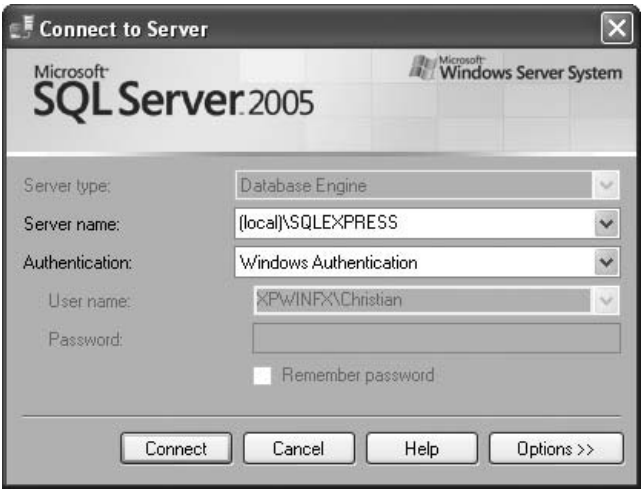


Рис. 1.5. *Окно регистрации SSMSE*

Далее, в окне SSMSE щелкните правой кнопкой мыши на папке с базами данных и выберите пункт контекстного меню Attach (присоединить). В открывшемся диалоге (рис. 1.6) щелкните на кнопке Add (добавить) и выберите файл *AdventureWorks_Data.mdf*. Дважды нажмите OK. После этого база данных *AdventureWorks* будет подключена к вашему серверу SQL Server 2005 Express Edition.

Установка Futures CTP

Процесс установки ASP.NET AJAX Futures CTP (часть более обширного пакета ASP.NET Futures CTP) мало чем отличается от установки ASP.NET AJAX Extensions. При установке в Visual Studio и Visual Web Developer также будет установлен шаблон веб-сайта. Шаблон имеет название *ASP.NET Futures AJAX Web Site* (не следует путать его с *ASP.NET Futures Web Site*, который служит для создания сайтов без поддержки Ajax). Прежде чем устанавливать пакет CTP, необходимо сначала установить пакет ASP.NET AJAX Extensions.

В примерах, представленных в этой книге, приложения ASP.NET AJAX Extensions получают стандартное имя `AJAXEnabledWebSite1`, а приложения CTP получают имя `AJAXFuturesEnabledWebSite1`. Приложения первого типа по умолчанию работают с портом 1234, а второго – с портом 1236. Порт с номером 1235 зарезервирован для примеров приложений ASP.NET AJAX Control Toolkit. Разумеется, вы можете определить иной номер порта.

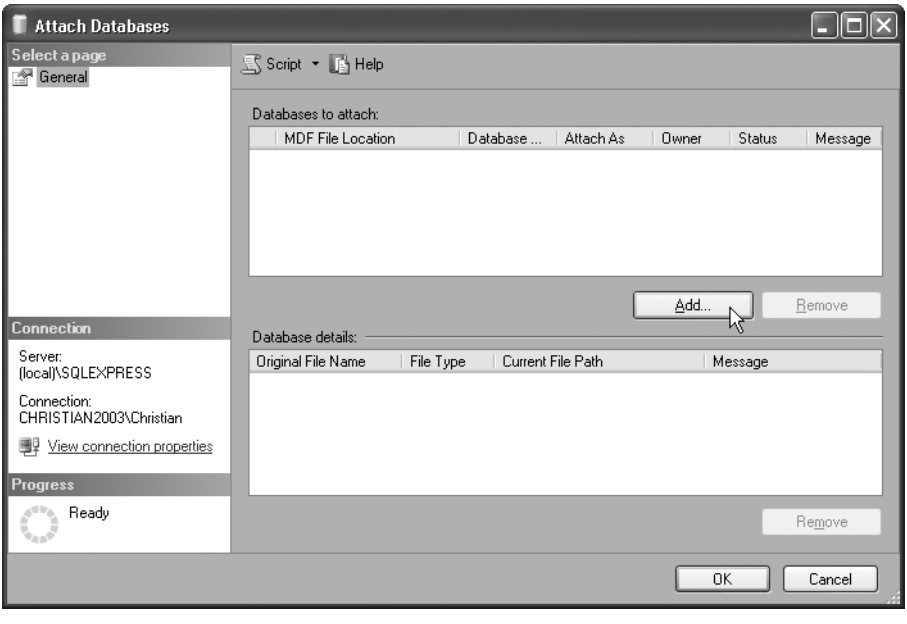


Рис. 1.6. *Подключение MDF-файла к серверу SQL Server 2005 Express Edition*

Структура и архитектура ASP.NET AJAX

Теперь можно приступать к работе с ASP.NET AJAX. Запустите VWD и создайте с помощью шаблона ASP.NET AJAX новый проект веб-сайта ASP.NET. Если теперь посмотреть на окно Solution Explorer, вы увидите проект обычного веб-сайта ASP.NET. Не удивляйтесь, это мастер установки поместил сборку ASP.NET AJAX непосредственно в GAC. Единственное, что его отличает, – это наличие файла *Web.config*, который содержит настройки, необходимые для работы ASP.NET AJAX.

ASP.NET AJAX состоит из серверных и клиентских компонентов. Есть возможность использовать только серверные или только клиентские компоненты. Единственное исключение: любое приложение ASP.NET AJAX требует наличие серверного элемента управления `ScriptManager`, который будет рассматриваться ниже, в этой же главе. Как правило, в приложениях используются и серверные, и клиентские компоненты.

Назначение клиентских и серверных компонентов в проекте ASP.NET AJAX станет понятнее, когда мы ближе познакомимся с тем, как Ajax-приложения используют в своей работе объект `XMLHttpRequest`.

На рис. 1.7 показана базовая структура ASP.NET AJAX. При стандартном подходе в ответ на каждый запрос со стороны клиента сервер отсылает одну веб-страницу, но при наличии поддержки Ajax веб-страницы могут производить непрерывный обмен данными с сервером. Платформа ASP.NET AJAX осуществляет поддержку такого обмена

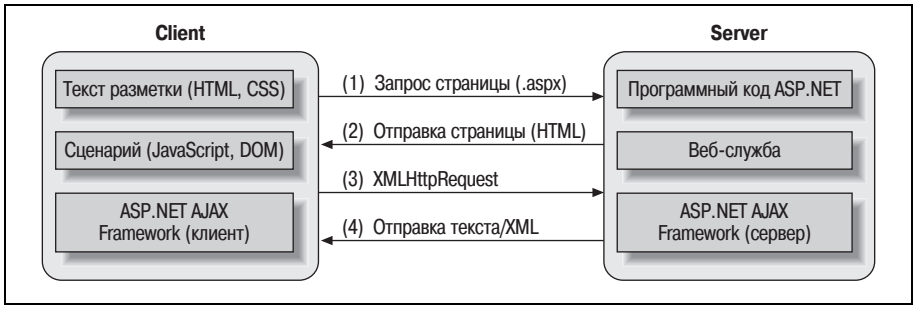


Рис. 1.7. *Жизненный цикл веб-страницы ASP.NET AJAX*

с обеих сторон соединения. Библиотеки с программным кодом сценариев (которые динамически загружаются компонентом `ScriptControl`, как вы вскоре увидите) облегчают взаимодействие броузера и веб-сервера и упрощают программирование клиентской части. Программный код, реализованный в сборке ASP.NET AJAX на стороне сервера, выполняет прием и обработку запросов от объекта `XMLHttpRequest` и, кроме того, обеспечивает некоторые дополнительные удобные серверные элементы управления, которые будут описаны далее в книге. В результате компоненты на стороне клиента и на стороне сервера получают возможность обмениваться данными, почти не требуя усилий программиста.

Платформа ASP.NET AJAX на стороне клиента (нижний уровень клиентского компонента на рис. 1.7) передается клиентскому броузеру при первом обращении к странице с поддержкой ASP.NET AJAX (шаги 1 и 2 на рис. 1.7). После этого из той же страницы Ajax-приложения можно производить запросы к серверу по протоколу HTTP и получать в ответ текст или XML (шаги 3 и 4 на рис. 1.7). Веб-страницы ASP.NET для выполнения различных задач могут производить как полноценную отправку форм с данными, так и асинхронные запросы к серверу.

Отдельные компоненты ASP.NET AJAX, работающие на стороне сервера и на стороне клиента, подробно будут рассматриваться на протяжении всей книги. Однако вы должны постоянно держать в уме эту базовую структуру, включающую в себя обмен данными между клиентом и сервером. Чем реже выполняется запрос полных страниц, тем лучше, по крайней мере, потому, что за счет этого можно избежать обновления страницы целиком.

Первый пример ASP.NET AJAX: Hello User

Чтобы проверить, насколько успешно прошла установка ASP.NET AJAX, и увидеть платформу в действии, закончим эту главу примером создания небольшого приложения. В этом примере страница ожидает ввода имени пользователя, отправляет его серверу (в фоновом режиме, с помощью объекта `XMLHttpRequest`), получает обратно имя, дополненное некоторым текстом, и выводит его перед пользователем. Данный

пример наглядно демонстрирует, насколько просто создаются приложения, обладающие функциональными возможностями ASP.NET AJAX. В последующих главах принцип действия внутренних механизмов будет описан более подробно.

В VWD создайте новый проект веб-сайта из шаблона ASP.NET AJAX. Затем в корневом каталоге веб-сайта создайте новую веб-службу (с помощью файла-шаблона веб-службы) с именем *WebService.asmx*. В файле *.asmx* реализуйте простой веб-метод, который принимает единственный строковый параметр, скопировав в файл программный код из примера 1.1. Обратите внимание на обязательный атрибут `[ScriptServices]`, который фактически определяется платформой ASP.NET AJAX (в пространстве имен `System.Web.Script.Services`).

Пример 1.1. Веб-служба

```
WebService.asmx

<%@ WebService Language="C#" Class="WebService" %>

using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://hauser-wenz.de/AspNetAJAX/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.Web.Script.Services.ScriptService]
public class WebService : System.Web.Services.WebService {
    [WebMethod]
    public string sayHello(string name) {
    return "Hello " + name + ", says the server!";
    }
}
```

Теперь попробуйте обратиться к этой веб-службе из веб-броузера, добавив символы */js* в конец адреса URL. При обращении по такому URL, как показано на рис. 1.8, возвращается программный код JavaScript, который фактически представляет собой реализацию класса-посредника JavaScript (подробное описание см. в главе 5). Самое важное здесь заключается в том, что этот программный код создает переменную `WebService`, которая обеспечивает взаимодействие с веб-службой.

Можно заметить, что шаблон ASP.NET AJAX уже создал файл *Default.aspx*, содержимое которого мы рассмотрим ниже. Следующий фрагмент – это текст, содержащийся в данном файле:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

```
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:ScriptManager ID="ScriptManager1" runat="server" />
        <div>
        </div>
    </form>
</body>
</html>
```

Первое, на что следует обратить внимание, – это новый элемент управления: `<asp:ScriptManager>`. Этот элемент управления является основой любой веб-страницы с поддержкой ASP.NET AJAX. Элемент ScriptManager будет рассмотрен гораздо подробнее ниже, в этой же главе (раздел «Элемент управления ScriptManager»).

Откройте файл *Default.aspx* в текстовом редакторе и добавьте в элемент `ScriptManager` ссылку на веб-службу, как показано ниже. При запуске страницы благодаря этой ссылке будет сгенерирован объект-посредник JavaScript, с помощью которого страница будет способна использовать программный код, динамически сгенерированный программным кодом веб-службы:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Services>
    <asp:ServiceReference Path="WebService.asmx" />
  </Services>
</asp:ScriptManager>
```

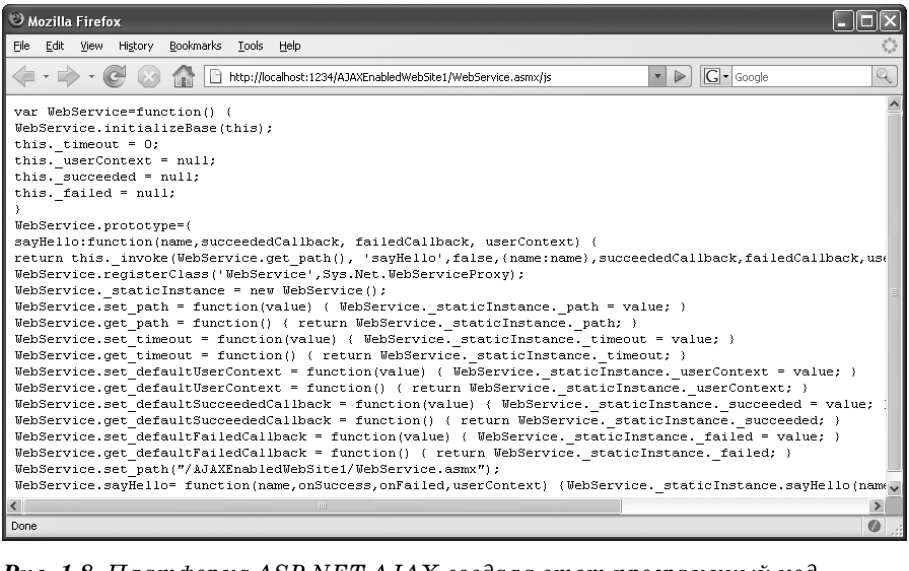


Рис. 1.8. *Платформа ASP.NET AJAX создала этот программный код JavaScript автоматически*

Теперь необходимо вставить в страницу несколько HTML-элементов. Добавьте текстовое поле ввода и кнопку в существующий элемент `<form>` (внутри элемента `<div>`, если вы собираетесь придерживаться требований стандарта XHTML):

```
<input type="text" id="name" name="name" />
<input type="button" value="Call Service" onclick="callService(this.form);" />
```

Обработчик события кнопки вызывает функцию JavaScript с именем `callService()` и передает ей ссылку на текущую форму. Метод `callService()` — это та самая точка в сценарии, где происходит обращение к веб-службе. Чтобы вызвать метод веб-службы `sayHello()`, сценарий должен воспользоваться объектом-посредником, который доступен через автоматически созданную переменную с именем `WebService`. (Имя `WebService` соответствует имени класса веб-службы, созданного ранее.)

Метод `sayHello()` ожидает получить не только строку, но и три ссылки на функции-обработчики: первая вызывается в случае успешного исполнения веб-службы (`callComplete`), вторая при появлении ошибки (`callError`) и третья по истечении предельного времени ожидания ответа (в данном примере мы используем только первые две функции).

Затем поместите следующий программный код в клиентский элемент `<script>` на странице:

```
function callService(f) {
  WebService.sayHello(
    f.elements["name"].value,
    callComplete,
    callError);
}
```

В заключение необходимо написать две функции, которые будут обрабатывать события `callComplete` и `callError`. Для этого добавьте следующий фрагмент в клиентский блок сценария, который только что был создан:

```
function callComplete(result) {
  window.alert(result);
}
function callError(result) {
  window.alert("Error! " + result);
}
```

В примере 1.2 приводится содержимое файла *Default.aspx* целиком.

Пример 1.2. Простая страница ASP.NET AJAX, которая обращается к веб-службе

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

```
    <title>ASP.NET AJAX</title>

    <script language="Javascript" type="text/javascript">
    function callService(f) {
      WebService.sayHello(
        f.elements["name"].value,
        callComplete,
        callError);
    }

    function callComplete(result) {
      window.alert(result);
    }

    function callError(result) {
      window.alert("Error! " + result);
    }

    </script>

  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager
        ID="ScriptManager1"
        runat="server">
        <Services>
          <asp:ServiceReference Path="WebService.asmx" />
        </Services>
      </asp:ScriptManager>
      <div>
        <input type="text" id="name" name="name" />
        <input type="button" value="Call Service"
               onclick="callService(this.form);" />
      </div>
    </form>

  </body>
  </html>
```

На рис. 1.9 показан внешний вид страницы после того, как она будет загружена и пользователь щелкнет на кнопке Call Service.

Запустите приложение (клавишей F5 или Ctrl+F5 в VWD). В окне броузера можно увидеть вполне предсказуемый результат, причем не только в Internet Explorer, но и в любом другом броузере, отвечающем нашим требованиям. Щелкните на кнопке несколько раз и убедитесь, что обновления страницы не происходит, хотя страница выполняет обмен данными с веб-службой на стороне сервера.

Элемент управления ScriptManager

Теперь, после рассмотрения первого примера, можно поговорить о том, как он работает и как работают остальные примеры ASP.NET AJAX из этой книги.

Рис. 1.9. *Приложение работает именно так, как и ожидалось*

Центральным элементом любой страницы с поддержкой ASP.NET AJAX является элемент управления `ScriptManager`. Он выполняет загрузку необходимых библиотек JavaScript платформы ASP.NET AJAX.

Если вы запустите приложение ASP.NET AJAX и затем просмотрете полученный исходный текст страницы в броузере, то увидите, что программный код немного изменился по сравнению с изначально введенным. Элемент `<asp:ScriptManager>` окажется замещен следующим программным кодом (учтите, что в вашей системе малопонятные значения в строках URL будут отличаться от тех, что приведены здесь):

```
<script src="/AJAXEnabledWebSite1/WebResource.axd?d=Jd4j-
uCaCWzJ5gY8Rtbjnw2&amp;t=632962478475625000" type="text/javascript"></script>

<script src="/AJAXEnabledWebSite1/ScriptResource.axd?d=4vKPTV3rK3vcGz1fNEcIXI-
FjnEGgHGMpqfUlmBk4NA5KxnrqcWXFT6hln9QkTuglOUzzonzRPSF5F3_-
OaWhWOb3FCqEciv4AZjgqdK5us1&amp;t=633074690770156250" type="text/javascript">
</script>
<script src="/AJAXEnabledWebSite1/ScriptResource.axd?d=4vKPTV3rK3vcGz1fNEcIXI-
FjnEGgHGMpqfUlmBk4NA5KxnrqcWXFT6hln9QkTuglOUzzonzRPSF5F3_-
OaWhdU1qTQPzcChFvbHT6FrI-81&amp;t=633074690770156250"
type="text/javascript"></script>
```

Кроме того, платформа ASP.NET AJAX сгенерирует программный код JavaScript, выполняющий инициализацию отдельных частей инфраструктуры Ajax.

При сборке приложения в режиме отладки программный код JavaScript, создаваемый ASP.NET, будет отформатирован, и в его состав будет включена обработка некоторых ошибок. Это удобно в процессе разработки, но не требуется (более того, рекомендуется обратное) при развертывании веб-сайта, поскольку это увеличивает объем программного кода JavaScript, а следовательно, и размер страницы.

Загрузка дополнительных файлов JavaScript

Элемент ScriptManager может также использоваться для загрузки дополнительных библиотек JavaScript: как тех, что входят в состав ASP.NET AJAX, так и ваших собственных:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Scripts>
    <asp:ScriptReference Path="MyScript.js" />
  </Scripts>
</asp:ScriptManager>
```

Обратите внимание: программный код JavaScript, сгенерированный ASP.NET AJAX, выполняет проверку, что все внешние файлы JavaScript загружены без ошибок. Если вы пишете свои собственные сценарии JavaScript, не забудьте добавить следующий фрагмент кода в конец каждого файла *.js*:

```
if (typeof(Sys) != "undefined") {
    Sys.Application.notifyScriptLoaded();
}
```

Этот фрагмент извещает ASP.NET AJAX, что был достигнут конец файла, и сигнализирует платформе Ajax, что все внешние сценарии были полностью загружены.

Элемент ScriptManager должен присутствовать на любой странице, которая использует функциональные возможности ASP.NET AJAX.

Если вы используете шаблонные страницы ASP.NET 2.0 и большая часть ваших страниц использует функциональные возможности платформы ASP.NET AJAX, есть смысл разместить элемент управления ScriptManager не в отдельных страницах, а в странице шаблона. Однако такой подход может вызвать сложности при необходимости обратиться из какой-то страницы к дополнительным файлам JavaScript или веб-службам (таким как в примере «Hello User», рассмотренном ранее). Поскольку на странице может находиться только один элемент ScriptManager, то вы будете вынуждены мириться со ссылками на файл JavaScript или веб-службу в каждой странице, использующей функциональность ASP.NET AJAX, даже если на этой странице внешние ресурсы будут невостребованы.

Для решения этой проблемы платформа ASP.NET AJAX предоставляет элемент управления ScriptManagerProxy. Данный элемент обеспечивает необходимую функциональность ScriptManager даже при наличии другого элемента ScriptManager.

```
<asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
  <Scripts>
    <asp:ScriptReference Path="MyScript.js" />
```

```
    </Scripts>
  </asp:ScriptManagerProxy>
```

Элемент ScriptManagerProxy удобно использовать при создании пользовательских элементов управления, обладающих функциональностью AJAX, когда на главной странице уже присутствует элемент ScriptManager.

Это был лишь первый шаг. А сколько всего откроется в следующих главах!

Подведение итогов

Эта глава – введение в платформу ASP.NET AJAX; в ней мы рассказали о том, какое отношение эта платформа имеет к Ajax и ASP.NET 2.0, и провели вас через процесс установки ASP.NET AJAX и другого программного обеспечения, необходимого для опробования примеров из этой книги, включая базу данных *AdventureWorks*. Здесь, кроме того, был представлен первый работающий пример ASP.NET AJAX и элемент управления ScriptManager – один из двух ключевых серверных элементов управления, которые входят в состав ASP.NET AJAX. В следующей главе будет рассматриваться язык программирования JavaScript, знание которого потребуется вам при работе с платформой ASP.NET AJAX.

Для дополнительного чтения

http://ajax.asp.net/

Домашняя страница проекта ASP.NET AJAX

http://ajax.asp.net/downloads/default.aspx?tabid=47

Страница, откуда можно загрузить ASP.NET AJAX

http://ajax.asp.net/docs/InstallingASPNETAJAX.aspx

Документация Microsoft с инструкциями по установке ASP.NET AJAX

http://blogs.msdn.com/webdevtools/archive/2007/07/28/upgrading-asp-net-ajax-1-0-websites-and-web-applications-to-net-framework-3-5.aspx

Важная информация по обновлению веб-сайтов с поддержкой ASP.NET AJAX, созданных в Visual Studio 2005, при переходе на Visual Studio 2008/.NET Framework 3.5

http://weblogs.asp.net/scottgu/archive/2007/08/04/fixes-for-common-vs-2008-and-net-3-5-beta2-issues.aspx

Перечень известных проблем в Visual Studio 2008 Beta 2 и способы их решения.

2

JavaScript

Интерактивность позволяет делать веб-страницы более полезными, динамичными и интересными для пользователя. Возможность встраивать сценарии в веб-страницы – это ключ к повышению интерактивности. Сценарии могут использоваться для обработки различных событий, таких как окончание загрузки страницы или щелчок на кнопке. Кроме того, сценарии могут использоваться для обработки данных, отправляемых на сервер или получаемых от сервера с использованием протокола HTTP.

Для большинства веб-разработчиков основным языком программирования является JavaScript, так как только он поддерживается всеми основными типами броузеров. Знание языка JavaScript совершенно необходимо, чтобы извлечь максимальную выгоду от использования платформы ASP.NET AJAX. Однако такая Ajax-платформа, как ASP.NET AJAX, позволяет использовать эту технологию, не требуя глубокого знания всех тонкостей. Фактически платформа ASP.NET AJAX организована так, что может использоваться даже теми разработчиками, кто вообще не знаком с языком JavaScript. Тем не менее ASP.NET AJAX – это всего лишь платформа, без знания JavaScript ваши возможности будут ограничены функциональными возможностями, которые предлагают элементы управления ASP.NET AJAX. Для некоторых пользовательских сценариев на основе ASP.NET AJAX требуется больше, чем просто использовать сценарии JavaScript, сгенерированные платформой. Таким образом, при разработке веб-сайтов с поддержкой технологии Ajax лучшим выбором будет использовать всю мощь обеих технологий и дополнить возможности платформы ASP.NET AJAX применением программирования на JavaScript.

Эта книга описывает платформу ASP.NET AJAX, и потому полноценное обучение JavaScript не входит в ее задачи. Цель этой главы состоит в том, чтобы ввести основные понятия и дать вам тем самым возмож-

ность разобраться в примерах, приводимых в книге. Следующее ниже описание JavaScript не претендует на полноту охвата и фокусируется на освещении лишь самых важных возможностей. За дополнительной информацией о JavaScript обращайтесь к ресурсам, указанным в разделе «Для дополнительного чтения» в конце этой главы.

JavaScript: история появления

Язык JavaScript был создан в 90-х годах прошлого века Бренданом Эйчем (Brendan Eich), инженером компании Netscape. Первоначально он назывался Mocha и впервые появился в 1995 году, в третьей бета-версии броузера Netscape Navigator 2.0. Позже, в этом же году, Netscape урегулировала спорные моменты с компанией Sun Microsystems – владельцем языка программирования Java, – касающиеся использования имени JavaScript (до этого в течении некоторого времени язык назывался LiveScript). Такое название вызвало некоторое замешательство, так как языки Java и JavaScript кроме С-подобного синтаксиса не имеют ничего общего.

Использование JavaScript позволило создавать по-настоящему динамические HTML-страницы: производить моментальную проверку данных форм, использовать графические эффекты, взаимодействовать с пользователем и многое другое. Во времена, когда пропускная способность сетей была весьма ограниченной (большинство пользователей имели медленные коммутируемые подключения к Интернету) и обмен данными с серверами был сопряжен с длительными задержками, JavaScript давал возможность разработчикам существенно повысить интерактивность своих сайтов. Когда рост популярности JavaScript стал очевиден, компания Microsoft добавила поддержку возможностей этого языка в свой броузер Internet Explorer. Из-за необходимости соблюдения авторских прав эта версия языка получила имя JScript, но по всем основным параметрам он был идентичен языку JavaScript.

В 1997 году война броузеров, разгоревшаяся между Netscape Navigator (в то время он занимал ведущее положение на рынке) и Internet Explorer (был близок к тому, чтобы занять ведущее положение), достигла своей кульминации. В июне в свет вышел Netscape 4, который имел поддержку JavaScript 1.2, обладавшего новыми возможностями. В этом же месяце организация ECMA (частная организация, деятельность которой посвящена стандартизации информационных и коммуникационных технологий) опубликовала стандарт ECMA-262, в котором содержалось формальное описание языка JavaScript (*http://www.ecma-international.org/publications/standards/Ecma-262.htm*).

В октябре того же года вышел Internet Explorer 4. Он поддерживал только JavaScript 1.1. (Кроме этого, он обладал поддержкой VBScript – языка сценариев, основанного на Visual Basic. Мы не будем рассматривать VBScript, так как его поддержка имеется только в Internet Explorer и потому не подходит для создания клиентских сценариев, ориентированных на широкое использование.) В то время броузеры были практически несовместимы; это было особенно заметно, когда дело доходило до реализации таких эффектов, как позиционирование и перемещение элементов. Смешение технологий, используемых для достижения подобных эффектов, привело к появлению динамического HTML (DHTML – Dynamic HTML). Вопреки расхожему мнению DHTML не является стандартом – это просто введенный термин, так же как и Ajax.

После этого ситуация кардинально изменилась. В Netscape отменили выход практически готовой 5-й версии броузера и решили полностью переписать его. Это привело к появлению сырой версии Netscape 6, основанной на новом варианте проекта Mozilla с открытыми исходными кодами и на механизме отображения Gecko. Задержки и проблемы с качеством (разумеется, были и другие факторы) привели Netscape к потере существенной доли рынка, и Internet Explorer занял лидирующие позиции.

Однако разработка Internet Explorer замерла на 6-й версии, в результате чего броузер Firefox (также основанный на Mozilla, но без поддержки дополнительных возможностей, таких как чтение новостей и электронной почты) стал отнимать часть рынка, которым владел Internet Explorer. Internet Explorer 7 стал первой новой версией броузера, выпущенной за последние шесть лет, и гонка возобновилась.

Что касается JavaScript, изменений за последние годы было не так много. После исчезновения Netscape 4 основные броузеры – Internet Explorer и Mozilla, а также Safari, Konqueror и Opera – стали практически совместимы в части поддержки JavaScript, хотя некоторые отличия и проблемы все же наблюдаются.

Недостаток инноваций в мире броузеров также явился фактором, сдерживающим дальнейшее развитие JavaScript. Даже книги, посвященные этой теме, обновлялись довольно редко за последние годы. Однако ситуация изменилась с появлением термина Ajax (подробнее об этом говорится в главе 3). Технологии, составляющие основу Ajax, были известны с 1998 года, но лишь совсем недавно они стали занимать господствующее положение в веб-программировании. Однако более подробно этот вопрос будет рассматриваться в главе 3.

Язык программирования JavaScript

Можно в общих чертах считать, что JavaScript основан на языке программирования C, поэтому программисты, обладающие навыками работы с языками C, C++, C# или Java, овладевают синтаксисом JavaScript за довольно короткое время. Удобство JavaScript обусловлено определенными аспектами. Например, он не является строго типизированным языком – программист не должен указывать типы данных, они определяются автоматически во время исполнения. Кроме того, JavaScript обладает некоторой поддержкой объектно-ориентированного стиля программирования, однако он не является обязательным к использованию, как это сделано в других языках программирования, таких как C# или Java.

Встраивание программного кода JavaScript в веб-страницы можно выполнить тремя способами: в виде сценариев, в виде обработчиков событий и в строке URL. В каждом из этих трех случаев используется различный синтаксис.

Сценарии, встроенные в веб-страницу

Обычно встраивание сценариев в HTML-страницу производится с помощью HTML-элемента `<script>`. Можно также использовать атрибут `src` тега `<script>` для задания URL, когда необходимо загрузить внешний файл сценария.

Во всех основных броузерах по умолчанию предполагается, что сценарии, заключенные в теги `<script>`, будут написаны на языке JavaScript. Однако для соответствия стандартам W3C (World Wide Web Consortium) и следуя необходимости обеспечить работоспособность сценариев в редко используемых или устаревших броузерах желательно указать язык, используя для этого следующий синтаксис:

```
<script language="JavaScript" type="text/javascript">
    ...
</script>
```

Использование сценариев для обработки событий

Программный код JavaScript может использоваться в тегах HTML, в виде значений атрибутов обработчиков событий, например:

```
<input type="button" onclick="doSomething();" />
```

Программный код JavaScript в строке URL

Допускается использование программного кода в строке URL с помощью указания специального псевдопротокола `javascript:`, что упрощает использование JavaScript в гиперссылках (например, `<a href="javascript: doSomething();">click me</a>`).

Наиболее часто используются первые два способа; они будут продемонстрированы в следующих разделах, где также будет рассказываться о ключевых элементах языка JavaScript.

Броузеры, не обладающие поддержкой JavaScript

Несколько лет тому назад элемент <script> обычно выглядел иначе:

```
<script language="JavaScript" type="text/javascript">
  <!--
    ...
  //-->
</script>
```

Элементы, обозначающие комментарии HTML (<!-- и -->), использовались для того, чтобы заставить броузеры, не имеющие поддержки JavaScript, игнорировать программный код. (Два символа слэша перед закрывающим тегом комментария (//-->) обозначают комментарий JavaScript, что заставляет интерпретатор JavaScript игнорировать закрывающий тег комментария HTML.)

Однако это осталось в прошлом. Сейчас, если броузер не поддерживает JavaScript, он просто игнорирует элементы <script>.

Общие методы JavaScript

В языке JavaScript имеются два метода, которые постоянно будут встречаться в примерах этой главы:

`document.write("текст")`

Этот метод выводит заданный текст в окне броузера.

`window.alert("текст")`

Этот метод открывает модальное окно и отображает в нем заданный текст сообщения.

В примере 2.1 приводится текст разметки страницы, которая использует второй метод для вывода текста предупреждения.

Пример 2.1. Использование JavaScript

JavaScript.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
  <script language="JavaScript" type="text/javascript">
  window.alert("Hello from JavaScript!");
  </script>
</head>
<body>
</body>
</html>
```

Рис. 2.1. *Модальное окно, созданное JavaScript-функцией alert*

На рис. 2.1 представлен результат, который был получен при исполнении примера 2.1. В Internet Explorer 6 SP 2 вы можете при этом получить предупреждение системы безопасности о том, что производится попытка запустить активное содержимое страницы, расположенной в локальной файловой системе. Если страница будет находиться на удаленном веб-сервере, это предупреждение выводиться не будет.

Переменные

Объявление переменных в JavaScript производится с помощью ключевого слова `var`. Совершенно необязательно следовать соглашениям об именовании переменных, таким как использование префиксов, определяющих типы переменных, как это принято в других языках, таких как Perl или PHP. Все переменные имеют глобальную область видимости, если они объявляются за пределами функций. Переменные не привязаны жестко к определенному типу данных, что позволяет изменять их тип во время исполнения. Тем не менее в JavaScript имеется несколько встроенных типов данных[1], включая следующие четыре, которые вы будете использовать неоднократно:

- `Number(1, -2, 3.14159)`
- `String("Hello", 'World')`
- `Boolean(true, false)`
- `RegEx(/d+/)`

[1] В спецификации ECMA-262 (ECMAScript Language Specification, 3rd edition) список основных типов данных представлен так: Number, String, Boolean, Undefined, Null. В спецификации также представлены объекты, среди которых, в частности, RegExp – объект для операций с регулярными выражениями. – *Примеч. науч. ред.*

Существует еще один тип данных – объекты. Например, Date, который используется для организации хранения значений даты. Однако в действительности он является не типом данных, а классом, который может использоваться для получения текущей даты и выполнения вычислений над ней.

Присваивание значений переменным производится с помощью оператора =, как показано ниже.

```
var i = 0;         // Объявляет переменную и присваивает ей значение 0
i = "JavaScript"; // Записывает строку в переменную
i = false;         // Записывает в переменную логическое значение
```

В отличие от других языков программирования, таких как Perl или PHP, в JavaScript не существует никаких отличий между кавычками и апострофами. Обратите внимание: символ завершения инструкции (;) не является обязательным, тем не менее рекомендуется использовать его, чтобы избежать случайного объединения инструкций.

Переменные JavaScript, в зависимости от текущего типа данных, поддерживают методы классов, ассоциированных с этим типом данных. Например, для любой строки может вызываться метод substring(), который возвращает часть строки, и метод indexOf(), который отыскивает наличие подстроки в текущей строке.

Массивы

Массив – это переменная, содержащая последовательность значений. Так как JavaScript не является строго типизированным языком, массив может одновременно хранить данные разных типов. Создать массив можно двумя способами. Один из них заключается в использовании конструкции new Array(), в которой указывается несколько значений. Индексация массивов начинается с нуля, таким образом, в следующем фрагменте в массив добавляется седьмой элемент:

```
var days = new Array("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday");
days[6] = "Saturday";
```

Современные броузеры предлагают альтернативную, более краткую форму определения массива, как показано ниже:

```
var days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday"];
```

Управляющие конструкции

Язык JavaScript поддерживает стандартный набор управляющих конструкций, включая switch, if...else и различные конструкции циклов (for, for...in, foreach, while, do...loop и do...until). Начнем с инструкции if. Пример 2.2 генерирует случайные числа с помощью Math.random() (этой возможностью мы воспользуемся в этой книге снова). Эта

встроенная функция возвращает случайное число в диапазоне от 0 (включительно) до 1 (не входит в диапазон). При умножении полученных чисел на 6 мы получим случайные числа от 0 (включительно) до 6 (не входит в диапазон). Округляя числа с помощью метода `Math.ceil()`, мы получим целые значения от 1 до 6 (включительно), имитируя тем самым один бросок игрального кубика.

Пример 2.2. Использование конструкции if...else и Math.random

```
JavaScript-if.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
  <script language="JavaScript" type="text/javascript">
  var rand = Math.random();
  rand = Math.ceil(6 * rand);
  if (rand % 2 == 1) {
    document.write("Odd number: ");
  } else {
    document.write("Even number: ");
  }
  document.write(rand);
  </script>
</head>
<body>
</body>
</html>
```

Результат работы сценария приводится на рис. 2.2.

В примере 2.2 были использованы некоторые дополнительные элементы JavaScript, описание которых приводится ниже.

Логические операторы

! (восклицательный знак) – в JavaScript это оператор логического отрицания; `||` – оператор логического ИЛИ; `&&` – оператор логического И.

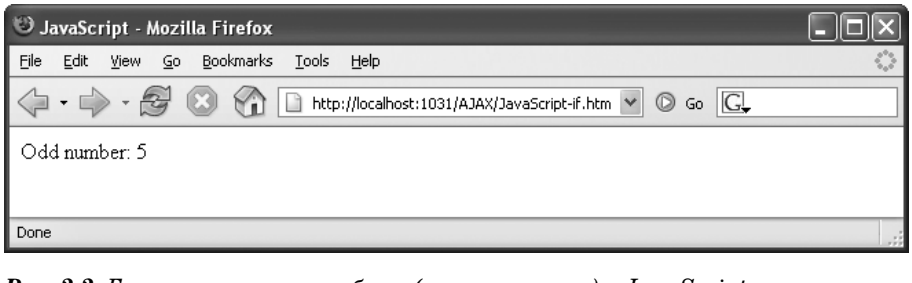


Рис. 2.2. *Бросок игрального кубика (виртуального) в JavaScript*

Операторы сравнения

Оператор == проверяет равенство операндов (в отличие от оператора =, который выполняет операцию присваивания); из других операторов сравнения имеются >=, >, <, <= и !=. Особое значение имеет оператор ===, который не только сравнивает значения операндов, но их типы; отрицательная форма этого оператора записывается как !==.

JavaScript поддерживает терциальный (трехместный) оператор, который удобно использовать в качестве сокращенной формы записи конструкции if...else. Следующее выражение

```
var output = (rand % 2 == 1) ? "odd" : "even";
```

эквивалентно такой форме записи:

```
if (rand % 2 == 1) {
  var output = "odd";
} else {
  var output = "even";
}
```

Вместо последовательности операторов if для проверки одного и того же выражения на равенство нескольким значениям можно воспользоваться инструкцией switch. Пример 2.3 демонстрирует, как это сделать.

Пример 2.3. Использование инструкции switch

```
JavaScript-switch.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
  <script language="JavaScript" type="text/javascript">
  var rand = Math.random();
  rand = Math.ceil(6 * rand);
    switch (rand) {
      case 1:
      case 3:
      case 5:
      document.write("Odd number: ");
      break;
    default:
      document.write("Even number: ");
    }
  document.write(rand);
  </script>
</head>
<body>
</body>
</html>
```

Как вы можете видеть, только инструкция `break` приводит к выходу из инструкции `switch`. Без этой инструкции интерпретатор JavaScript продолжил бы исполнение оставшихся инструкций, даже если выражение в инструкции `switch` совпало с одним из значений `case`.

Для организации повторяющегося исполнения фрагментов программного кода удобно использовать циклы. Например, цикл `for` можно использовать для обхода элементов массивов. Любой массив имеет свойство (`length`), которое возвращает количество элементов в массиве. В примере 2.4 используется цикл `for` для вывода значений всех элементов массива.

Пример 2.3. Использование цикла for

```
JavaScript-for.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
  <script language="JavaScript" type="text/javascript">
  var days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday"];
  for (var i=0; i < days.length; i++) {
    document.write(days[i] + "<br />");
  }
  </script>
</head>
<body>
</body>
</html>
```

Результат работы сценария приводится на рис. 2.3.

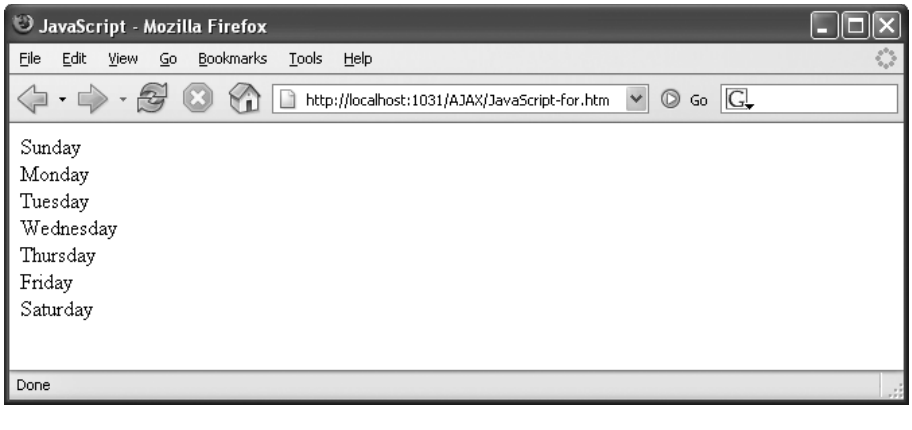


Рис. 2.3. *Обход элементов массива с помощью цикла for*

В примере 2.4 были использованы некоторые дополнительные элементы языка JavaScript. Выражение `i++`, используемое в цикле `for` для организации итераций, – это краткая форма записи выражения `i = i + 1`. (`i++` – это родственное выражение). Примите во внимание: оператор + может использоваться не только для сложения чисел, но также для конкатенации строк.

Кроме того, в языке имеется инструкция цикла `for...in`, которая аналогична инструкции `foreach` в языке C# и других родственных языках. Ее использование демонстрируется в примере 2.5. На каждой итерации в переменную цикла записывается значение текущего элемента. Если вы используете `foreach` для обхода объекта, вы получаете доступ ко всем свойствам и методам этого объекта. В случае с массивом вы получаете отдельно индексы элементов массива. Таким образом, при использовании массива `days` из предыдущего примера в переменную цикла последовательно будут попадать значения от 0 до 6, а не строки от `"Sunday"` до `"Saturday"`.

Пример 2.5. Использование цикла for...in

```
JavaScript-for-in.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
  <script language="JavaScript" type="text/javascript">
  var days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "
Friday", "Saturday"];
  for (var day in days) {
    document.write(days[day] + "<br />");
  }
  </script>
</head>
<body>
</body>
</html>
```

В JavaScript имеется еще несколько инструкций циклов, которые работают похожим образом: они продолжают работу либо *пока* условие соблюдается, либо *до тех пор, пока* соблюдается условие[1]. В примере 2.6 демонстрируется использование цикла `while`, который используется наиболее часто.

Пример 2.6. Использование цикла while

```
JavaScript-while.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

[1] Имеются в виду конструкции операторов *while(...) {...}* и *do {...} while(...)*. – *Примеч. науч. ред.*

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
  <script language="JavaScript" type="text/javascript">
  var days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday"];
  var i = 0;
  while (i < days.length) {
    document.write(days[i] + "<br />");
    i++;
  }
  </script>
</head>
<body>
</body>
</html>
```

Встроенные методы, собственные функции и обработка событий

В JavaScript имеется целый ряд встроенных объектов, но есть возможность создавать и свои собственные функции (и объекты). Объявление функции начинается с ключевого слова `function`. В JavaScript отсутствует возможность указать тип данных возвращаемого значения (и, как следствие, в языке отсутствует ключевое слово `void`); функция может ничего не возвращать. Если вы предполагаете возвращать какое-либо значение, используйте инструкцию `return`.

В примере 2.7 демонстрируется использование метода `replace()`, доступного для всех строк, который обеспечивает поддержку регулярных выражений. Как можно заметить, следующий сценарий преобразует некоторые символы, имеющие специфическое назначение в языке разметки HTML, в соответствующие специальные последовательности символов. Сценарий производит несколько обращений к методу `replace()` подряд. Первым вызовом производится замещение символа `&` на его эквивалент (`&`), затем друг за другом производится замещение остальных служебных символов (`<`, `>`, `"` и `'`) на соответствующие им «экранированные» сущности. Таким образом, любая строка, обработанная сценарием, будет преобразована в разметку HTML, как это сделал бы ASP.NET-метод `Server.HtmlEncode()`.

Пример 2.7. Создание собственной функции

```
JavaScript-function.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
```

```
</head>
<body>
  <script language="JavaScript" type="text/javascript">
  function HtmlEscape(s) {
    var result = s.replace(/&/g, "&amp;")
                  .replace(/</g, "&lt;")
                  .replace(/>/g, "&gt;")
                  .replace(/"/g, "&quot;")
                  .replace(/'/g, "&apos;");
    return result;
  }
  document.write(HtmlEscape("<hr />"));
  </script>
</body>
</html>
```

В результате исполнения этого сценария будет выведена строка `<hr />`, которая будет отображаться броузером как `<hr />`, но не будет приводить к выводу горизонтальной линии.

Язык JavaScript не поддерживает возможность перегрузки функций; он не фиксирует число их аргументов. Если функции передать число аргументов меньшее, чем было указано при ее объявлении, недостающие аргументы внутри функции получат значение `null`.

При этом, как демонстрирует пример 2.8, если функции передается большее число аргументов, чем было предусмотрено, доступ к избыточным аргументам можно получить с помощью свойства `arguments` (сокращенно от `<ИмяФункции>.arguments`).

Пример 2.8. Создание собственной функции с переменным числом аргументов

JavaScript-function-arguments.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
</head>
<body>
  <script language="JavaScript" type="text/javascript">
  function OutputList() {
    document.write("<ul>");
    for (var i=0; i < arguments.length; i++) {
      document.write("<li>" +
                     arguments[i] +
                    "</li>");
    }
    document.write("</ul>");
  }
  OutputList("one", "two", "three");
  </script>
```

```
</body>
</html>
```

Результат работы сценария приводится на рис. 2.4.

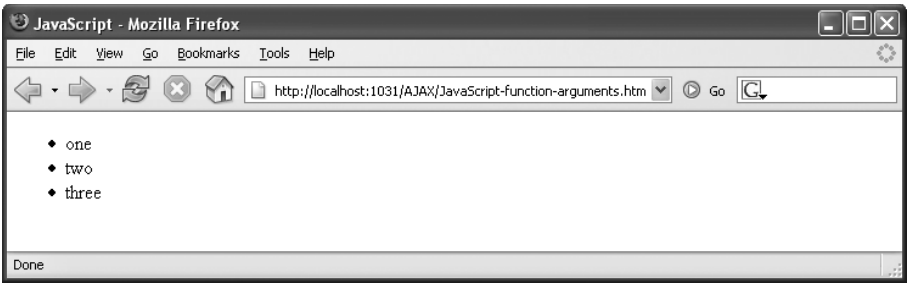


Рис. 2.4. *Собственная функция генерирует маркированный список*

Кроме того, JavaScript поддерживает *анонимные функции*, то есть функции, которые не имеют имени. В JavaScript анонимные функции иногда используются для обработки событий. Например, атрибуту `onload` тега `<body>` можно присвоить в качестве значения программный код JavaScript, который будет исполнен единственный раз, сразу по завершении загрузки HTML-страницы. (В действительности таким образом можно обрабатывать любые события, привязанные к разметке HTML.) Это – встроенный обработчик события на языке JavaScript.

Чтобы определить обработчик события, достаточно присвоить атрибуту, имя которого является результатом конкатенации строки `on` с наименованием события, программный код JavaScript, который должен быть исполнен при появлении этого события. Существуют и другие способы привязать программный код к событию, но методика на основе анонимных функций и атрибутов HTML используется наиболее часто.

Следующий фрагмент выведет окно со словом `"Loaded"` по окончании загрузки страницы:

```
<body onload="alert('Loaded.');">
```

В общем случае синтаксис определения обработчика события выглядит так, как показано ниже:

```
<body onload=" "Functionname();">
```

Связать событие с функцией-обработчиком можно и программным способом. Базовый объект страницы называется `window`, таким образом можно присвоить функцию свойству `window.onload`, как показано в следующем примере:

```
function Functionname() {
    // обработка события
}
window.onload = Functionname;
```

Если данная функция нигде в программе больше не используется, вы можете присвоить непосредственно свойству события анонимную функцию. Такой прием позволит несколько сократить предыдущий пример, как показано ниже:

```
window.onload = function() {
    // обработка события
}
```

Наконец, ниже приводится пример анонимной функции, которая принимает два аргумента:

```
window.onload = function(a, b) {
    // обработка события с использованием аргументов a и b
}
```

Такой подход очень удобен и, как можно заметить, часто используется платформой ASP.NET AJAX.

Объектно-ориентированное программирование (ООП)

JavaScript – это язык, основанный на объектах, но он не является объектно-ориентированным (ООП) языком. В JavaScript есть некоторые аспекты, напоминающие ООП, но поддержка возможностей, обычных для ООП, весьма ограничена. Например, область видимости членов класса (`public`, `private`, `protected` и прочие) может быть реализована с существенными ограничениями. Однако в JavaScript есть возможность создавать классы и даже имеется рудиментарная поддержка механизма наследования.

Классы в JavaScript реализуются в виде функций. Программный код внутри этих функций – это реализация конструктора класса. Она должна содержать объявления и определения всех элементов экземпляров (то есть всех общедоступных и частных свойств и методов). Для ссылки на текущий экземпляр класса можно использовать ключевое слово `this`, благодаря чему конструктор – и методы класса – получают доступ к свойствам текущего класса. Точно так же программный код класса имеет возможность присваивать свойства значениям и определять методы, которые затем будут доступны из программного кода за пределами класса.

В примере 2.9 показан простой класс, который реализует абстракцию книги. Обратите внимание: нами предусмотрено хранение данных в частных свойствах класса, которые недоступны извне класса. Далее, мы определяем общедоступные методы чтения и записи для доступа к этим свойствам. Такой прием называется *инкапсуляцией* и позволяет унифицировать доступ к свойствам и организовать проверку значений до того, как они будут записаны в свойства.

Пример 2.9. Использование ООП-особенностей JavaScript

```
JavaScript-class.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
</head>
<body>
  <script language="JavaScript" type="text/javascript">
  function Book(isbn, author, title) {
    var _isbn = isbn;
    var _author = author;
    var _title = title;

    this.get_isbn = function() {
      return _isbn;
    }
    this.set_isbn = function(value) {
      _isbn = value;
    }
    this.get_author = function() {
      return _author;
    }
    this.set_author = function(value) {
      _author = value;
    }
    this.get_title = function() {
      return _title;
    }
    this.set_title = function(value) {
      _title = value;
    }

    this.toString = function() {
      return this.get_author() + ": " + this.get_title() +
                " (" + this.get_isbn( ) + ")";
    }
  }
  var atlas = new Book("0792275438", "National Geographic");
  atlas.set_title("Atlas of the World");
  document.write(atlas.toString( ));
  </script>
</body>
</html>
```

Данный пример выведет следующий текст:

```
National Geographic: Atlas of the World (0792275438)
```

Поскольку JavaScript не поддерживает модификаторы доступа (такие как `private`, `protected` и `public` в языке C#), для реализации частных

свойств мы используем локальные переменные. Как уже говорилось, все свойства экземпляров, инициализированные с помощью ключевого слова this, будут доступны из-за пределов класса, а переменные, объявленные внутри класса, останутся частными. Частные переменные могут использоваться только внутри класса и не видны внешнему программному коду. Это единственный способ реализовать сокрытие данных и определить некое подобие частных свойств и методов. Механизм наследования в JavaScript также имеет ограниченные возможности. Для определения свойств и методов, которые будут доступны всем экземплярам классов-наследников, можно использовать свойство prototype. Следующий фрагмент добавит новый метод ко всем массивам:

```
Array.prototype.empty = function() {
  this.length = 0;
}
```

Следующее выражение определяет класс, наследующий свойства и методы другого класса:

```
DerivedClass.prototype = new BaseClass();
```

В примере 2.10 выполняется расширение класса Book с помощью класса DigitalBook, в котором объявляется дополнительное частное поле (реализованное в виде локальной переменной _size, обратиться к которой можно через методы доступа) и переопределяется метод toString(). Обратите внимание: в JavaScript нет защищенных свойств (свойства, которые доступны из подклассов), поэтому все поля-переменные базового класса необходимо объявлять повторно. («Подкласс» представлен здесь в контексте JavaScript – поскольку в этом языке программирования отсутствует «настоящий» механизм наследования ООП, то нет и самого понятия подкласса, но у вас имеется возможность определить похожее поведение, как это сделано в данном примере.) Однако существующие методы get и set будут доступны по-прежнему. Для доступа к этим методам необходимо использовать метод call() базового объекта (напоминает обращение к base в классах C#).

Пример 2.10. Использование механизма наследования в JavaScript

```
JavaScript-class-prototype.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
</head>
<body>
  <script language="JavaScript" type="text/javascript">
  function Book(isbn, author, title) {
    var _isbn = isbn;
    var _author = author;
    var _title = title;
```

```
    this.get_isbn = function() {
      return _isbn;
    }
    this.set_isbn = function(value) {
      _isbn = value;
    }
    this.get_author = function() {
      return _author;
    }
    this.set_author = function(value) {
      _author = value;
    }
    this.get_title = function() {
      return _title;
    }
    this.set_title = function(value) {
      _title = value;
    }

    this.toString = function() {
      return this.get_author() + ": " + this.get_title() + " (" +
                this.get_isbn( ) + ")";
    }
  }

  // класс, производный от класса Book
  function DigitalBook(isbn, author, title, size) {
    Book.call(this, isbn, author, title);
    var _size = (size != null) ? size : 0;
    this.get_size = function() {
      return _size;
    }
    this.set_size = function(value) {
      _size = value;
    }

    this.toString = function() {
      return this.get_author() + ": " + this.get_title() + " (" +
                this.get_isbn( ) + ")" +
                " - " + this.get_size() + " KB";
    }
  }
  DigitalBook.prototype = new Book(); // Породить от book

  var atlas = new DigitalBook("0123456789", "International Graphics",
                              "Atlas of the City");
  atlas.set_size(1024);
  document.write(atlas.toString());
  </script>
</body>
</html>
```

Результат работы сценария приводится на рис. 2.5.

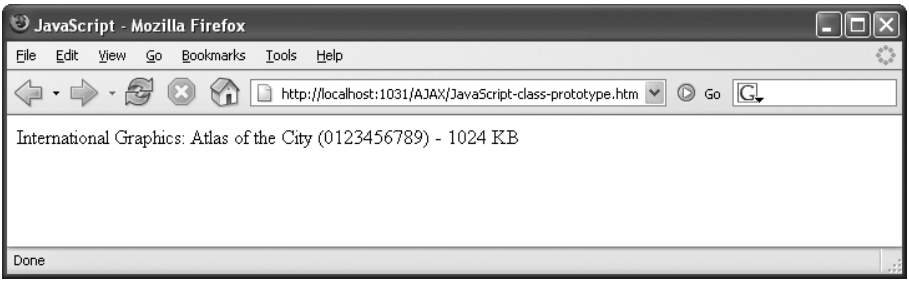


Рис. 2.5. *Результат работы метода toString( ) в порожденном объекте*

Доступ к элементам страницы

Хотя все современные броузеры поддерживают W3C DOM как средство доступа к элементам текущей HTML-страницы, тем не менее существуют более простые способы организации взаимодействия с элементами на странице, в чем вы убедитесь немного позднее (дополнительная информация приводится в разделе «Методы DOM» ниже, в этой же главе). Два из них будут рассмотрены в этом разделе.

Доступ к элементам формы

Объект JavaScript `document` обеспечивает доступ ко всем элементам текущей страницы. Данный объект является представлением объектной модели документа (DOM) в JavaScript. Чтобы сделать обращение к элементам страницы максимально удобным, у объекта предусмотрено наличие нескольких свойств, которые позволяют напрямую обращаться к специальным элементам страницы:

`document.embeds`

Массив, содержащий все данные, встроенные в текущую страницу с помощью тега `<embed>`

`document.forms`

Массив, содержащий все элементы `<form>` в текущей странице

`document.frames`

Массив, содержащий все фреймы на текущей странице

`document.images`

Массив всех изображений в текущей странице

`document.links`

Массив всех гиперссылок в текущей странице

Чаще других используется свойство `document.forms`, которое позволяет обращаться ко всем элементам `<form>` в текущей странице, таким как текстовые поля и кнопки. Обычно на странице имеется одна форма;

при этом свойство document.forms[0] обеспечивает доступ ко всем элементам первой формы. Далее, для доступа к отдельным элементам объект формы поддерживает массив elements[]. Благодаря доступу к элементам формы в веб-страницу можно добавлять новые возможности, например проверку данных на стороне клиента, как показано ниже:

```
<form>
  <input type="text" name="TextBox1" />
</form>
```

Выражение document.forms[0].elements["TextBox1"] обеспечивает доступ к элементу формы с именем TextBox1, находящемуся внутри первой формы в текущей странице. (Сокращенно – document.forms[0].TextBox1; однако, такая форма записи недопустима, если в атрибуте name использованы специальные символы – пробелы или дефисы.) В зависимости от типа элемента формы (например, текстовое поле, радиокнопка, флажок) порядок обращения к его значению (это может быть текст в текстовом поле или отмеченная радиокнопка) может отличаться, но, как правило, атрибут value содержит требуемую информацию аналогично тому, как HTML-атрибут value содержит значение для большей части полей формы.

Пример 2.11 отображает данные, введенные пользователем в текстовое поле, после щелчка на кнопке. Ниже приводится текст разметки для кнопки:

```
<input type="button" onclick="ShowText(this.form);" />
```

Когда вы щелкаете на кнопке, вызывается функция ShowText() с параметром this.form, который является ссылкой на родительскую форму элемента. Это несколько упрощает доступ к данным, так как удается избежать использования конструкции document.forms[0] в вызываемой функции. В примере 2.11 приводится полный текст примера.

Пример 2.11. Доступ к элементам формы

JavaScript-form-textbox.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
  <script language="JavaScript" type="text/javascript">
    function ShowText(f) {
      alert("Entered text: " + f.elements["TextBox1"].value);
    }
  </script>
</head>
<body>
  <form action="">
    <input type="text" name="TextBox1" />
    <input type="button" value="Show text" onclick="ShowText(this.form);" />
  </form>
```

```
</body>
</html>
```

Результат работы сценария приводится на рис. 2.6.

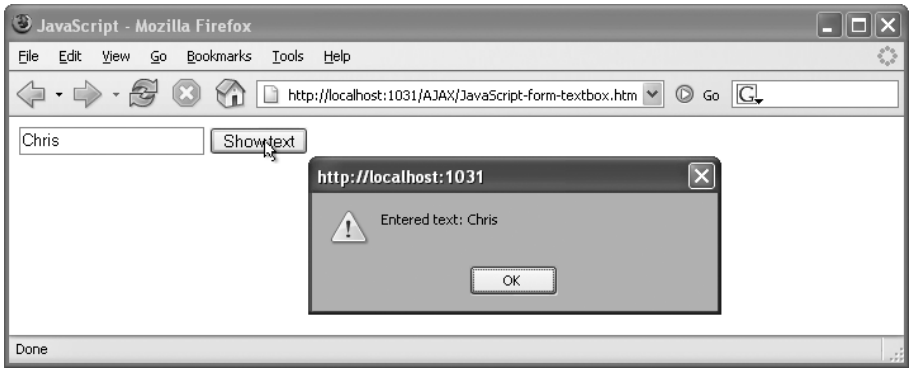


Рис. 2.6. *Данные формы отображаются в окне*

В табл. 2.1 перечислены свойства, которые можно использовать для доступа к значениям наиболее часто используемых типов полей формы. Например, значение текстового поля, описанного разметкой `<input type="text" name="Name" />`, может быть получено с помощью выражения `document.forms[0].elements["Name"].value` (исходя из предположения, что текстовое поле находится в первой или единственной форме в документе).

Таблица 2.1. HTML-элементы форм и связанные с ними свойства

Поле формы	Разметка HTML	Свойство

Текстовые поля и поля ввода пароля	`<input type="text">`
	`<input type="password">`
	`<textarea>`
		`value`: Возвращает или изменяет данные в текстовом поле

Радиокнопки	`<input type="radio">`	`checked`: Определяет, отмечена радиокнопка или нет, или устанавливает значение
Флажки	`<input type="checkbox">`	`checked`: Определяет, отмечен флажок или нет, или устанавливает значение
Списки выбора	`<select>`	`selectedIndex`: Возвращает или изменяет индекс первого выбранного элемента (значение `-1` означает, что ни один элемент списка не выбран)
Элементы списков выбора	`<option>`	`checked`: Определяет, выбран этот элемент списка или нет, или устанавливает значение `value`: Возвращает значение элемента

Доступ к произвольным элементам

Свойство `document.forms` очень удобно использовать для доступа к данным формы. Одна из основных задач сценариев JavaScript, особенно когда они используются как часть реализации Ajax, заключается в отображении данных в таких элементах, как параграф (`<p>`) или текстовая метка (`<span>` или `<label>`). Получить требуемый результат можно, выполнив следующие действия:

1. С помощью атрибута name определить уникальный идентификатор текстового элемента. Этот идентификатор никак не используется в запросах HTTP при отправке формы, но он часто используется для доступа к значениям элементов из JavaScript.
2. В сценарии JavaScript получить ссылку на элемент, воспользовавшись выражением `document.getElementById()`.
3. Записать содержимое элемента в свойство `innerHTML`.

В примере 2.12 производится вывод данных, которые снова извлекаются из текстового поля ввода, но на этот раз данные записываются в элемент `<span>`.

Пример 2.12. Запись разметки HTML и текста в элемент

```
JavaScript-form-label.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
  <script language="JavaScript" type="text/javascript">
  function HtmlEscape(s) {
    var result = s.replace(/&/g, "&amp;")
                  .replace(/</g, "&lt;")
                  .replace(/>/g, "&gt;")
                  .replace(/"/g, "&quot;")
                  .replace(/'/g, "&apos;");
    return result;
  }

  function ShowText(f) {
    var label = document.getElementById("Label1");
    label.innerHTML = HtmlEscape(f.elements["TextBox1"].value);
  }
  </script>
</head>
<body>
  <form action="">
    <input type="text" name="TextBox1" />
    <input type="button" value="Show text" onclick="ShowText(this.form);" />
    <p>Entered text: <span id="Label1">---</span></p>
  </form>
```

```
</body>
</html>
```

По умолчанию элемент содержит три дефиса (-). После щелчка на кнопке дефисы будут замещены данными из текстового поля, в которых служебные символы HTML будут замещены их эквивалентами. Результат приводится на рис. 2.7.

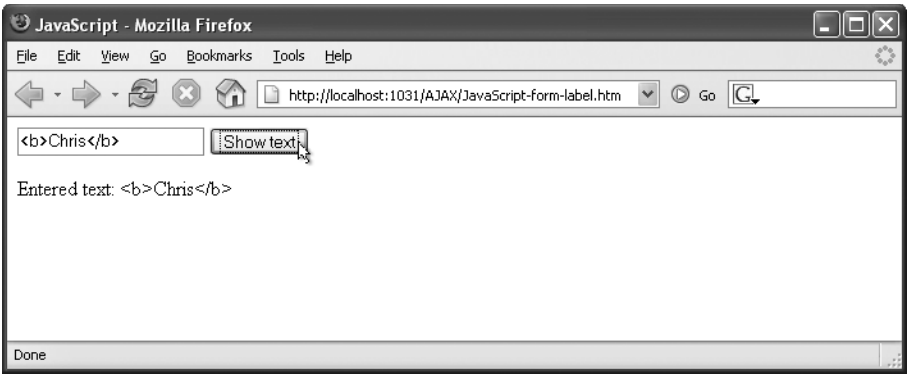


Рис. 2.7. *После замены служебных символов HTML на их эквиваленты текст выводится в элементе *

Методы DOM

В большинстве ситуаций, когда необходимо организовать взаимодействие с элементами HTML-страницы, используется свойство `innerHTML` элементов, доступ к которым организован с помощью специального объекта JavaScript `document.forms`, родственных ему объектов или с помощью метода `document.getElementById()` (который возвращает элемент по заданному идентификатору элемента). Однако иногда возникают ситуации, когда требуется получить доступ к самой DOM. В списке ниже приведены наиболее важные методы, предоставляющие такую возможность:

`getElementsByTagName(`*`name`*`)`

Возвращает массив всех элементов в странице с именем элемента *name*

`createElement(`*`name`*`)`

Создает новый узел DOM с именем элемента *name*

`createAttribute(`*`name`*`)`

Создает новый атрибут узла с именем атрибута *name*

`createTextNode(`*`text`*`)`

Создает новый текстовый узел DOM (текст внутри элемента) и записывает в него текст *text*

`appendChild(node)`

Добавляет дочерний узел *node* к текущему элементу

В приложении В приводится полный список поддерживаемых методов доступа к DOM.

В примере 2.13 демонстрируется, как можно использовать некоторые из этих методов, чтобы получить функциональный аналог предыдущего примера, но в этом случае элемент `<span>` и текстовый узел создаются динамически. В этом примере в игру вступает метод `appendChild()`. С его помощью сначала текстовый узел добавляется в элемент `<span>`, а затем сам элемент `<span>` добавляется в элемент параграфа.

Пример 2.13. Использование DOM в JavaScript

JavaScript-DOM.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
  <script language="JavaScript" type="text/javascript">
  function ShowText(f) {
    var paragraph = document.getElementsByTagName("p")[0];
    var label = document.createElement("span");
    var text = document.createTextNode(f.elements["TextBox1"].value);
    label.appendChild(text);
    paragraph.appendChild(label);
  }
  </script>
</head>
<body>
  <form action="">
    <input type="text" name="TextBox1" />
    <input type="button" value="Show text" onclick="ShowText(this.form);" />
    <p>Entered text: </p>
  </form>
</body>
</html>
```

Подведение итогов

В этой главе мы рассмотрели основы программирования на языке JavaScript на стороне клиента. О дополнительных возможностях JavaScript будет рассказываться в следующих главах. Тем не менее в этой главе вам были представлены все основные концепции, необходимые для понимания остальной части книги.

Для дополнительного чтения

Flanagan, David. JavaScript: The Definitive Guide, Fifth Edition (O'Reilly)[1]

Полное руководство программиста и справочник по языку JavaScript.

Flanagan, David. JavaScript Pocket Reference, Second Edition (O'Reilly)

Краткий, но достаточно полный обзор языка JavaScript.

[1] Дэвид Флэнаган «JavaScript. Подробное руководство», 5-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2008.

3

Ajax

Ajax – это набор технологий, на основе которых построена платформа ASP.NET AJAX. И хотя ASP.NET AJAX старательно скрывает механизмы работы Ajax, для понимания возможностей ASP.NET AJAX и написания профессиональных приложений, настраивающих платформу под ваши нужды, вам необходимо глубокое понимание сущности Ajax.

Термин «Ajax» был введен в обиход Джесси Джеймсом Гарреттом (Jesse James Garrett) в начале 2005 года в статье «A New Approach to Web Applications» (*http://www.adaptivepath.com/publications/essays/archives/000385.php*). Однако новым является только сам термин, но не технологии. Хотя XML, равно как и каскадные таблицы стилей, могут (но не обязательно) быть частью Ajax-приложения, тем не менее основу приложений с поддержкой Ajax составляет JavaScript.

В этой главе мы будем создавать веб-страницы, которые включают в себя как сценарии, исполняемые броузером на стороне клиента, так и сценарии, исполняемые на стороне сервера. Поэтому примеры в этой и в остальных главах книги будут строиться с привлечением ASP.NET, в виде страниц *.aspx*. Кроме того, мы рассмотрим три наиболее важные технологии на основе JavaScript, которые используются для поддержки Ajax в веб-приложениях. Список этих технологий приводится ниже:

`XMLHttpRequest`

Объект JavaScript, который реализует возможность передачи (асинхронных) вызовов HTTP.

`XMLDocument`

Объект JavaScript, который используется для разбора и для доступа к данным в формате XML.

`JavaScript Object Notation (JSON)`

Формат представления данных, который может использоваться вместо XML для организации обмена информацией между клиентом и сервером без дополнительной нагрузки, связанной с разбором XML.

Объект XMLHttpRequest

Основу Ajax составляет объект `XMLHttpRequest`, который позволяет отправлять HTTP-запросы и принимать ответы без необходимости отправлять на сервер форму целиком и полностью обновлять страницу.

История появления объекта XMLHttpRequest

Впервые объект `XMLHttpRequest` был реализован в 1999 году, в Internet Explorer 5. Эта версия броузера включала объект ActiveX с именем `XMLHttpRequest`, о назначении которого недвусмысленно говорит его название – он выполняет HTTP-запрос и получает ответ. (Сообщения могут быть в формате XML, но это совершенно необязательно.) Первоначально такого рода функциональность в Internet Explorer потребовалась для создания веб-интерфейса в Outlook (Outlook Web Access [OWA]), благодаря чему разработчики смогли реализовать приложение OWA, напоминающее своим поведением обычные настольные приложения. Несмотря на свою полезность, появившийся в Internet Explorer объект `XMLHttpRequest` некоторое время оставался незамеченным разработчиками веб-приложений. Однако позже разработчики альтернативных броузеров включили совместимую версию объекта в свои приложения. Поскольку только Internet Explorer поддерживает элементы управления ActiveX, во всех остальных броузерах объект `XMLHttpRequest` был реализован непосредственно в самих броузерах.

Первым броузером после Internet Explorer, в котором появилась поддержка `XMLHttpRequest`, стал броузер Mozilla 1.0 (не путайте с кодовым названием первых броузеров Netscape). Поддержка `XMLHttpRequest` сохранилась в последующих версиях Mozilla, а также в производных от него броузерах, таких как Camino (Mac OS X) и Firefox. Вслед за этим компания Apple добавила соответствующую поддержку в версию 1.2 броузера Safari. Этот броузер основан на механизме отображения KHTML, который является составной частью Konqueror, веб-броузера рабочей среды KDE в операционной системе Linux. Позднее инженерами из Apple поддержка `XMLHttpRequest` была перенесена и в Konqueror.

Opera 8.0 и последующие версии также включают поддержку `XMLHttpRequest`. Аналогичной поддержкой обладает и довольно экзотичная система Open Laszlo компании IBM.

XMLHttpRequest и стандарты

Несмотря на то, что объект `XMLHttpRequest` поддерживается большинством броузеров, он по-прежнему не является стандартом, так как не является частью спецификации ECMAScript. Однако существует спецификация W3C, которая определяет похожую функциональную особенность, а именно возможность динамической загрузки и отправки на сервер данных в формате XML. Эта спецификация называется «DOM Level 3 Load and Save» и была рекомендована консорциумом W3C в апреле 2004 года (*http://www.w3.org/TR/DOM-Level-3-LS*). Данный стандарт еще не реализован ни в одном из наиболее популярных броузеров и скорее всего пройдет какое-то время, прежде чем разработчики начнут работу над ним.

С другой стороны, недавно W3C выступил с инициативой стандартизации `XMLHttpRequest`. За дополнительной информацией обращайтесь по адресу: *http://www.w3.org/TR/XMLHttpRequest*.

Существенная часть броузеров на рынке обеспечивает поддержку `XMLHttpRequest`, а тем самым и совместимость с технологиями Ajax. Согласно исследованиям, проведенным компанией Net Applications в ноябре 2005 года (*http://www.netapplications.com*), примерно 99% используемых броузеров – это Internet Explorer версии 5 и выше, Mozilla 1.0 и выше, Firefox 1.0 и выше, Opera 8 и выше, Safari 1.2 и выше или KDE 3 и выше. Означает ли это, что практически любой может работать с Ajax-приложениями?

К сожалению, ответ на этот вопрос – нет. По разным исследованиям, от 5 до 15 процентов пользователей отключают поддержку JavaScript в своих броузерах. Возможно, из-за появляющихся время от времени сообщений о найденных в броузерах уязвимостях или из-за корпоративной политики безопасности.

В результате, несмотря на повсеместное распространение современных броузеров, возможна ситуация, когда существенная часть ваших пользователей *не сможет* использовать приложения, зависящие от поддержки JavaScript, включая и Ajax-приложения. Поэтому вы всегда должны иметь запасной план на тот случай, когда ваше приложение столкнется с броузером, неспособным использовать технологии Ajax.

Программирование объекта XMLHttpRequest

Порядок создания экземпляра объекта `XMLHttpRequest` зависит от броузера, в котором исполняется программный код. Следующий фрагмент будет работать в Internet Explorer 5 и выше. Из-за разных версий библиотеки Microsoft XML, которые поставляются с разными версиями Internet Explorer, в данном фрагменте производится попытка соз-

дать экземпляр XMLHttpRequest двумя разными способами. Чтобы избежать появления сообщений об ошибках при неудачном завершении одного из методов, используются две конструкции try-catch:

```
var XMLHTTP = null;
try {
  XMLHTTP = new ActiveXObject("Msxml2.XMLHTTP");
} catch (e) {
  try {
    XMLHTTP = new ActiveXObject("Microsoft.XMLHTTP");
  } catch (e) {
  }
}
```

В броузерах, отличных от Internet Explorer, синтаксис создания объекта намного проще:

```
XMLHTTP = new XMLHttpRequest();
```

Таким образом, все, что необходимо сделать, – это определить тип используемого броузера и выбрать соответствующий метод создания экземпляра XMLHttpRequest. Например, следующий фрагмент выясняет возможность создания объекта ActiveX путем проверки наличия свойства ActiveXObject у объекта window. Если свойство присутствует, значит, сценарий исполняется в Internet Explorer.

```
if (window.ActiveXObject) {
  // скорее всего, это IE
}
```

Для тех же целей можно использовать следующий фрагмент, который проверяет присутствие объекта XMLHttpRequest. При положительном результате это означает, что сценарий исполняется в Mozilla или его производных, Opera, Konqueror или Safari:

```
if (XMLHttpRequest) {
  // скорее всего, это не IE
}
```

Однако, как показано на рис. 3.1, непосредственная проверка наличия объекта XMLHttpRequest в Internet Explorer вызывает появление сообщения об ошибке «XMLHttpRequest is undefined» (объект XMLHttpRequest не определен). Положение вещей должно измениться с выходом Internet Explorer 7, который будет иметь встроенную реализацию объекта XMLHttpRequest[1].

Необходимо реализовать какой-то иной подход к выполнению проверок, показанных выше. В JavaScript имеется оператор typeof, который используется для определения типа выражения и возвращает строку

[1] Такая поддержка для IE7 уже объявлена ко времени подготовки перевода книги к изданию. – *Примеч. науч. ред.*

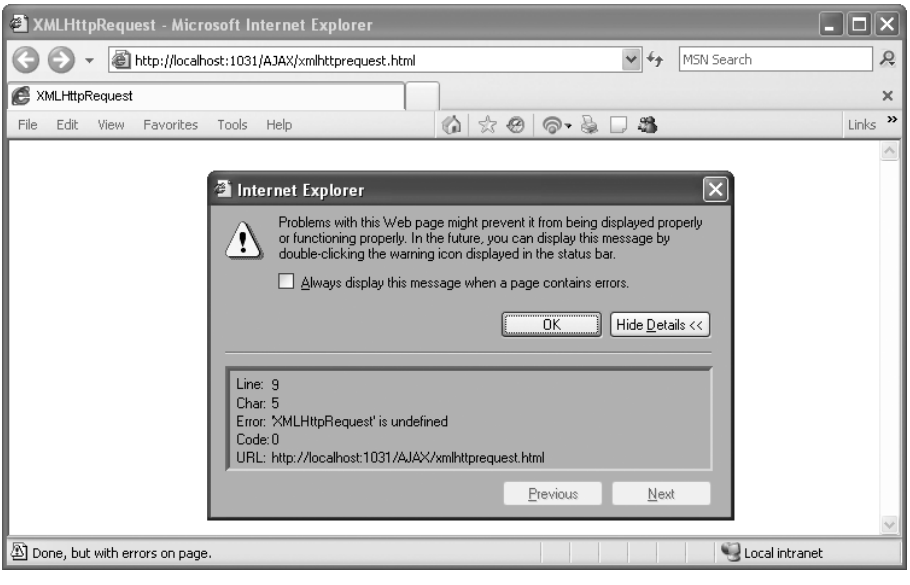


Рис. 3.1. *Наш программный код не понравился броузеру Internet Explorer*

«undefined», если выражение дает в результате значение «undefined». Следующий фрагмент позволяет определить броузеры, отличные от Internet Explorer:

```
if (typeof XMLHttpRequest != "undefined") {
  // это не IE версии 6 или ниже
}
```

Далее представлен исходный текст функции `getXMLHTTP()`, которая объединила в себе все предыдущие фрагменты. Она возвращает объект `XMLHttpRequest` независимо от того, какой броузер используется, главное, чтобы он обладал поддержкой Ajax и JavaScript.

```
function getXMLHTTP() {
  var XMLHTTP = null;
  try {
    XMLHTTP = new ActiveXObject("Msxml2.XMLHTTP");
  } catch (e) {
    try {
      XMLHTTP = new ActiveXObject("Microsoft.XMLHTTP");
    } catch (e) {
      if (typeof XMLHttpRequest != "undefined") {
        XMLHTTP = new XMLHttpRequest( );
      }
    }
  }
  return XMLHTTP;
}
```

Другой способ заключается в том, чтобы использовать стандартный для JavaScript способ определения функциональных возможностей броузера и выполнить проверку наличия свойства window.XMLHttpRequest (а не просто XMLHttpRequest), чтобы убедиться в наличии или отсутствии встроенной поддержки объекта XMLHttpRequest в броузере. Для реализации этой методики функция, которая возвращает объект, может быть переписана с небольшими изменениями, как показано в следующем фрагменте:

```
function getXMLHTTP() {
  var XMLHTTP = null;
  if (window.ActiveXObject) {
    try {
      XMLHTTP = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (e) {
      try {
        XMLHTTP = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (e) {
      }
    }
  } else if (window.XMLHttpRequest) {
    try {
      XMLHTTP = new XMLHttpRequest( );
    } catch (e) {
    }
  }
  return XMLHTTP;
}
```

Объект XMLHttpRequest, независимо от того, в каком броузере он был создан, имеет набор свойств и методов, которые используются для передачи HTTP-запросов и получения ответов от сервера. В большинстве ситуаций, чтобы сгенерировать HTTP-запрос и получить возвращаемые значения, необходимо выполнить следующие четыре действия:

1. Создать объект XMLHttpRequest, как было показано в предыдущих примерах.
2. Вызвать метод open() объекта для подготовки запроса.

   Метод open() ожидает получить пять параметров, но обычно достаточно определить два первых: тип метода выполнения запроса (обычно «GET» или «POST») и строку адреса URL (относительный или абсолютный).

   Третий параметр метода open() по умолчанию принимает значение true – это означает асинхронный режим выполнения запроса. Если в этом аргументе передать значение false, запрос будет исполнен в синхронном режиме, то есть исполнение сценария будет приостановлено до момента получения ответа сервера. В большинстве случаев используется асинхронный режим, поэтому вам нужно либо опустить этот параметр, либо передать в нем значение true. Если для вы-

полнения HTTP-запроса требуется аутентификация, тогда в четвертом и пятом аргументах можно передать имя пользователя и пароль.

3. Записать в свойство `onreadystatechange` ссылку на функцию обратного вызова.

   Эта функция будет вызвана при получении ответа на запрос.

4. Отправить HTTP-запрос вызовом метода `send()`.

   Этот метод отправит HTTP-запрос. Если перед этим был выбран асинхронный режим работы, метод `send()` немедленно вернет управление и пользователь сможет продолжить работу со страницей.

Поскольку весь программный код JavaScript работает на стороне клиента, нет никакого надежного способа предотвратить просмотр исходных текстов пользователем. Чтобы хоть как-то разрешить эту ситуацию, можно предпринять некоторые меры, включая отключение реакции на щелчок правой кнопкой мыши или «затуманивание» (*obfuscation*) программного кода JavaScript на стороне клиента. Однако все эти ухищрения вполне преодолимы. Вообще программный код JavaScript не самое безопасное место и потому не следует помещать в него секретную информацию, такую как имя пользователя и пароль. Поэтому четвертый и пятый параметры метода `open()` используются очень редко. Но даже если сценарий получает секретные данные от пользователя, вам следует использовать SSL, чтобы избежать передачи этих данных в простом текстовом виде через незащищенную сеть.

Посредством свойства `onreadystatechange` объект `XMLHttpRequest` предоставляет механизм обратных вызовов, который используется для обработки ответа. Как следует из имени свойства, это должна быть функция, которая определяет реакцию на изменение значения другого свойства объекта `XMLHttpRequest` — `readyState`. Свойство `readyState` указывает на состояние объекта `XMLHttpRequest` и может иметь одно из пяти возможных значений, которые перечислены в табл. 3.1.

Всякий раз, когда изменяется значение `readyState`, вызывается функция, указанная в свойстве `onreadystatechange`. В этой функции сначала необходимо проверить значение `readyState`. Как правило, наибольший интерес представляет значение `4`, которое означает, что был получен полный ответ.

Таблица 3.1. Возможные значения свойства readyState

Значение свойства readyState	Описание
0	Объект не инициализирован
1	Запрос подготовлен
2	Запрос отправлен
3	Ожидание получения ответа
4	Ответ полностью получен

Когда функция вызывается в ответ на изменения в readyState, в игру вступают и другие свойства объекта XMLHttpRequest. Свойство status содержит код HTTP-статуса, полученный в ответ на запрос – если все в порядке, статус имеет значение 200. Свойство statusText содержит соответствующее текстовое описание HTTP-статуса. Например, для статуса с кодом 200 в свойстве statusText будет находиться строка "OK". Однако лучше выполнять проверку свойства status, потому что различные веб-серверы могут возвращать различные текстовые описания для одних и тех же кодов статуса.

Доступ к данным, полученным от сервера, можно получить с помощью двух свойств:

responseText

Возвращает полученные данные в виде строки

responseXML

Возвращает полученные данные в виде XML-документа (подробнее об этом рассказывается в разделе «Объект XMLDocument», далее в этой главе)

Следующий сценарий – это небольшой пример, иллюстрирующий порядок использования объекта XMLHttpRequest. В примере выполняется запрос к странице ASP.NET с именем *ajax.aspx*. На первом этапе с помощью функции getXMLHTTP(), которая была описана ранее, создается объект XMLHttpRequest. В случае успеха (то есть когда функция возвращает значение, отличное от null) выполняется подготовка GET-запроса с параметром sendData=ok (это значение было выбрано лишь для этого примера, а вообще оно может быть любым другим). Затем в свойство onreadystatechange записывается ссылка на функцию, и в заключение запрос отправляется серверу.

```
var XMLHTTP = getXMLHTTP();
if (XMLHTTP != null) {
  XMLHTTP.open("GET", "ajax.aspx?sendData=ok");
  XMLHTTP.onreadystatechange = stateChanged;
  XMLHTTP.send(null);
}
```

Функция stateChanged() может выглядеть примерно так, как показано ниже (обработка ошибок была опущена). Этот фрагмент отображает текст, который был получен в виде ответа от сервера.

```
function stateChanged( ) {
  if (XMLHTTP.readyState == 4 &&
      XMLHTTP.status == 200) {
      window.alert(XMLHTTP.responseText);
  }
}
```

Обратите внимание: функция, которая вызывается в ответ на изменение значения readyState, не имеет входных аргументов. Отсюда следует, что

объект XMLHttpRequest должен быть глобальным, в противном случае вы не сможете получить к нему доступ при асинхронном вызове функции.

Разумеется, на стороне сервера должен иметься некоторый программный код, который возьмет на себя обработку запроса, произведенного объектом XMLHttpRequest. В следующем фрагменте, написанном на языке C#, демонстрируется функция-обработчик Page_Load, которая находится на странице ASP.NET и отвечает на асинхронный запрос, выполняемый объектом XMLHttpRequest:

```
void Page_Load() {
  if (Request.QueryString["sendData"] != null &&
      Request.QueryString["sendData"] == "ok")
  {
    Response.Write("Hello from the server!");
    Response.End();
  }
}
```

В примере 3.1 демонстрируется страница *ajax.aspx*, которая объединяет в себе все составные части (клиентский сценарий и программный код, исполняемый на стороне сервера).

Чтобы опробовать этот пример, вам необходимо запускать страницу *ajax.aspx*, используя веб-сервер (IIS или ASP.NET Development Server, который поставляется вместе с VisualStudio и VWD) с компьютера, на котором установлена платформа .NET Framework.

Анонимные функции JavaScript

Для обеспечения реакции на стороне клиента в ответ на изменение значения readyState вместо ссылки на отдельно определенную функцию можно использовать анонимные функции JavaScript. Эти функции не имеют имен и являются составной частью выражения. В следующем примере демонстрируется, как это может выглядеть:

```
var XMLHTTP = getXMLHTTP();
if (XMLHTTP != null) {
  XMLHTTP.open("GET", "ajax.aspx?sendData=ok");
  XMLHTTP.onreadystatechange = function() {
    if (XMLHTTP.readyState == 4 &&
        XMLHTTP.status == 200) {
      window.alert(XMLHTTP.responseText);
    }
  };
  XMLHTTP.send(null);
}
```

Пример 3.1. Простой пример, объединяющий Ajax и ASP.NET

```
ajax.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  void Page_Load( )
  {
    if (Request.QueryString["sendData"] != null &&
        Request.QueryString["sendData"] == "ok")
    {
      Response.Write("Hello from the server!");
      Response.End( );
    }
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Ajax with ASP.NET</title>

  <script language="Javascript" type="text/javascript">
    function getXMLHTTP() {
      var XMLHTTP = null;
      if (window.ActiveXObject) {
        try {
          XMLHTTP = new ActiveXObject("Msxml2.XMLHTTP");
        } catch (e) {
          try {
            XMLHTTP = new ActiveXObject("Microsoft.XMLHTTP");
          } catch (e) {
          }
        }
      } else if (window.XMLHttpRequest) {
        try {
          XMLHTTP = new XMLHttpRequest( );
        } catch (e) {
        }
      }
      return XMLHTTP;
    }

    var XMLHTTP = getXMLHTTP();
    if (XMLHTTP != null) {
      XMLHTTP.open("GET", "ajax.aspx?sendData=ok");
      XMLHTTP.onreadystatechange = stateChanged;
      XMLHTTP.send(null);
    }

    function stateChanged() {
      if (XMLHTTP.readyState == 4 &&
          XMLHTTP.status == 200) {
```

```
            window.alert(XMLHTTP.responseText);
          }
        }
      </script>
    </head>
    <body>
      <p>Wait and see ...</p>
    </body>
    </html>
```

Если вы увидели текст «Wait and see...», но диалог с надписью «Hello from the server!» так и не появился, перепроверьте еще раз имя файла страницы, оно должно быть *ajax.aspx*.

Как видно на рис. 3.2, 3.3 и 3.4, этот пример прекрасно работает в Internet Explorer, Firefox и Konqueror (в последнем случае ASP.NET работает в среде Mono), то есть в наиболее распространенных броузерах. Этот пример должен работать и в других броузерах при условии, что в них будет включена поддержка JavaScript.

Это одно из немногих мест в книге, где я привожу снимки экрана сразу для нескольких броузеров. Вообще говоря, примеры из этой книги будут работать с платформой ASP.NET 2.0 на стороне сервера и с любым достаточно современным броузером, обладающим поддержкой JavaScript. Большинство снимков экрана в этой книге были сделаны при использовании Firefox 2.0. Если для каких-либо примеров существенное значение будут играть различия между броузерами, это будет отмечено особо.

Если требуется отправлять HTTP-запросы методом POST, необходимо соответствующим образом определить первый параметр метода `open()`.

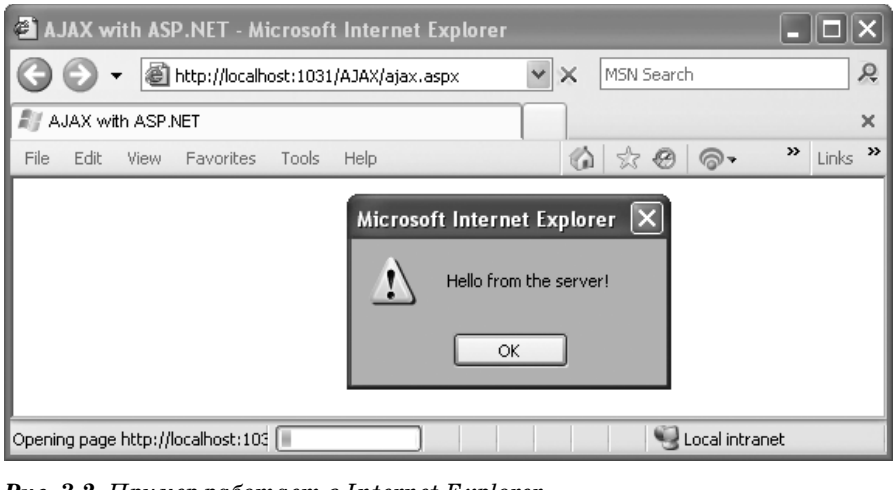


Рис. 3.2. *Пример работает в Internet Explorer*

Рис. 3.3. *Пример работает в Firefox*

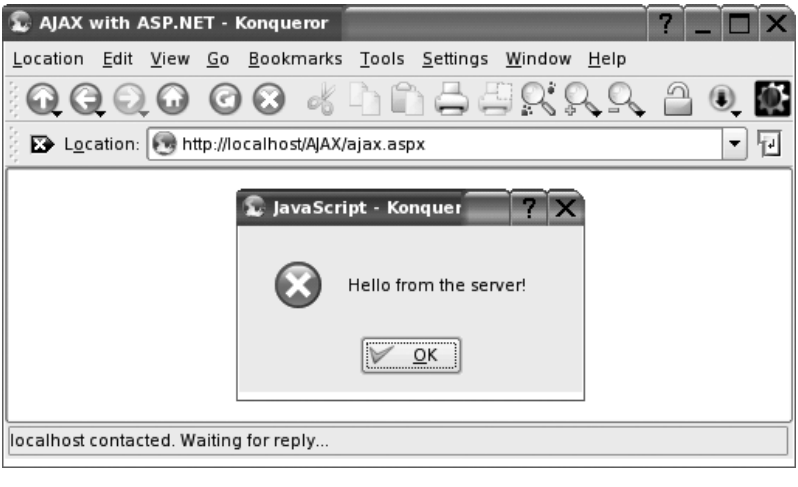


Рис. 3.4. *Пример работает в Konqueror и других броузерах*

Использование метода POST имеет смысл, когда возникает необходимость передачи больших объемов данных, от 500 и более байт (то есть может быть превышена максимальная длина строки URL, которую способен принимать сервер), или когда требуется предотвратить кэширование на прокси-сервере. Отправляемые данные передаются функции `send()` в виде пар имя-значение, при необходимости с URL-кодированием (URL-encoded), как показано в следующем фрагменте:

```
XMLHTTP.open("POST", "ajax.aspx");
XMLHTTP.onreadystatechange = stateChanged;
XMLHTTP.send("sendData=ok&returnValue=123");
```

На стороне сервера для чтения данных, отправленных методом POST, вместо свойства `Request.QueryString`, которое используется для доступа к GET-запросам, следует использовать `Request.Form`.

При обращении к веб-службам, использующим протокол SOAP, можно отправлять текст непосредственно в формате XML, без преобразования его в формат URL. Однако, чтобы такой прием работал в броузерах Safari и Konqueror (что позволит максимально расширить круг посетителей), необходимо явно установить тип содержимого запроса в значение `text/xml`. (Другие броузеры не требуют явного указания типа содержимого.) В следующем фрагменте демонстрируется, как это сделать:

```
XMLHTTP.open("POST", "ajax.aspx");
XMLHTTP.onreadystatechange = stateChanged;
XMLHTTP.setRequestHeader("Content-Type", "text/xml");
XMLHTTP.send("<soap:Envelope>...</soap:Envelope>");
```

Полный справочник по свойствам и методам объекта `XMLHttpRequest` приводится в приложении А.

Теперь немного о безопасности: по умолчанию объект `XMLHttpRequest` обладает доступом только к ресурсам, расположенным в том же домене, где находится страница, содержащая клиентский сценарий. К сожалению, это ограничивает возможности технологии, поскольку нет достаточно простого способа обращения к веб-службам с использованием Ajax, размещенным за пределами вашего домена. Броузеры Mozilla поддерживают возможность доступа к удаленным серверам в других доменах, но при этом они явно запрашивают у пользователя разрешение на получение дополнительных привилегий. На рис. 3.5 показано сообщение с запросом разрешения у пользователя. Однако такой

Рис. 3.5. *Запрос на получение дополнительных привилегий в броузерах Mozilla*

подход имеет свои отрицательные стороны и зависит от типа используемого броузера, поэтому он используется достаточно редко и не рассматривается в данной книге. Отсюда следует, что все приведенные в этой книге HTTP-запросы посылаются серверу, откуда была получена сама страница.

Объект XMLDocument

Свойство `responseXML` объекта `XMLHttpRequest` предполагает возвращение результата обращения к удаленному серверу в виде объекта `XMLDocument`. Для этого необходимо, чтобы программный код на стороне сервера возвращал данные в формате XML; в этом случае сценарий на стороне клиента сможет выполнить разбор этих данных. Получить доступ к данным в формате XML несложно, т. к. вы обладаете полной поддержкой DOM.

JavaScript поддерживает всю необходимую функциональность DOM, чтобы обращаться к конкретным узлам в файле XML или производить обход древовидной структуры XML-документа. Полный перечень свойств и методов объекта `XMLDocument` приводится в приложении B. В следующем примере демонстрируется порядок использования некоторых из них. В этом примере предполагается, что от сервера были получены данные в формате XML, приведенные ниже:

```
<book title="Programming ASP.NET AJAX" author="Christian Wenz">
  <chapters>
    <chapter number="1" title="Introduction" />
    <chapter number="2" title="JavaScript" />
    <chapter number="3" title="Ajax" />
  </chapters>
</book>
```

Очень важно, чтобы при получении от сервера данных в формате XML HTTP-заголовок `Content-Type` ответа имел значение `"text/xml"`. При отсутствии этого заголовка некоторые броузеры (особенно Mozilla и его производные) отказываются выполнять синтаксический анализ полученных данных и возвращают значение `null` в свойстве `responseXML`. Следующий фрагмент на языке C# в странице ASP.NET показывает, как корректно установить тип содержимого ответа:

```
void Page_Load( )
{
  if (Request.QueryString["sendData"] != null &&
      Request.QueryString["sendData"] == "ok")
  {
    string xml = "<book title=\"Programming ASP.NET AJAX\"
    author=\"Christian Wenz\"><chapters><chapter number=\"1\"
title=\"Introduction\"
/><chapter number=\"2\" title=\"JavaScript\" /><chapter
```

```
number=\"3\" title=\"Ajax\"
/></chapters></book>";
    Response.ContentType = "text/xml";
    Response.Write(xml);
    Response.End();
  }
}
```

Клиентский сценарий на JavaScript в этом примере извлекает из XML-документа и выводит такие данные, как атрибуты корневого узла и сведения о различных главах объекта `book`.

Для вывода информации специально не использовался метод `document.write()`, потому что это привело бы к очистке содержимого текущей страницы, кроме того, такой способ более-менее работает в броузерах Mozilla, но, похоже, не поддерживается в Internet Explorer. Вместо этого сценарий создает новые HTML-элементы. Вообще существует два универсальных подхода: помещать данные в существующие элементы или создавать новые.

Чтобы поместить данные в существующий элемент, следует записать их в свойство `innerHTML` элемента. Рассмотрим HTML-документ, в котором имеется следующий элемент `<p>`:

```
<p id="output">Wait and see ...</p>
```

Заменить содержимое этого элемента можно с помощью следующего фрагмента на JavaScript:

```
document.getElementById("output").innerHTML = "Now you see!";
```

Альтернативный вариант заключается в создании новых элементов, которые затем будут добавлены в страницу. Фрагмент ниже создает пустой маркированный список:

```
<ul id="list"></ul>
```

Следующий фрагмент программного кода JavaScript добавит два элемента в этот список:

```
var list = document.getElementById("list");
var listItem1 = document.createElement("li");
var listItemText1 = document.createTextNode("Item 1");
listItem1.appendChild(listItemText1);
list.appendChild(listItem1);
var listItem2 = document.createElement("li");
var listItemText2 = document.createTextNode("Item 2");
listItem2.appendChild(listItemText2);
list.appendChild(listItem2);
```

Вернемся к задаче чтения данных из XML-документа. Фактически существует два подхода к решению этой задачи. Первый, как показано ниже, состоит в том, чтобы получать непосредственный доступ к тегам по их именам, а затем извлекать значения их атрибутов.

```
var xml = XMLHTTP.responseXML;
var root = xml.documentElement;
document.getElementById("output").innerHTML =
  root.getAttribute("title") +
  " by " +
  root.getAttribute("author");

var list = document.getElementById("list");
var chapters = xml.getElementsByTagName("chapter");
for (var i=0; i<chapters.length; i++) {
  var listItem = document.createElement("li");
  var listItemText = document.createTextNode(
    chapters[i].getAttribute("number") +
    ": " +
    chapters[i].getAttribute("title"));
  listItem.appendChild(listItemText);
  list.appendChild(listItem);
}
```

Альтернативный вариант заключается в том, чтобы выполнить обход дерева XML-документа с учетом его структуры. Следующий фрагмент сначала выбирает элемент `<chapters>`, с помощью `getElementsByTagName()`, а затем, двигаясь по дереву документа, отыскивает все вложенные элементы первого элемента `<chapters>`. Когда обнаруживается элемент `<chapter>`, сценарий выводит значения его атрибутов.

```
var xml = XMLHTTP.responseXML;
var root = xml.documentElement;
document.getElementById("output").innerHTML =
  root.getAttribute("title") +
  " by " +
  root.getAttribute("author");

var list = document.getElementById("list");
var chapters = xml.getElementsByTagName("chapters")[0];
for (var i=0; i<chapters.childNodes.length; i++) {
  if (chapters.childNodes[i].nodeName == "chapter") {
    var listItem = document.createElement("li");
    var listItemText = document.createTextNode(
      chapters.childNodes[i].getAttribute("number") +
      ": " +
      chapters.childNodes[i].getAttribute("title"));
    listItem.appendChild(listItemText);
    list.appendChild(listItem);
  }
}
```

Но и это еще не все. Дело в том, что Internet Explorer ведет себя по-разному в некоторых системах (в зависимости от загруженности или быстродействия), особенно при втором подходе. Причина: обращение к объекту `XMLHttpRequest` может происходить «слишком рано» (с точки зрения нашего примера), то есть когда к моменту запуска программно-

го кода разбор HTML-документа еще не завершился. Поэтому необходимо обеспечить запуск механизмов Ajax только после того, как документ будет полностью загружен и разобран. Сделать это можно с помощью анонимной функции, как показано ниже:

```
var XMLHTTP;
window.onload = function( ) {
  XMLHTTP = getXMLHTTP( );
  if (XMLHTTP != null) {
    XMLHTTP.open("GET", "xmldocument2.aspx?sendData=ok");
    XMLHTTP.onreadystatechange = stateChanged;
    XMLHTTP.send(null);
  }
}
```

Предыдущий фрагмент обратится к объекту `XMLHttpRequest` только после того, как HTML-страница будет полностью загружена.

Теперь подведем итоги. В примере 3.2 приводятся полные исходные тексты реализации первого подхода, этот пример также можно загрузить в виде файла *xmldocument.aspx* из репозитария этой книги (*http://www.oreilly.com/catalog/9780596514242*). Реализация второго подхода (здесь не приводится) находится в файле *xmldocument2.aspx*.

Пример 3.2. Чтение и запись данных с использованием JavaScript, Ajax и DOM

```
xmldocument.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  void Page_Load()
  {
    if (Request.QueryString["sendData"] != null &&
        Request.QueryString["sendData"] == "ok")
    {
      string xml = "<book title=\"Programming ASP.NET AJAX\" author=\"
Christian Wenz\"><chapters><chapter number=\"1\"title=\"Introduction\" /
><chapter number=\"2\" title=\"JavaScript\" /><chapter number=\"3\" title=\"
Ajax\"/></chapters></book>";
      Response.ContentType = "text/xml";
      Response.Write(xml);
      Response.End();
    }
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>Ajax with ASP.NET</title>

  <script language="Javascript" type="text/javascript">
```

```
function getXMLHTTP() {
  var XMLHTTP = null;
  if (window.ActiveXObject) {
    try {
      XMLHTTP = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (e) {
      try {
        XMLHTTP = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (e) {
      }
    }
  } else if (window.XMLHttpRequest) {
    try {
      XMLHTTP = new XMLHttpRequest( );
    } catch (e) {
    }
  }
  return XMLHTTP;
}

var XMLHTTP;
window.onload = function() {
  XMLHTTP = getXMLHTTP();
  if (XMLHTTP != null) {
    XMLHTTP.open("GET", "xmldocument.aspx?sendData=ok");
    XMLHTTP.onreadystatechange = stateChanged;
    XMLHTTP.send(null);
  }
}

function stateChanged() {
  if (XMLHTTP.readyState == 4 &&
      XMLHTTP.status == 200) {
    var xml = XMLHTTP.responseXML;
    var root = xml.documentElement;

    document.getElementById("output").innerHTML =
      root.getAttribute("title") +
      " by " +
      root.getAttribute("author");

    var list = document.getElementById("list");
    var chapters = xml.getElementsByTagName("chapter");
    for (var i=0; i<chapters.length; i++) {
      var listItem = document.createElement("li");
      var listItemText = document.createTextNode(
        chapters[i].getAttribute("number") +
        ": " +
        chapters[i].getAttribute("title"));
      listItem.appendChild(listItemText);
      list.appendChild(listItem);
    }
  }
```

```
    }
  </script>
</head>
<body>
  <p id="output">Wait and see ...</p>
  <ul id="list"></ul>
</body>
</html>
```

Результат работы этого сценария приводится на рис. 3.6.

Рис. 3.6. *Данные XML, представленные в удобочитаемом виде*

JSON

Третья основная технология, помимо XMLHttpRequest и XML, часто используемая в Ajax-приложениях, – это JavaScript Object Notation (JSON – представление объектов JavaScript, *http://www.json.org/*). Использование формата JSON позволяет хранить (сериализовать, или переводить в последовательную форму) объекты JavaScript или данные в компактном и понятном виде, при этом не требуя большого объема программного кода JavaScript для чтения и записи данных (выполняется и для XML). Формат JSON основан на использовании часто упускавшейся из виду особенности JavaScript, или, если быть более точным, спецификации языка ECMAScript, известной также как ECMA-262.

Формат JSON используется внутренними механизмами текущей версии ASP.NET AJAX и в общем случае может использоваться для обмена с сервером данными, имеющими сложную структуру. Он является «родным» для JavaScript и позволяет избежать порой достаточно громоздкого процесса разбора формата XML. Следующий фрагмент определяет объект `book` в формате JSON:

```
{"book": {
  "title": "Programming ASP.NET AJAX",
```

```
    "author": "Christian Wenz",
    "chapters": {
      "chapter": [
        {"number": "1", "title": "Introduction"},
        {"number": "2", "title": "JavaScript"},
        {"number": "3", "title": "Ajax"}
      ]
    }
  }}
```

Это те же данные, которые вы видели ранее в этой главе в формате XML. Объект `book` обладает свойствами `title`, `author` и `chapters`. Свойство `chapters` содержит ряд вложенных элементов `chapter`, каждый из которых, в свою очередь, обладает свойствами `number` и `title`. Чтобы было более понятно, ниже приводятся те же данные в формате XML.

```
<book title="Programming ASP.NET AJAX" author="Christian Wenz">
  <chapters>
    <chapter number="1" title="Introduction" />
    <chapter number="2" title="JavaScript" />
    <chapter number="3" title="Ajax" />
  </chapters>
</book>
```

Пример 3.3 демонстрирует, как выполняется разбор данных в формате JSON.

Пример 3.3. Формат JSON упрощает создание объектов

```
json.html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JSON</title>
</head>
<body>

  <script language="JavaScript" type="text/javascript">
    var json = '{"book": { "title": "Programming ASP.NET AJAX", "author": "
Christian Wenz","chapters": {"chapter": [ {"number": "1", "title": "
Introduction"}, {"number": "2", "title": "JavaScript"}, {"number": "3", "
title": "Ajax"} ]} }}';
    var obj = eval("(" + json + ")");
    for (var i=0; i < obj.book.chapters.chapter.length; i++) {
      document.write(
        "<p>" +
        obj.book.chapters.chapter[i].number +
        ": " +
        obj.book.chapters.chapter[i].title +
        "</p>"
      );
    }
```

```
    </script>
  </body>
  </html>
```

Результат работы этого сценария приводится на рис. 3.7.

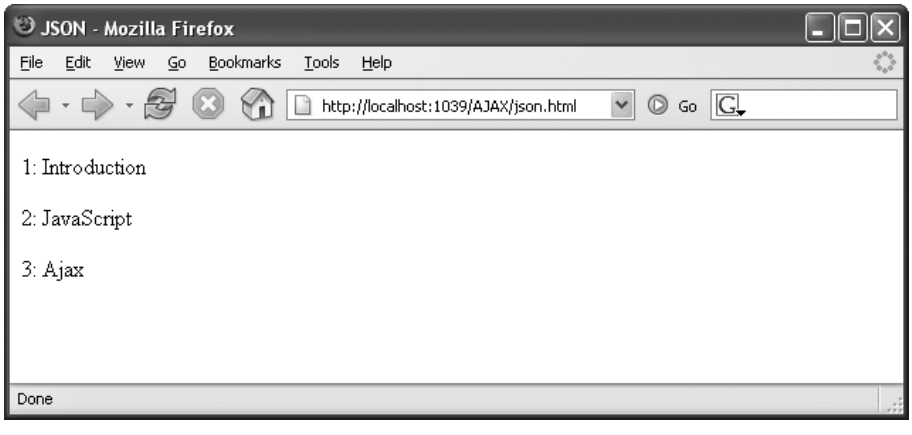


Рис. 3.7. *Результат анализа данных в формате JSON*

Как видно из рис. 3.7, данные в формате JSON – названия трех глав – были выведены броузером. Фигурные скобки, которые можно видеть в примере 3.3, используются для определения свойств объектов, а квадратные скобки – для определения массивов.

Кроме того, вы должны обратить внимание на фрагмент, который с точки зрения безопасности выглядит весьма сомнительно. Следующая строка производит интерпретацию кода JSON во время исполнения:

```
eval("var obj = " + json + ";");
```

В этой строке используется встроенная функция JavaScript – `eval()`, которая производит интерпретацию программного кода во время исполнения. Некоторые программисты считают использование этой функции дурным тоном, но здесь кроется еще большая опасность, которая заключается в том, что функция `eval()` слепо доверяет программному коду, который она интерпретирует. В примере выше данные в формате JSON являются составной частью сценария, поэтому им можно доверять. В Ajax-приложениях данные обычно поступают с того же сервера, откуда была получена сама страница. Безоговорочное доверие к данным, которое проявляет функция `eval()`, может оказаться опасным, особенно если страница, откуда поступают объекты JSON, неподконтрольна вам или имеет место злонамеренный сбой настроек компьютера, на котором выполняется сценарий (то есть действует шпионская программа, которая перенаправляет запросы с одного сервера на другой). Поэтому будьте осторожны в обращении с функцией `eval()` и используйте ее для интерпретации только тех данных, которым вы полностью доверяете.

Использование (и предотвращение) кэширования на стороне клиента

Броузеры часто используют механизм кэширования, поскольку он позволяет им ускорять загрузку страниц. Веб-мастера ничего не имеют против механизма кэширования, так как он помогает снизить нагрузку на сервер. Однако иногда для разработчиков механизм кэширования становится нежелательным. Например, если броузер берет устаревшую версию страницы из кэша, это может существенно усложнить процесс отладки. В мире Ajax это очень распространенная проблема. Однако имеется два простых решения. Первое решение заключается в том, чтобы добавить в URL ложный параметр GET-запроса для объекта `XMLHttpRequest`, который не оказывает никакого влияния на конечный результат, но позволяет избежать кэширования данных, потому что строка URL изменяется от запроса к запросу. Следующий фрагмент демонстрирует, как это можно реализовать на практике.

```
XMLHTTP.open("GET", "xmldocument.aspx?sendData=ok&token="
             + Math.random( ));
```

Этот фрагмент будет добавлять в URL нечто подобное `&token=0.19964476288175226`, придавая ему уникальность. Второе решение заключается в установке в HTTP-запросе дополнительного заголовка `If-Modified-Since`, обращенного к моменту времени в прошлом. Это заставит броузер всякий раз запрашивать новую версию данных; запрос приведен ниже:

```
XML.setRequestHeader(
  "If-Modified-Since",
  "Tuesday, 1 Jan 1980 12:00:00 GMT");
```

Любой из этих приемов может использоваться для облегчения отладки на этапе разработки. Однако на этапе эксплуатации встроенный в броузеры механизм кэширования может способствовать повышению производительности вашего приложения (если это не приводит к появлению нежелательных побочных эффектов), а кэширование на стороне сервера может дать еще больший эффект. Результат всегда зависит от конкретного сценария, для которого применяется механизм кэширования.

Подведение итогов

В этой главе были рассмотрены три технологии, составляющие основу Ajax, из которых наиболее важным является объект `XMLHttpRequest`. Кроме того, вы увидели, как структурированные данные, получаемые

от сервера, обрабатываются средствами JavaScript с использованием XML или JSON.

Для дополнительного чтения

http://www.adaptivepath.com/publications/essays/archives/000385.php

Статья, с которой все начиналось

http://www.json.org/

Неофициальная страница проекта JSON

Perry, Bruce W. Ajax Hacks (O'Reilly)

Советы и рекомендации по разработке Ajax-приложений

McLaughlin, Brett. Head Rush Ajax (O'Reilly)

Быстро развивающееся введение в Ajax

II

Расширения ASP.NET AJAX

Глава 4. Использование расширений JavaScript в ASP.NET AJAX

Глава 5. Веб-службы

Глава 6. UpdatePanel: обновление только части страницы

Глава 7. Использование службы управления профилями в ASP.NET AJAX

Глава 8. Использование службы аутентификации в ASP.NET AJAX

Глава 9. Локализация и глобализация приложений

4

Использование расширений JavaScript в ASP.NET AJAX

Помимо богатых функциональных возможностей Ajax, доступных через удобную в использовании платформу, ASP.NET AJAX предоставляет еще целый ряд дополнений к JavaScript, которые упрощают разработку клиентских сценариев. Среди них такие объектно-ориентированные конструкции, как пространства имен, наследование и интерфейсы, а также клиентские реализации некоторых конструкций .NET, таких как `StringBuilder`. Кроме того, отдельные объекты JavaScript дополнены новыми возможностями.

Псевдонимы и вспомогательные функции ASP.NET AJAX

Включая в веб-страницу `ScriptManager` – элемент управления ASP.NET AJAX, вы автоматически получаете ряд удобных вспомогательных функций и псевдонимов для доступа к важным особенностям JavaScript. Некоторые из этих функций просто позволяют сократить время, затрачиваемое на ввод с клавиатуры. Но другие дают гораздо более существенное преимущество: независимость от типа броузера. Например, Internet Explorer с одной стороны и все современные броузеры – с другой, обеспечивают свой собственный способ организации подписки на обработку событий (глава 2). Платформа ASP.NET AJAX определяет тип броузера и автоматически использует функцию, соответствующую той или иной системе.

Псевдонимы

При создании современных веб-сайтов, использующих JavaScript, разработчики очень часто используют метод `document.getElementById()`. Не-

которые инструментальные средства Ajax предоставляют для такой, достаточно длинной формы записи метода, псевдоним `$()`. В ASP.NET AJAX предпринята попытка обеспечить мирное сосуществование с другими платформами и потому в качестве псевдонима было выбрано новое имя: `$get()`.

Эта замена позволяет сэкономить всего несколько символов, в то время как новые вспомогательные функции, выполняющие обработку событий, имеют гораздо более важное значение. Для связывания функции-обработчика с событием вы можете использовать функцию `$addHandler()`.

```
function $addHandler (element, eventName, handler) {}
```

Функции передается элемент `element`, к которому должен быть подключен обработчик события `handler` (это может быть ссылка на функцию или анонимная функция) для обработки события с именем `eventName` (без префикса «`on`»!). Ниже приводится пример, в котором обеспечивается вывод окна с предупреждением по щелчку на кнопке:

```
$addHandler("Button1", "click", function() { alert("Ouch!"); } );
```

Когда необходимо связать функции-обработчики с разными событиями одного и того же элемента, можно вставить в текст сценария несколько вызовов функции `$addHandler()` или воспользоваться функцией `$addHandlers()`, которой в виде аргументов необходимо передать элемент, массив имен событий и функции-обработчики.

Убрать определенный обработчик события можно с помощью функции `$removeHandler()`, как показано ниже:

```
function $removeHandler(element, eventName, handler) {}
```

Заметьте, что для удаления обработчика необходимо опять передавать имя функции-обработчика. Поэтому в большинстве случаев удобнее будет воспользоваться функцией `$clearHandlers()`, которая удаляет все обработчики событий для заданного элемента:

```
function $clearHandlers(element) {}
```

Команда разработчиков ASP.NET AJAX проделала огромную работу по добавлению в JavaScript расширений ASP.NET, имеющих отношение к жизненному циклу страницы. Сам по себе JavaScript поддерживает единственное событие `load`, чего бывает недостаточно для некоторых приложений. Кроме того, есть еще одно узкое место: это событие происходит в момент, когда полностью завершается загрузка HTML-разметки страницы. Однако страницы, созданные на основе ASP.NET AJAX, загружают несколько внешних библиотек JavaScript. Обычно эти библиотеки еще недоступны к тому моменту, когда разметка HTML уже отображена броузером. Поэтому попытка события `load` запустить какой-либо программный код ASP.NET AJAX происходит слишком рано.

Добавление обработчиков, альтернативный способ

Помимо функций `$addHandler()`, `$addHandlers()` и `$clearHandlers()` платформа ASP.NET AJAX поддерживает специальные шаблоны подключения обработчиков к событиям элементов: методы `add_xxx()`. Например, Sys.Application – это JavaScript-объект в ASP.NET AJAX, который является представлением текущей страницы. Так, чтобы обеспечить запуск некоторого программного кода после полной загрузки страницы, можно воспользоваться следующим фрагментом:

```
Sys.Application.add_load(function() {
  /* ... */
}).
```

Это очень удобно при использовании клиентских классов элементов DOM, которые в настоящее время являются частью пакета ASP.NET AJAX Futures CTP. Более подробно об этом пакете будет рассказываться в главе 15.

Событие `load`, задаваемое ASP.NET AJAX, возникает только после загрузки всех внешних файлов JavaScript. Организовать запуск сценария по этому событию можно двумя способами. Можно воспользоваться методом `Sys.Application.add_load()` (как описано во врезке «Добавление обработчиков, альтернативный способ») или написать свою функцию `pageLoad()`. Когда платформа ASP.NET AJAX определяет, что все внешние файлы были загружены, она вызывает функцию `pageLoad()`, если таковая существует в текущей странице, – очень похоже на то, как ASP.NET, на стороне сервера, вызывает метод `Page_Load()`, если он определен. Этот метод обеспечивает надежный способ запуска использования платформы ASP.NET AJAX в нужный момент.

```
function pageLoad() {
  /* ...*/
}
```

По окончании жизненного цикла страницы, когда пользователь закрывает броузер или переходит по другому адресу URL, возникает событие `unload`. Обработать это событие можно, написав свою функцию с именем `pageUnload()`.

```
function pageUnload() {
  /* ...*/
}
```

Если эта функция определена, платформа ASP.NET AJAX автоматически вызовет ее в нужное время.

Методы элементов DOM

ASP.NET AJAX обеспечивает элементы DOM специализированными методами для использования в таких распространенных случаях, как, например, применение классов CSS. Эти методы определены в классе `Sys.UI.Element`. Для таких стандартных ситуаций, как задание или удаление классов CSS, предлагаемые методы, обслуживающие классы CSS, уменьшают объем ввода, облегчая жизнь разработчика.

`Sys.UI.DomElement.addCssClass(element, className)`

Добавляет класс CSS (`className`) к HTML-элементу `element`.

`Sys.UI.DomElement.containsCssClass(element, className)`

Определяет, содержит ли заданный элемент `element` указанный класс CSS.

`Sys.UI.DomElement.removeCssClass`

Удаляет класс CSS (`className`) из HTML-элемента `element`.

`Sys.UI.DomElement.toggleCssClass(element, className)`

Определяет, содержит ли заданный элемент `element` указанный класс CSS (`className`). Если содержит, то удаляет этот класс, в противном случае добавляет класс CSS. Помимо классов CSS, ASP.NET AJAX предоставляет ряд вспомогательных методов для доступа к таким наиболее часто используемым свойствам HTML-элементов, как ширина, высота и координаты.

`Sys.UI.DomElement.getBounds(element)`

Возвращает объект со свойствами `x`, `y`, `height`, `width`, в которых содержатся координата x, координата y, высота и ширина заданного элемента.

`Sys.UI.DomElement.getLocation(element)`

Возвращает объект со свойствами `x` и `y`, в которых содержатся координаты x и y заданного элемента.

`Sys.UI.DomElement.setLocation(element, x, y)`

Устанавливает координаты x и y для заданного элемента.

Кроме того, в классе `Sys.UI.DomElement` определен также метод `getElementById()`; вы уже с ним (и с его псевдонимом `$get()`) знакомы.

Расширения к существующим объектам JavaScript

В главе 2 упоминалось о добавлении методов к базовым объектам JavaScript. Разработчики ASP.NET AJAX очень широко пользуются этой возможностью. В результате дополнительными характеристиками были дополнены следующие базовые типы JavaScript:

- Array
- Boolean
- Date
- Error
- Number
- Object
- String

Каждое из этих дополнений по отдельности не является чем-то особенным, но все вместе они могут дать существенный выигрыш при написании больших объемов программного кода JavaScript. Не забывайте, что основная идея любой рабочей среды Ajax состоит в том, чтобы минимизировать объем программного кода, который придется писать вручную.

Вместо полного списка новых методов я представлю лишь простой пример. Представим, что нам нужен новый метод Array.forEach(), который будет применять некоторую функцию к каждому элементу заданного массива.

```
function Array$forEach(a, fnct) {
  for (var i = 0; i = a.length; i++) {
    if (typeof(a[i]) != "undefined") {
      fnct.call(null, a[i], i, a);
    }
  }
}
```

Оказывается, что имеется встроенный в ASP.NET AJAX метод forEach(), позволяющий сэкономить время, которое пришлось бы затратить на ввод с клавиатуры, отладку и сопровождение.

Прекрасный обзор новых свойств объектов JavaScript можно найти в файлах форматов PDF и XPS[1], доступных для бесплатного скачивания. Подробнее о доступе к этим файлам говорится в разделе «Для дополнительного чтения» в конце этой главы.

Объектно-ориентированные возможности JavaScript в ASP.NET AJAX

В главе 2 мы узнали, что в JavaScript имеются некоторые особенности, характерные для объектно-ориентированного программирования, но они не вполне соответствуют возможностям ООП, которые имеются в других языках программирования, таких как Visual Basic или C#.

[1] Формат XPS (XML Paper Specification), прежде называвшийся Metro, продвигается Microsoft, в частности, как альтернатива формату PDF. Поддержка XPS-файлов в Windows Vista устанавливается по умолчанию; для Windows XP требуется отдельная установка средств просмотра XPS-документов. – *Примеч. науч. ред.*

Однако недостающие возможности легко можно добавить в JavaScript средствами самого JavaScript, что и было сделано разработчиками ASP.NET AJAX.

Для облегчения разработки в объектно-ориентированном стиле платформа ASP.NET AJAX расширяет JavaScript некоторыми особенностями ООП-типизации, которые рассматриваются в этой главе. В число этих особенностей входят пространства имен, абстрактные классы и интерфейсы. Все эти дополнительные возможности призваны помочь вам проектировать и писать более структурированный программный код для клиентской части. Они могут использоваться не только в Ajax-приложениях, но и при написании любых других сценариев на языке JavaScript.

Пространства имен

Ключевой особенностью объектно-ориентированных расширений JavaScript в ASP.NET AJAX является добавление функциональности пространств имен (*namespaces*). Они позволяют объединять функции в логические группы под общим именем. Помогают избежать конфликтов имен, когда имеется несколько функций с одинаковыми именами, но преследующих различные цели. Спецификация языка JavaScript не предусматривает наличие пространств имен, поэтому данная возможность отсутствует в самом языке. Однако платформа ASP.NET AJAX способна эмулировать пространства имен, используя для этого довольно простой прием. Этот прием заключается в следующем: сначала создается новый класс (который выступает в роли «пространства имен»), а затем создается другой (новый) класс, как свойство класса пространства имен. Это позволяет обращаться к классу с использованием нотации *ИмяКлассаПространстваИмен.ИмяВашегоКласса*.

Одним из базовых классов платформы ASP.NET AJAX является класс `Type`. Этот класс имеет два удобных метода, которые позволяют создавать пространства имен в ASP.NET AJAX:

`Type.registerNamespace(`*`name`*`)`

Регистрирует пространство имен

`Type``.registerClass(`*`name, base type, interface type`*`)`

Регистрирует класс как член пространства имен

Для демонстрации использования этой техники мы создадим пространство имен `OReilly`, которое будет объединять в себе несколько классов, используемых в этой книге. Допустим, что один из этих классов называется `Software` и имеет два свойства: `name` и `vendor`. Для начала необходимо зарегистрировать пространство имен `OReilly`:

```
Type.registerNamespace("OReilly");
```

Затем с помощью следующего фрагмента создадим класс `Software`, как элемент пространства имен `OReilly`:

```
OReilly.Software = function(name, vendor) {
  var _name = (name != null) ? name : "unknown";
  var _vendor = (vendor != null) ? vendor : "unknown";

  this.getName = function() {
    return _name;
  }

  this.setName = function(name) {
    _name = name;
  }

  this.getVendor = function() {
    return _vendor;
  }

  this.setVendor = function(vendor) {
    _vendor = vendor;
  }
}
```

Конструктор класса ожидает получить значения для этих двух свойств. Из соображений безопасности значения сохраняются в виде отдельных переменных, а в самом классе реализованы методы доступа к этим свойствам. Обратите внимание: язык JavaScript не поддерживает частных или защищенных свойств. Поэтому все члены класса являются общедоступными. Сокрытие данных, реализованное здесь, не обеспечивает защиту от несанкционированного доступа – это лишь вспомогательный инструмент, позволяющий структурировать программный код и обеспечить единообразие доступа к данным. Конечно, в большинстве технологий, поддерживающих спецификаторы области видимости `private` и `protected`, имеется возможность доступа к таким свойствам с использованием механизма рефлексии.

Наконец, необходимо зарегистрировать имя `OReilly.Software` как класс, чтобы приложение могло использовать его. Для этого необходимо вызвать метод `registerClass()`. Этот метод может принимать до трех параметров:

name

Имя класса

base type

Базовый тип класса, если таковой существует, в виде ссылки на данный тип

interface type

Тип интерфейса класса, если таковой существует, в виде ссылки на данный тип

Класс `OReilly.Software` не имеет базового типа и не реализует какой-либо интерфейс. Поэтому в данном случае при вызове метода `registerClass()` второй и третий аргументы были опущены:

```
Type.registerClass("OReilly.Software");
```

В ASP.NET AJAX реализовано несколько типов, но один из них вы будете использовать наиболее часто – это `Sys.IDisposable` (потому что с его помощью вы сможете написать свой метод `dispose()`, который будет вызываться автоматически по завершении работы сценария), даже несмотря на то, что в JavaScript имеется простой сборщик мусора. При этом вам не обязательно выполнять реализацию интерфейса. Если вы не используете интерфейс, то для доступа к классу совсем не обязательно регистрировать его с помощью метода `Type.registerClass()`. Для получения дополнительных возможностей вызов этого метода должен производиться обязательно (подробнее об этом говорится в следующих разделах).

Теперь можно создавать экземпляры класса `Software` с помощью ключевого слова `new`, а затем получать и устанавливать значения их свойств. В примере 4.1 создаются два экземпляра: один для Microsoft Internet Explorer, а второй – для Mozilla Foundation Firefox. Кроме того, в примере 4.1 используется одна очень удобная особенность ASP.NET AJAX. После того как обе страницы и все библиотеки, используемые ASP.NET AJAX, будут загружены, вызывается функция `pageLoad()` (если она существует). Не забывайте, что `window.onload` не принимает во внимание загрузку внешних файлов, таких как библиотеки JavaScript. Поэтому при работе с ASP.NET AJAX для решения такого рода задач всегда необходимо использовать функцию `pageLoad()`.

Пример 4.1. Использование пространств имен ASP.NET AJAX

```
ClientNamespaces.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function pageLoad() {
    var s = "";

    Type.registerNamespace("OReilly");
    OReilly.Software = function(name, vendor) {
      var _name = (name != null) ? name : "unknown";
      var _vendor = (vendor != null) ? vendor : "unknown";

      this.getName = function() {
        return _name;
      }

      this.setName = function(name) {
        _name = name;
      }
```

```
        this.getVendor = function() {
          return _vendor;
        }

        this.setVendor = function(vendor) {
          _vendor = vendor;
        }
      }

      Type.registerClass("OReilly.Software");

      var ie = new OReilly.Software("Internet Explorer", "Microsoft");
      s = ie.getName() + " from " + ie.getVendor() + "<br />";

      var ff = new OReilly.Software();
      ff.setName("Firefox");
      ff.setVendor("Mozilla Foundation");
      s += ff.getName() + " from " + ff.getVendor();

      document.getElementById("output").innerHTML = s;
    }
    </script>
  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server">
      </asp:ScriptManager>
      <div id="output">
      </div>
    </form>
  </body>
  </html>
```

На рис. 4.1 приводится результат работы сценария после полной загрузки страницы.

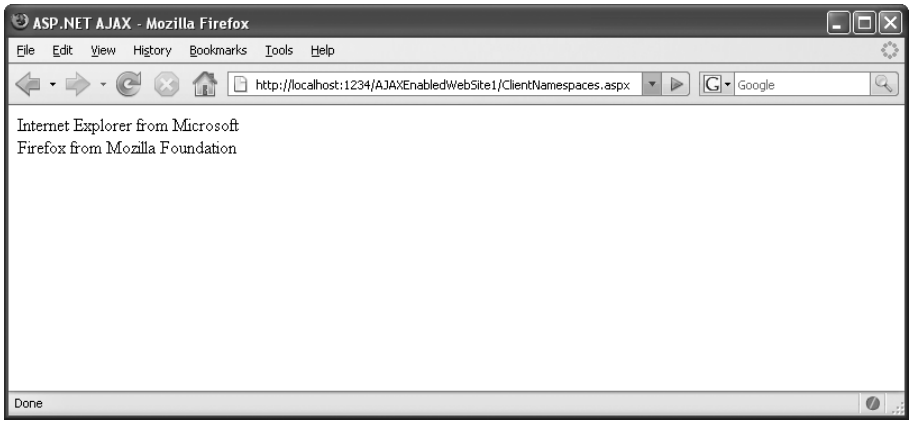


Рис. 4.1. *Создание двух экземпляров класса внутри одного и того же пространства имен*

Классы пространств имен в ASP.NET AJAX не являются настоящими пространствами имен, но тем не менее они позволяют структурировать сложный программный код JavaScript и не требуют для этого больших усилий.

Наследование классов

Как отмечалось в главе 2, ограниченная поддержка механизма наследования классов в JavaScript обеспечивается с помощью свойства `prototype`. Платформа ASP.NET AJAX обеспечивает дополнительный уровень абстракции. Механизм `prototype` поддерживается и для классов пространств имен, которые регистрируются с помощью метода *`ClassName`*`.registerClass()`. Во втором аргументе этого метода можно указать базовый класс. То есть можно задать, от какого класса порожден текущий класс.

Производные классы

Давайте попробуем создать класс, производный от класса `Software`. Броузеры являются одной из разновидностей программного обеспечения, поэтому создадим класс `Browser`. В дополнение к свойствам порождающего класса `Software`, броузеры должны обладать некоторыми дополнительными свойствами. Свойство `isJavaScriptSupported` может давать информацию о том, поддерживает ли данный броузер возможность исполнять сценарии на языке JavaScript.

```
OReilly.Browser = function(name, vendor, isJavaScriptSupported) {
  //...
}
```

Ниже демонстрируется порядок регистрации класса. Обратите внимание на то, как новый класс (строковый параметр) наследует класс `OReilly.Software` (это уже не строка!).

```
OReilly.Browser.registerClass("OReilly.Browser", OReilly.Software);
```

Разумеется, можно было бы снова создать методы для доступа к свойствам `name` и `vendor` и написать программный код конструктора. Но одно из преимуществ наследования (фактически, главное преимущество) состоит в том, что существует возможность повторного использования функциональности базового класса. Поскольку `OReilly.Browser` является наследником `OReilly.Software`, можно использовать уже имеющиеся методы получения и задания значений (то есть сами свойства), а также «частные» члены `_name` и `_vendor`. Нам только потребуется добавить методы доступа и частные члены для нового свойства `isJavaScriptSupported`, как показано ниже:

```
var _isJavaScriptSupported = (isJavaScriptSupported != null) ?
  isJavaScriptSupported : false;

this.getIsJavaScriptSupported = function() {
  return _isJavaScriptSupported;
```

```
}
this.setIsJavaScriptSupported = function(isJavaScriptSupported) {
  _isJavaScriptSupported = isJavaScriptSupported;
}
```

Теперь все, что осталось сделать, – это написать конструктор. Но вместо того, чтобы снова писать его с самого начала, можно воспользоваться конструктором базового класса. Для этой цели ASP.NET AJAX предоставляет метод `initializeBase()`. В первом параметре этому методу передается экземпляр класса, для которого должен быть вызван конструктор базового класса – как правило, используется значение `this`. Второй параметр – это массив аргументов для передачи конструктору базового класса (набор аргументов определяется базовым конструктором). В нашем случае этот массив будет содержать название броузера и производителя.

```
OReilly.Browser.initializeBase(this, new Array(name, vendor));
```

Эту форму записи можно немного сократить, если воспользоваться форматом JSON:

```
OReilly.Browser.initializeBase(this, [name,vendor]);
```

В примере 4.2 приводится программный код, который создает и использует новый производный класс `Browser`.

Пример 4.2. Использование механизма наследования ASP.NET AJAX

```
ClientInheritance.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function pageLoad() {
    var s = "";

    Type.registerNamespace("OReilly");
    OReilly.Software = function(name, vendor) {
      var _name = (name != null) ? name : "unknown";
      var _vendor = (vendor != null) ? vendor : "unknown";

      this.getName = function() {
        return _name;
      }

      this.setName = function(name) {
        _name = name;
      }

      this.getVendor = function() {
```

```
        return _vendor;
      }

      this.setVendor = function(vendor) {
        _vendor = vendor;
      }
    }
    Type.registerClass("OReilly.Software");

    OReilly.Browser = function(name, vendor, isJavaScriptSupported) {
      OReilly.Browser.initializeBase(this, new Array(name, vendor));
        var _isJavaScriptSupported = (isJavaScriptSupported != null) ?
          isJavaScriptSupported : false;

        this.getIsJavaScriptSupported = function() {
        return _isJavaScriptSupported;
      }

      this.setIsJavaScriptSupported = function(isJavaScriptSupported) {
        _isJavaScriptSupported = isJavaScriptSupported;
      }

    }
    OReilly.Browser.registerClass("OReilly.Browser", OReilly.Software);

    var ie = new OReilly.Browser("Internet Explorer", "Microsoft", true);
    s = ie.getName() + " from " + ie.getVendor() +
      (ie.getIsJavaScriptSupported() ? " (w/ JS)" : " (w/o JS)") +
      "<br />";

    var lynx = new OReilly.Browser("Lynx");
    lynx.setIsJavaScriptSupported(false);
    s += lynx.getName() + " from " + lynx.getVendor() +
      (lynx.getIsJavaScriptSupported() ? " (w/ JS)" : " (w/o JS)");

    document.getElementById("output").innerHTML = s;
  }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <div id="output">
    </div>
  </form>
</body>
</html>
```

На рис. 4.2 приводится результат работы сценария после загрузки страницы.

Кстати, если вы зададитесь вопросом, действительно ли у текстового броузера Lynx имеется официальный «производитель». Авторскими правами на этот броузер владеет университет штата Канзас.

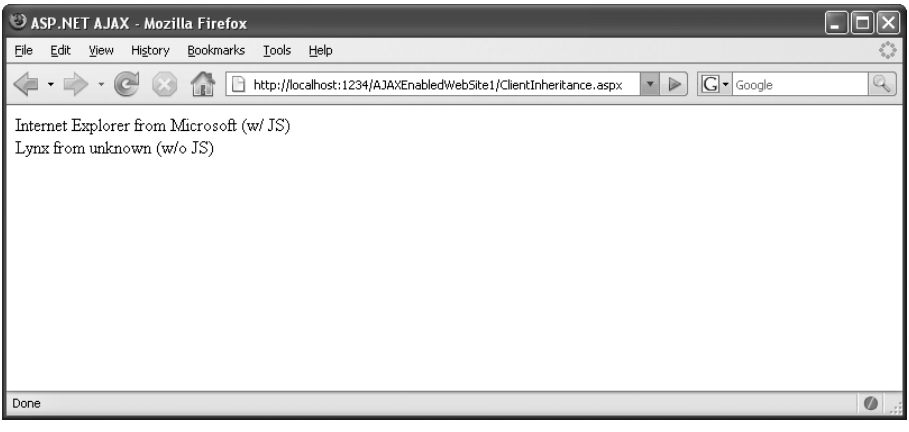


Рис. 4.2. *Создание объектов, производных от одного и того же базового класса*

Доступ к методам базового класса

Когда речь заходит о наследовании классов, сам собой напрашивается вопрос – возможно ли перекрыть родительские методы в производном классе. Ответ на этот вопрос – да. Следующий вопрос: существует ли возможность обратиться к аналогичным методам базового класса (то есть к перекрытым методам)? И снова ответ – да, платформа ASP.NET AJAX позволяет делать это. Для демонстрации этих возможностей добавим в класс `OReilly.Software` метод `toString()`, который будет выводить названия продукта и производителя, хранящиеся в классе. Свойство `prototype` обеспечивает работу механизма наследования; оно же позволяет обращаться к методам базового класса.

```
OReilly.Software.prototype.toString = function() {
  return this.getName() + " from " + this.getVendor();
}
```

Точно так же имеется возможность доступа к свойствам `_name` и `_vendor` как к переменным. Использование методов доступа – это лишь вопрос личных предпочтений. Между этими двумя способами нет никаких функциональных различий.

В классе `OReilly.Browser` можно было бы реализовать похожий метод `toString()`:

```
OReilly.Browser.prototype.toString = function() {
  return this.getName() + " from " + this.getVendor() +
    (this.getIsJavaScriptSupported() ? " (w/ JS)" : " (w/o JS)");
}
```

Однако снова было бы желательно повторно использовать имеющийся программный код, в данном случае – метод `toString()` базового класса. Платформа ASP.NET AJAX предоставляет вспомогательный метод

`callBaseMethod()`, который вызывает метод родительского класса. Этот метод принимает три аргумента:

instance

Экземпляр класса, для которого должен быть вызван метод базового класса (обычно значение `this`)

methodName

Имя метода (в виде строки)

baseArguments

Аргументы метода, если таковые имеются (в виде массива)

В данном случае метод `toString()` класса `OReilly.Browser` может быть реализован следующим образом:

```
OReilly.Browser.prototype.toString = function() {
  return OReilly.Browser.callBaseMethod(this, "toString") +
    (this.getIsJavaScriptSupported() ? " (w/ JS)" : " (w/o JS)");
}
```

Теперь можно сократить фрагмент сценария, который выводит информацию о броузерах, как показано ниже:

```
var s = "";
var ie = new OReilly.Browser("Internet Explorer", "Microsoft", true);
s = ie.toString() + "<br />";

var lynx = new OReilly.Browser("Lynx", null, false);
s += lynx.toString();
document.getElementById("output").innerHTML = s;
```

В примере 4.3 приводится полный листинг.

Пример 4.3. Доступ к методам базового класса

```
ClientBaseMethods.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function pageLoad() {
    var s = "";

    Type.registerNamespace("OReilly");
    OReilly.Software = function(name, vendor) {
      var _name = (name != null) ? name : "unknown";
      var _vendor = (vendor != null) ? vendor : "unknown";

      this.getName = function() {
        return _name;
```

```
      }

      this.setName = function(name) {
        _name = name;
      }

      this.getVendor = function() {
        return _vendor;
      }

      this.setVendor = function(vendor) {
        _vendor = vendor;
      }
    }
    Type.registerClass("OReilly.Software");

    OReilly.Browser = function(name, vendor, isJavaScriptSupported) {
      OReilly.Browser.initializeBase(this, new Array(name, vendor));
      var _isJavaScriptSupported = (isJavaScriptSupported != null) ?
isJavaScriptSupported : false;
      this.getIsJavaScriptSupported = function() {
        return _isJavaScriptSupported;
      }

      this.setIsJavaScriptSupported = function(isJavaScriptSupported) {
        _isJavaScriptSupported = isJavaScriptSupported;
      }
    }
    OReilly.Browser.registerClass("OReilly.Browser", OReilly.Software);

    OReilly.Software.prototype.toString = function() {
      return this.getName() + " from " + this.getVendor();
    }
    OReilly.Browser.prototype.toString = function() {
      return OReilly.Browser.callBaseMethod(this, "toString") +
         (this.getIsJavaScriptSupported() ? " (w/ JS)" : " (w/o JS)");
    };

    var ie = new OReilly.Browser("Internet Explorer", "Microsoft", true);
    s = ie.toString() + "<br />";

    var lynx = new OReilly.Browser("Lynx", null, false);
    s += lynx.toString();

    document.getElementById("output").innerHTML = s;
  }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <div id="output">
    </div>
  </form>
```

```
</body>
</html>
```

Примечательно, что результат работы этого сценария совпадает с тем, что был продемонстрирован на рис. 4.2.

Интерфейсы

Последняя объектно-ориентированная особенность, появившаяся в JavaScript благодаря ASP.NET AJAX, – это интерфейсы. Они вообще не содержат в себе реализации чего бы то ни было, а просто определяют набор членов, которые должны быть реализованы в классах-наследниках. При наследовании интерфейсов вы не получаете никакой реализации для использования. Напротив, вы должны создать методы, объявленные в интерфейсе. Это прекрасный способ разделения структуры класса и его реализации.

Как вы уже, вероятно, догадались, интерфейсы создаются вызовом метода `Type.registerInterface()`. Имя вновь созданного интерфейса затем может быть передано методу `registerClass()` в третьем (необязательном) аргументе. Таким образом, если начинать реализацию приложения с интерфейса, мы могли бы начать со следующего фрагмента:

```
OReilly.IProduct = function() {
  this.toString = Function.abstractMethod;
}
Type.registerInterface("OReilly.IProduct");
```

Ниже приводится определение абстрактного класса `OReilly.Product`. К сожалению, окончательная версия ASP.NET AJAX не поддерживает абстрактные классы (в предварительной версии такая поддержка существовала). Таким образом, можно считать, что чисто технически между обычными и абстрактными классами нет никакой разницы.

В следующем примере класс OReilly.Product объявляет и реализует свойства `name` и `vendor`.

```
OReilly.Product = function(name, vendor) {
  var _name = (name != null) ? name : "unknown";
  var _vendor = (vendor != null) ? vendor : "unknown";

  this.getName = function() {
    return _name;
  }

  this.setName = function(name) {
    _name = name;
  }

  this.getVendor = function() {
    return _vendor;
  }

  this.setVendor = function(vendor) {
```

```
      _vendor = vendor;
    }
  }
  Type.registerClass("OReilly.Product");
```

Следующий на очереди класс — `OReilly.Software`. Поскольку мы не предполагаем создавать экземпляры этого класса (для этих целей мы будем использовать классы, подобные `OReilly.Browser`), то он будет выполнять роль абстрактного класса. Он наследует свойства класса `OReilly.Product` (`name` и `vendor`), а также реализует интерфейс `OReilly.IProduct` (метод `toString()`). После объявления класса мы зарегистрируем его следующим вызовом метода `Type.registerClass()`:

```
OReilly.Software.registerClass("OReilly.Software", OReilly.Product,
OReilly.IProduct);
```

Остальная часть программного кода останется без изменений. Сценарий получился достаточно длинным, поэтому, чтобы не загромождать файл *.aspx*, имеет смысл поместить его в отдельный файл с расширением *.js*. Полный исходный текст приводится в примере 4.4.

Пример 4.4. Использование интерфейса для структурирования программного кода

```
ClientInterface.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
    function pageLoad() {
      var s = "";

    Type.registerNamespace("OReilly");

    OReilly.IProduct = function() {
      this.toString = Function.abstractMethod;
    }
    Type.registerInterface("OReilly.IProduct");

    OReilly.Product = function(name, vendor) {
      var _name = (name != null) ? name : "unknown";
      var _vendor = (vendor != null) ? vendor : "unknown";

      this.getName = function() {
        return _name;
      }

      this.setName = function(name) {
        _name = name;
      }
```

```
  this.getVendor = function() {
    return _vendor;
  }

  this.setVendor = function(vendor) {
    _vendor = vendor;
  }
}
Type.registerClass("OReilly.Product");

OReilly.Software = function(name, vendor) {
  var _name = (name != null) ? name : "unknown";
  var _vendor = (vendor != null) ? vendor : "unknown";
  this.getName = function() {
    return _name;
  }

  this.setName = function(name) {
    _name = name;
  }

  this.getVendor = function() {
    return _vendor;
  }

  this.setVendor = function(vendor) {
    _vendor = vendor;
  }
}
OReilly.Software.registerClass("OReilly.Software", OReilly.Product,
                               OReilly.IProduct);
OReilly.Software.prototype.toString = function() {
  return this.getName() + " from " + this.getVendor();
}

OReilly.Browser = function(name, vendor, isJavaScriptSupported) {
  OReilly.Browser.initializeBase(this, new Array(name, vendor));
  var _isJavaScriptSupported =
    (isJavaScriptSupported != null) ? vendor : false;

  this.getIsJavaScriptSupported = function() {
    return _isJavaScriptSupported;
  }

  this.setIsJavaScriptSupported = function(isJavaScriptSupported) {
    _isJavaScriptSupported = isJavaScriptSupported;
  }
}
OReilly.Browser.registerClass("OReilly.Browser", OReilly.Software);
OReilly.Browser.prototype.toString = function() {
  return OReilly.Browser.callBaseMethod(this, "toString") +
    (this.getIsJavaScriptSupported() ? " (w/ JS)" : " (w/o JS)");
}
var ie = new OReilly.Browser("Internet Explorer", "Microsoft", true);
```

```
    s = ie.toString() + "<br />";
    var lynx = new OReilly.Browser("Lynx", null, false);
    s += lynx.toString();
    document.getElementById("output").innerHTML = s;
  }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <div id="output">
    </div>
  </form>
</body>
</html>
```

Клиентские версии классов .NET

В дополнение к объектно-ориентированным особенностям, привнесенным в JavaScript, ASP.NET AJAX предоставляет реализацию некоторых классов .NET для использования на стороне клиента, достигая следующего результата:

- Функциональность, отсутствующая в JavaScript, реализуется как часть ASP.NET AJAX
- Разработчики, знакомые с .NET и имеющие при этом мало опыта в работе с JavaScript, получают возможность использовать ряд уже знакомых программных конструкций

На мой взгляд, это одна из областей, где развитие будет продолжаться и с каждой новой версией в ASP.NET AJAX будет добавляться все больше новых возможностей, поэтому следующий список классов нельзя рассматривать как исчерпывающий и окончательный. На сегодняшний день доступны две полезные особенности – это `Sys.StringBuilder` и перечисления (enumerations).

Sys.StringBuilder

Одной из новых возможностей, появившихся в .NET 1.0, которая действительно позволяет поднять производительность, стал класс `StringBuilder`. Приложения, как правило, включают в себя массу фрагментов кода, напоминающих тот, что приводится ниже:

```
string s = "", t;
while () {
  t = <value>;
  s += t;
}
```

Проблема кроется в инструкции `s += t`, которая эквивалентна инструкции `s = s + t`. Всякий раз, когда исполняется такая инструкция, в памяти создаются копии `s` и `t`, затем эти копии объединяются и результат сохраняется в переменной `s`. Однако создание копии `s` снижает эффективность этой операции. Поэтому в классе `StringBuilder` используется оптимизированный алгоритм конкатенации строк.

Если говорить о JavaScript, то такой подход не дает сколько-нибудь заметного эффекта в экономии памяти (а реализация класса работает даже несколько медленнее, чем стандартный подход). Кроме того, на стороне клиента вопрос обеспечения высокой производительности стоит не так остро, как на стороне сервера. Тем не менее для поддержания единого стиля программирования на стороне клиента и на стороне сервера вы можете опираться на свои навыки программирования в .NET и использовать класс `StringBuilder` на стороне клиента. В примере 4.5 демонстрируется работа класса `StringBuilder`. С его помощью выполняется конкатенация строк с целью создания некоего подобия шахматной доски в HTML.

Пример 4.5. Использование класса StringBuilder в ASP.NET AJAX

```
ClientStringBuilder.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  window.onload = function() {
    var sb = new Sys.StringBuilder();
    for (var i = 8; i >= 1; i--) {
      for (var j = 97; j <= 104; j++) {
        sb.append(String.fromCharCode(j));
        sb.append(i);
        sb.append(" ");
      }
      sb.appendLine();
      sb.appendLine();
    }
    document.getElementById("output").innerHTML = "<pre>" +
                                                  sb.toString()+"</pre>";
  }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
```

```
    <div id="output"></div>
  </form>
</body>
</html>
```

Встроенная в JavaScript функция `String.fromCharCode()` преобразует код ASCII в соответствующий ему символ, таким образом, вложенный цикл `for` выполняет перебор символов от «`a`» до «`h`». Как показано на рис. 4.3, сценарий из примера 4.5 воссоздает простейшее представление шахматной доски.

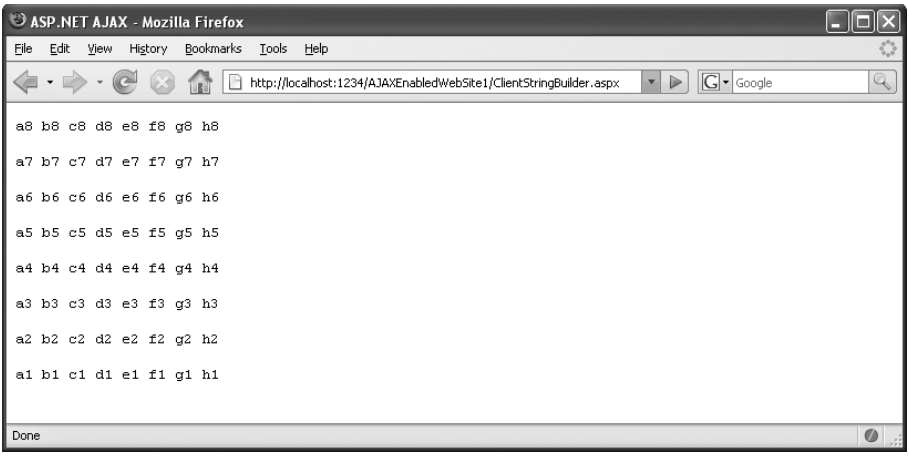


Рис. 4.3. *Шахматная доска (при некоторых допущениях)*

Перечисления

Еще один тип, который эмулируется платформой ASP.NET AJAX в JavaScript, – это `Enum`. Он позволяет создавать собственные перечисления с помощью метода `createEnum()`. В процессе перехода от Atlas к ASP.NET AJAX прикладной интерфейс данной особенности изменился весьма незначительно. В текущем виде класс `Enum` позволяет программисту создавать свои собственные перечисления, как показано в следующем листинге, но не предусматривает возможности выполнять обход значений в цикле. При желании для хранения перечислений можно создать свое пространство имен:

```
Type.registerNamespace("ORA.MyEnums");
```

Затем можно создать свое перечисление, присвоив ему пустую функцию:

```
ORA.MyEnums.Ajax = function( ) {};
```

После этого выполняется объявление значений в перечислении, как показано ниже:

```
ORA.MyEnums.Ajax.prototype = {
  "Asynchronous": 0,
  "JavaScript": 1,
  "and": 2,
  "XML": 3
};
```

И в заключение перечисление необходимо зарегистрировать:

```
ORA.MyEnums.Ajax.registerEnum("ORA.MyEnums.Ajax");
```

В примере 4.6 демонстрируется порядок создания перечисления и его использования.

Пример 4.6. Использование класса Enum платформы ASP.NET AJAX

```
ClientEnum.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function pageLoad() {
    Type.registerNamespace("ORA.MyEnums");
    ORA.MyEnums.Ajax = function() {};
    ORA.MyEnums.Ajax.prototype = {
      "Asynchronous": 0,
      "JavaScript": 1,
      "and": 2,
      "XML": 3
    };
    ORA.MyEnums.Ajax.registerEnum("ORA.MyEnums.Ajax");

    document.getElementById("output").innerHTML +=
      ORA.MyEnums.Ajax.Asynchronous + " " +
      ORA.MyEnums.Ajax.JavaScript + " " +
      ORA.MyEnums.Ajax.and + " " +
      ORA.MyEnums.Ajax.XML;
  }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <div id="output"></div>
  </form>
</body>
</html>
```

Этот пример выводит внутри элемента `<div>` строку `"0 1 2 3"` (ключи элементов перечисления).

Аналогичным образом перечисления используются платформой ASP.NET AJAX для определения кнопок мыши (как показано в следующем фрагменте, который был подредактирован и отформатирован для лучшего восприятия).

```
Sys.UI.MouseButton = function( ) { };
Sys.UI.MouseButton.prototype = {
  leftButton:0,
  middleButton:1,
  rightButton:2
};
Sys.UI.MouseButton.registerEnum("Sys.UI.MouseButton");
```

Подведение итогов

Библиотека клиентских сценариев платформы ASP.NET AJAX реализует несколько удобных особенностей, отсутствующих в стандартном JavaScript, включая объектно-ориентированные возможности и эквиваленты функциональных возможностей .NET Framework на стороне клиента. Эти особенности могут использоваться любым программистом на JavaScript, без оглядки на ASP.NET или серверные функции ASP.NET AJAX.

Для дополнительного чтения

http://www.kevlindev.com/tutorials/javascript/inheritance

Электронное учебное пособие по объектно-ориентированным особенностям в JavaScript

http://aspnetresources.com/blog/ms_ajax_cheat_sheets_batch2.aspx

Шпаргалки по расширениям, которые привносит платформа ASP.NET AJAX в JavaScript

http://ajax.asp.net/docs/ClientReference/Global/default.aspx

Документация с описанием вспомогательных функций и расширений к базовым типам JavaScript

http://quickstarts.asp.net/Futures/ajax/doc/cssselectors.aspx

В пакете ASP.NET AJAX Futures имеются вспомогательные функции JavaScript, позволяющие отбирать элементы на основе правил CSS

5

Веб-службы

Для организации обмена данными между сервером и клиентом в самом первом примере «Hello User» (в главе 1) мы использовали веб-службу. Однако для работы с веб-службами, помимо полного знания JavaScript, необходимо овладеть некоторыми дополнительными навыками. Сюда можно отнести обработку ошибок, встроенные веб-службы (методы веб-службы в текущей странице *.aspx*, которые вызываются другими методами страницы), а также использование веб-служб и JavaScript без помощи .NET Framework.

В этой главе будут рассматриваться некоторые специальные возможности поддержки веб-служб в ASP.NET AJAX, включая обработку ошибок и управление состоянием сеанса. Здесь также будет показано, как работать с веб-службами (не ASP.NET) из JavaScript.

Обработка ошибок

До сих пор, работая с веб-службой, мы предполагали, что программный код, обрабатывающий наши запросы на стороне сервера, всегда работает безошибочно. Однако в действительности при обработке запросов могут возникать исключительные ситуации, которые мы еще не рассматривали.

При работе с веб-службами, исполняемыми на удаленных серверах (в дальнейшем обсуждении мы будем подразумевать серверы из других доменов), разработчики очень часто не включают программный код обработки ошибок. Одна из причин состоит в том, что веб-служба может быть реализована на основе любой технологии, и каждая из технологий предполагает свой собственный способ обработки исключений. Некоторые реализации вообще не возбуждают исключительные ситуации.

Однако в случае ASP.NET AJAX и Ajax порядок взаимодействия с веб-службами несколько иной. Непосредственное обращение к удаленной

службе невозможно из-за ограничений, накладываемых моделью безопасности. По умолчанию JavaScript и объект `XMLHttpRequest` имеют доступ только к URI, находящимся в том же домене, что и текущая страница. Таким образом, при использовании ASP.NET AJAX вы будете обращаться к веб-службам, которые находятся в том же домене, то есть к веб-службам, построенным на базе технологий .NET (или WCF, это платформа следующего поколения организации взаимодействий в среде Windows – Windows Communication Foundation). Как следствие, вам необходимо знать, какая модель исключений используется.

Платформа ASP.NET AJAX позволяет получать доступ из сценариев JavaScript к исключениям, возникшим при работе веб-службы. Для демонстрации этой возможности мы создадим простую веб-службу, выполняющую операцию деления двух чисел. Вы уже наверняка догадались, к чему я клоню: если пользователь попытается выполнить деление на ноль, служба возбудит исключение `DivideByZeroException`. Программный код веб-службы (*MathService.asmx*), которая возбуждает исключение, приводится в примере 5.1. Один важный момент, который следует запомнить: веб-службы, построенные на основе ASP.NET AJAX, обязательно должны иметь атрибуты `[ScriptService]` и `[WebMethod]`.

Пример 5.1. Веб-служба, которая возбуждает исключение

```
MathService.asmx

<%@ WebService Language="C#" Class="MathService" %>

using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://hauser-wenz.de/AspNetAJAX/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.Web.Script.Services.ScriptService]
public class MathService : System.Web.Services.WebService {

    [WebMethod]
    public float DivideNumbers(int a, int b) {
      if (b == 0) {
        throw new DivideByZeroException();
      } else {
        return (float)a / b;
      }
    }
}
```

Теперь создадим страницу, которая будет обращаться к веб-службе. Нам потребуются два поля ввода, куда будут вводиться делимое и делитель. Кроме того, нам потребуются два контейнера: один для вывода результата деления и один для вывода сообщений об ошибках. Кнопка

будет инициировать запуск функции на стороне клиента, а функция, в свою очередь, будет обращаться к веб-службе.

```
<nobr>
  <input type="text" id="a" name="a" size="2" />
  /
  <input type="text" id="b" name="b" size="2" />
  =
  <span id="c" style="width: 50px;" />
</nobr>
<br />
<input
  type="button" value="Divide Numbers" onclick="callService(this.form);" />
<br />
<div id="output" style="width: 600px; height: 300px;">
</div>
```

Что касается серверных элементов управления на странице, нам потребуются два таких элемента: ScriptManager и встроенная в него ссылка на используемую веб-службу.

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Services>
    <asp:ServiceReference Path="MathService.asmx" />
  </Services>
</asp:ScriptManager>
```

Теперь можно обращаться к веб-службе через объект-посредник MathService, который будет создан автоматически. При вызове веб-метода не забывайте передавать ему параметры: первыми должны указываться параметры веб-метода, а затем функция обратного вызова, которая будет вызываться по завершении работы веб-службы.

На этот раз методу DivideNumbers() мы будем передавать более одного параметра. Вслед за функцией, выполняющей обработку результатов вызова веб-службы в случае нормального завершения, мы укажем еще одну функцию обратного вызова. Эта новая функция будет вызываться при появлении ошибок, включая ошибку превышения времени ожидания.

```
function callService(f) {
  document.getElementById("c").innerHTML = "";
  MathService.DivideNumbers(
    parseInt(f.elements["a"].value),
    parseInt(f.elements["b"].value),
    callComplete,
    callError
  );
}
```

Эта функция обработки ошибок будет получать объект с информацией об ошибке, обладающий пятью методами:

`get_exceptionType()`

Возвращает тип исключения

`get_message()`

Возвращает сообщение об ошибке, сопровождающей исключение

`get_stackTrace()`

Возвращает трассировочную информацию об ошибке

`get_statusCode()`

Возвращает код состояния, отправленный сервером

`get_timeOut()`

Возвращает признак превышения предельного времени ожидания

Данная информация выводится в элемент `<div>`, который был создан специально для этих целей:

```
function callError(result) {
  document.getElementById("output").innerHTML =
    "<b>" +
    result.get_exceptionType() +
    "</b>: " +
    result.get_message() +
    "<br />" +
    result.get_stackTrace();
}
```

Остальная часть примера не содержит ничего сложного. В случае нормального завершения веб-службы результат деления будет выводиться в элемент `<span>`. Полный текст примера находится в примере 5.2.

Об отображении (и сокрытии) сообщений об ошибках

Возможность отображения сообщений об ошибках на стороне клиента представляет собой прекрасное подспорье при отладке приложений, но в окончательной версии она может нести определенную угрозу. Сообщения об ошибках могут содержать информацию секретного характера, например строку подключения. Даже если вы не предусматриваете вывод такой информации, ASP.NET AJAX может сам отправлять ее клиенту каким-либо способом. Уменьшить потенциальную угрозу можно двумя способами. Первый заключается в том, чтобы отказаться от вывода на стороне клиента подробных сведений об ошибке, таких как трассировочная информация. Второй способ: при возбуждении исключений на стороне сервера старайтесь встраивать в них как можно меньше информации.

Пример 5.2. Страница, которая выводит информацию об исключении, возбужденном веб-службой MathService.asmx

```
Error.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function callService(f) {
    document.getElementById("c").innerHTML = "";
    document.getElementById("output").innerHTML = "";
    MathService.DivideNumbers(
      parseInt(f.elements["a"].value),
      parseInt(f.elements["b"].value),
      callComplete,
      callError);
  }

  function callComplete(result) {
    document.getElementById("c").innerHTML = result;
  }

  function callError(result) {
    document.getElementById("output").innerHTML =
      "<b>" +
      result.get_exceptionType() +
      "</b>: " +
      result.get_message() +
      "<br />" +
      result.get_stackTrace();
  }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Services>
        <asp:ServiceReference Path="MathService.asmx" />
      </Services>
    </asp:ScriptManager>
    <div>
      <nobr>
        <input type="text" id="a" name="a" size="2" />
        /
        <input type="text" id="b" name="b" size="2" />
        =
        <span id="c" style="width: 50px;"></span>
      </nobr>
```

```
        <br />
        <input type="button" value="Divide Numbers"
               onclick="callService(this.form);" />
        <br />
        <div id="output" style="width: 600px; height: 300px;">
        </div>
      </div>
    </form>
  </body>
  </html>
```

Теперь, выполнив деление числа 6 на 7, вы получите, как и следовало ожидать, число `0.8571429`. Однако если попытаться разделить число 6 на 0, веб-служба выставляет вполне предсказуемое исключение. Результат такой попытки, включая некоторую трассировочную информацию, приводится на рис. 5.1.

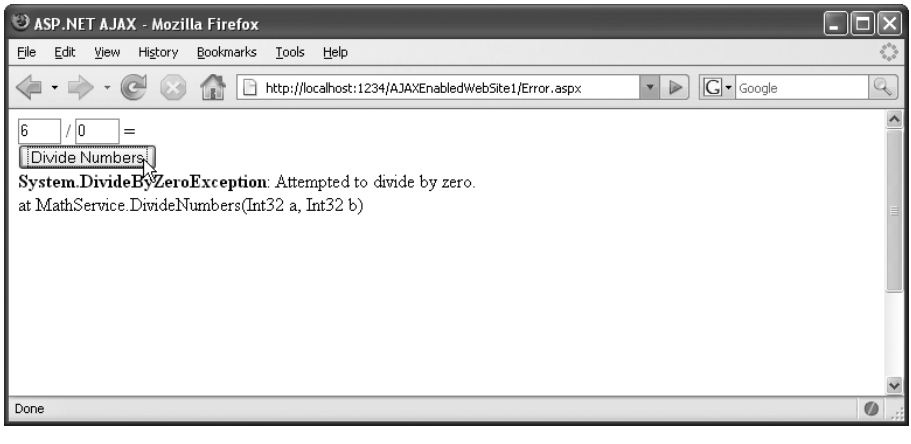


Рис. 5.1. *Информация об исключении*

Методы страницы

Вероятно, вы уже заметили, что размещение всех веб-методов приложения в отдельном файле – занятие несколько утомительное. С архитектурной точки зрения такой способ размещения программного кода выглядит неплохо. Но в случае простых сценариев или приложений (как большинство примеров в этой книге) дополнительные файлы *.asmx* приводят только к загромождению проекта.

Добавив лишь немного дополнительного кода, вы сможете разместить все необходимое в одном месте, а именно – в главном файле *.aspx* (или в файле с реализацией класса). Для этого необходимо выполнить два условия. Первое: с помощью директивы `@ Import` импортировать пространство имен веб-службы в файл страницы, как показано ниже:

```
<%@ Import Namespace="System.Web.Services" %>
```

Второе: разместить в странице программный код веб-методов. Чтобы идентифицировать их как методы веб-службы (или, если быть более точным, как методы, которые работают подобно веб-методам), необходимо предварять их атрибутом [WebMethod], как это делается в файле *.asmx*. Однако при использовании такой поддержки встроенных методов веб-службы в ASP.NET AJAX необходимо соблюдать следующие требования:

- Методы должны отмечаться атрибутом ScriptMethod, определение которого находится в System.Web.Script.Services.
- Метод должен быть объявлен как public.
- Метод должен быть объявлен как static.

Ниже приводится пример простого метода, который соответствует всем требованиям:

```
<script runat="server">
  [WebMethod]
  [System.Web.Script.Services.ScriptMethod]
  public static float DivideNumbers(int a, int b)
  {
    if (b == 0)
    {
      throw new DivideByZeroException();
    }
    else
    {
      return (float)a / b;
    }
  }
</script>
```

Платформа ASP.NET AJAX автоматически отыскивает все такие методы и инкапсулирует их в виде клиентского класса PageMethods. Таким образом, вызов метода должен иметь форму PageMethods.DivideNumbers(), как показано ниже:

```
function callService(f) {
  document.getElementById("c").innerHTML = "";
    PageMethods.DivideNumbers(
      parseInt(f.elements["a"].value),
      parseInt(f.elements["b"].value),
      callComplete,
      callError);
}
```

На заключительном этапе необходимо разрешить вызов встроенных методов веб-службы. В терминологии ASP.NET AJAX они называются «методами страницы», а элемент управления ScriptManager поддерживает свойство, с помощью которого можно разрешить поддержку методов страницы:

```
<asp:ScriptManager ID="a1" runat="server" EnablePageMethods="true" />
```

В примере 5.3 приводится полный текст страницы ASP.NET, где в одном файле размещены как сам текст страницы, так и программный код методов веб-службы.

Пример 5.3. Программный код веб-службы и ASP.NET AJAX в одном файле

```
Inline.aspx

<%@ Page Language="C#" %>
<%@ Import Namespace="System.Web.Services" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  [WebMethod]
   [System.Web.Script.Services.ScriptMethod]
  public static float DivideNumbers(int a, int b)
  {
    if (b == 0)
    {
      throw new DivideByZeroException();
    }
    else
    {
      return (float)a / b;
    }
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function callService(f) {
    document.getElementById("c").innerHTML = "";
    PageMethods.DivideNumbers(
      parseInt(f.elements["a"].value),
      parseInt(f.elements["b"].value),
      callComplete,
      callError);
  }

  function callComplete(result) {
    document.getElementById("c").innerHTML = result;
  }

  function callError(result) {
    document.getElementById("output").innerHTML =
      "<b>" +
      result.get_exceptionType() +
      "</b>: " +
```

```
        result.get_message() +
        "<br />" +
        result.get_stackTrace();
    }
    </script>

  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server"
                         EnablePageMethods="true">
      </asp:ScriptManager>
      <div>
        <nobr>
          <input type="text" id="a" name="a" size="2" />
          /
          <input type="text" id="b" name="b" size="2" />
          = <span id="c" style="width: 50px;"></span>
        </nobr>
        <br />
        <input type="button" value="Divide Numbers"
               onclick="callService(this.form);" />
        <br />
        <div id="output" style="width: 600px; height: 300px;">
        </div>
      </div>
    </form>
  </body>
  </html>
```

На рис. 5.2 показан результат, который будет отображен после того, как выполнена загрузка страницы, введены два числа и выполнен щелчок на кнопке Divide Numbers.

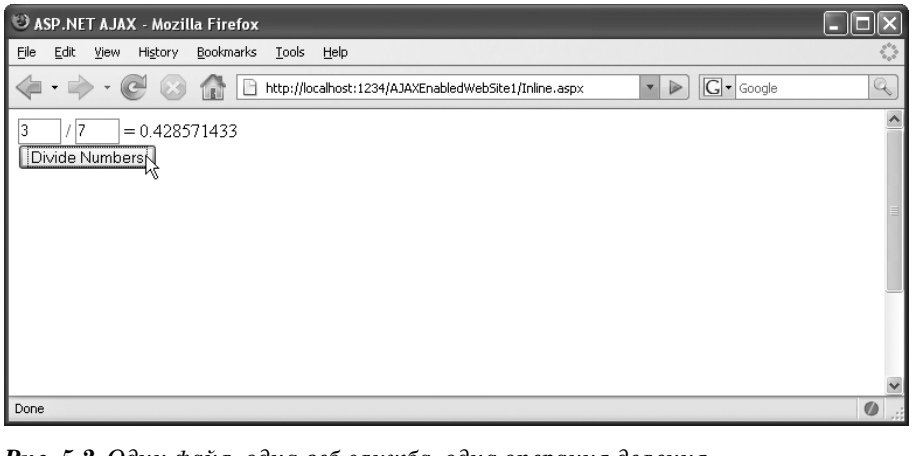


Рис. 5.2. *Один файл, одна веб-служба, одна операция деления*

Управление состоянием сеанса

Иногда веб-службы критикуются за то, что, несмотря на все свое великолепие, данная технология не имеет ничего общего собственно с Веб. Однако поскольку веб-службы .NET эффективно интегрированы с ASP.NET, то при использовании ASP.NET у вас есть возможность создавать сценарии, которые не могли бы быть реализованы только на основе технологии веб-служб.

Например, с помощью веб-служб .NET можно управлять состоянием сеансов. И это состояние, если используется технология Ajax, может быть доступно из приложения ASP.NET AJAX. Так, платформа ASP.NET AJAX предоставляет возможность разным Ajax-приложениям, работающим на одном и том же сервере, совместно использовать информацию об одном и том же пользователе.

Реализация подобной возможности выглядит гораздо проще, чем ее описание. Вся хитрость заключается в свойстве `EnableSession` атрибута `[WebMethod]`, который используется при объявлении веб-методов .NET:

```
[WebMethod(EnableSession=true)]
```

После такого объявления вы получаете возможность обращаться непосредственно к объекту `Session` платформы ASP.NET и читать или записывать данные в него. Поскольку все веб-методы объявляются как статические, вместо простого имени массива `Session` нужно использовать массив `HttpContext.Current.Session`, который доступен только в фактических экземплярах класса `Page`.

В следующем фрагменте иллюстрируются две функции: одна сохраняет текущее время в сеансе, а другая определяет разницу между текущим временем и меткой времени (timestamp), сохраненной в сеансе. Если метки времени в сеансе нет, возвращается значение `-1`.

```
[WebMethod(EnableSession = true)]
[System.Web.Script.Services.ScriptMethod]
public static bool SaveTime()
{
  HttpContext.Current.Session["PageLoaded"] = DateTime.Now;
  return true;
}

[WebMethod(EnableSession = true)]
[System.Web.Script.Services.ScriptMethod]
public static double CalculateDifference()
{
  if (HttpContext.Current.Session["PageLoaded"] == null) {
    return -1;
  } else {
    DateTime then = (DateTime)HttpContext.Current.Session["PageLoaded"];
    TimeSpan diff = DateTime.Now.Subtract(then);
```

```
        return diff.TotalSeconds;
      }
    }
```

Теперь вернемся к приложению, которое выполняет деление двух чисел. Когда будет загружена страница, содержащая фрагмент, приведенный выше, и произойдет вызов метода `SaveTime()`, внутри сеанса будет сохранено текущее время. Во время выполнения операции деления будет вычислена разность между двумя значениями времени. Это даст возможность определить, какой промежуток времени прошел между открытием формы и запросом на выполнение операции деления (конечно, то же самое можно было бы сделать обычными средствами JavaScript, но для демонстрации мы используем обходной путь через сервер).

Следующий фрагмент программного кода на JavaScript сразу после загрузки страницы сохраняет текущее время вызовом метода `SaveTime()` веб-службы. Поскольку возвращаемое значение этого метода не представляет никакого интереса, мы можем использовать функцию обратного вызова, которая ничего не делает.

```
function pageLoad(){
  PageMethods.SaveTime(doNothing, doNothing);
}

function doNothing(result) {
  // ничего не делает :-)
}
```

Как уже было показано ранее, чтобы вызвать метод `CalculateDifference()` веб-службы, необходимо обратиться к методу `callService()`. В следующем фрагменте производится вызов *двух* методов веб-службы. Первый вычисляет разницу между временем загрузки страницы и текущим временем. Второй выполняет ту же арифметическую операцию, что и раньше.

```
function callService(f) {
  document.getElementById("c").innerHTML = "";
  PageMethods.CalculateDifference(
    showDifference,
    callError);
  PageMethods.DivideNumbers(
    parseInt(f.elements["a"].value),
    parseInt(f.elements["b"].value),
    callComplete,
    callError);
}
```

Наконец, необходимо предусмотреть код разметки для отображения разности времен. Для этих целей мы будем использовать контейнер `<div>` со значением атрибута `"id=output"`. Обратите внимание: когда веб-метод возвращает значение `-1`, это означает, что текущее время не было сохранено в сеансе и потому разность времен не может быть вычислена.

```
function showDifference(result) {
  if (result != -1) {
    document.getElementById("output").innerHTML =
      "The form has been open for " + result + " seconds";
  }
}
```

Полный код разметки и текст сценария, которые необходимо реализовать, приводятся в примере 5.4, а все сделанные изменения выделены жирным шрифтом. Обратите внимание: для элемента управления ScriptManager, как и прежде, необходимо установить атрибут EnablePageMethods="true". Если этот атрибут не установлен, методы страницы не будут работать.

Пример 5.4. Управление состоянием сеанса средствами ASP.NET AJAX и ASP.NET

WebServiceSession.aspx

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Web.Services" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  [WebMethod(EnableSession = true)]
  [System.Web.Script.Services.ScriptMethod]
  public static bool SaveTime()
  {
    HttpContext.Current.Session["PageLoaded"] = DateTime.Now;
    return true;
  }

  [WebMethod(EnableSession = true)]
  [System.Web.Script.Services.ScriptMethod]
  public static double CalculateDifference()
  {
    if (HttpContext.Current.Session["PageLoaded"] == null) {
      return -1;
    } else {
      DateTime then = (DateTime)HttpContext.Current.Session["PageLoaded"];
      TimeSpan diff = DateTime.Now.Subtract(then);
      return diff.TotalSeconds;
    }
  }

  [WebMethod]
  [System.Web.Script.Services.ScriptMethod]
  public static float DivideNumbers(int a, int b)
  {
    if (b == 0)
    {
      throw new DivideByZeroException();
```

```
    }
    else
    {
      return (float)a / b;
    }
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function pageLoad() {
    PageMethods.SaveTime(doNothing, doNothing);
  }

  function doNothing(result) {
    // ничего не делает :-)
  }

  function callService(f) {
    document.getElementById("c").innerHTML = "";
    PageMethods.CalculateDifference(
      showDifference,
      callError);
    PageMethods.DivideNumbers(
      parseInt(f.elements["a"].value),
      parseInt(f.elements["b"].value),
      callComplete,
      callError);
  }
  function showDifference(result) {
    if (result != -1) {
      document.getElementById("output").innerHTML =
        "The form has been open for " + result + " seconds";
    }
  }

  function callComplete(result) {
    document.getElementById("c").innerHTML = result;
  }

  function callError(result) {
    if (result == null) {
      window.alert("Error!");
    } else {
      document.getElementById("output").innerHTML =
        "<b>" +
        result.get_exceptionType() +
        "</b>: " +
        result.get_message() +
        "<br />" +
        result.get_stackTrace();
```

```
      }
    }
    </script>
  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server"
        EnablePageMethods="true">
      </asp:ScriptManager>
      <div>
        <nobr>
          <input type="text" id="a" name="a" size="2" />
          /
          <input type="text" id="b" name="b" size="2" />
          = <span id="c" style="width: 50px;"></span>
        </nobr>
        <br />
        <input type="button" value="Divide Numbers"
               onclick="callService(this.form);" />
        <br />
        <div id="output" style="width: 600px; height: 300px;">
        </div>
      </div>
    </form>
  </body>
  </html>
```

При вызове из броузера метода `DivideNumbers()` можно заметить два отличия от предыдущих примеров. Первое: веб-сервер отправляет сеансовый cookie (если в файле *Web.config* вы не задали использование сеансов без cookie). Если броузер настроен на запрос подтверждения перед приемом cookie, вы увидите диалог, как показано на рис. 5.3. Второе: при обращении к веб-службе производится сохранение данных сеанса (рис. 5.4).

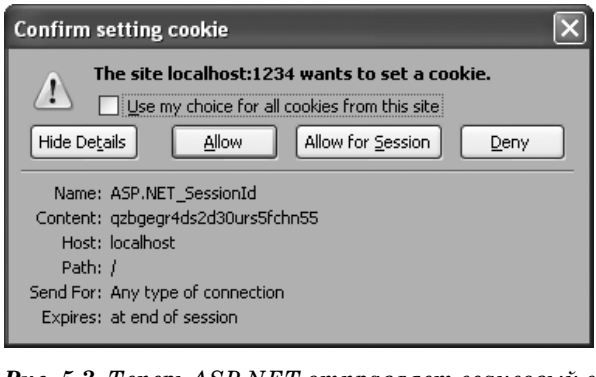


Рис. 5.3. *Теперь ASP.NET отправляет сеансовый cookie от имени страницы*

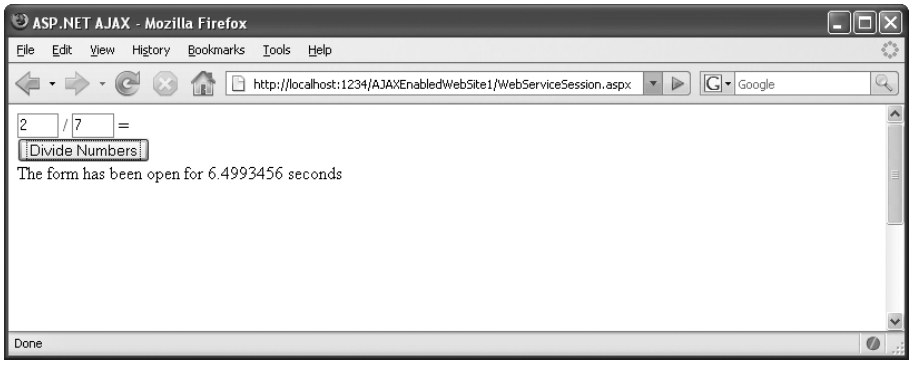


Рис. 5.4. *Использование состояния сеанса для сохранения момента времени, на основе которого вычисляется интервал времени, прошедшего от момента загрузки страницы до обращения к веб-службе*

Обмен данными со сложной структурой

До сих пор в обмене данными между клиентом и сервером участвовали только строки или данные элементарных типов (строки, логические значения, числа). Однако существует возможность организовать обмен данными, имеющими более сложную структуру. По разнообразию типов данных JavaScript не может конкурировать с .NET Framework, тем не менее формат JSON обеспечивает, хотя и рудиментарную, поддержку массивов и объектов.

Платформа ASP.NET AJAX поставляется с возможностью сериализации и десериализации данных, реализуемых в JSON. Попробуем добавить новую функциональную возможность к веб-службе деления двух чисел из примеров 5.1 и 5.2. Для этого мы создадим новый веб-метод, который будет возвращать сразу два значения – результат деления и текущее время на сервере. Для этого мы сначала создадим в файле *MathService.asmx* новый класс, на основе которого будет создаваться возвращаемый объект:

```
public class DivisionData
{
  public float result;
  public string calculationTime;
}
```

Затем добавим метод, что приводится ниже, который создает и возвращает объект:

```
[WebMethod]
public DivisionData ExtendedDivideNumbers(int a, int b) {
  if (b == 0) {
    throw new DivideByZeroException();
  } else {
```

```
      float res = (float)a / b;
      string stamp = DateTime.Now.ToLongTimeString();
      DivisionData d = new DivisionData();
      d.result = res;
      d.calculationTime = stamp;
      return d;
    }
  }
```

Чтобы возвращаемый объект мог быть воспринят сценарием JavaScript, платформа ASP.NET AJAX должна его сериализовать в подходящий формат JSON. Атрибут `GenerateScriptType` (определен в `System.Web.Script.Services` [там же, где определены и атрибуты `ScriptService` и `ScriptMethod`]) сообщает платформе корректный тип представления объекта:

```
[System.Web.Script.Services.GenerateScriptType(typeof(DivisionData))]
```

Это все, что необходимо было добавить на стороне сервера. Исправленный программный код *MathService.asmx* приводится в примере 5.5.

Пример 5.5. Исправленный файл MathService

```
MathService.asmx

<%@ WebService Language="C#" Class="MathService" %>

using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

public class DivisionData
{
  public float result;
  public string calculationTime;
}

[WebService(Namespace = "http://hauser-wenz.de/AspNetAJAX/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.Web.Script.Services.ScriptService]
[System.Web.Script.Services.GenerateScriptType(typeof(DivisionData))]
public class MathService : System.Web.Services.WebService
{

  [WebMethod]
  public float DivideNumbers(int a, int b)
  {
    if (b == 0)
    {
      throw new DivideByZeroException();
    }
    else
    {
      return (float)a / b;
    }
```

```
    }

    [WebMethod]
    public DivisionData ExtendedDivideNumbers(int a, int b)
    {
      if (b == 0)
      {
        throw new DivideByZeroException();
      }
      else
      {
        float res = (float)a / b;
        string stamp = DateTime.Now.ToLongTimeString();
        DivisionData d = new DivisionData();
        d.result = res;
        d.calculationTime = stamp;
        return d;
      }
    }

  }
```

Обратное преобразование объекта DivisionData (десериализация) на стороне клиента выполняется автоматически. Результаты обращения к веб-службе содержатся в свойствах result и calculationTime объекта DivisionData. Дополненный программный код JavaScript, который выполняет обращение к веб-службе, демонстрируется в примере 5.6.

Пример 5.6. Страница, принимающая от веб-службы данные со сложной структурой

```
Complex.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function callService(f) {
    document.getElementById("c").innerHTML = "";
    document.getElementById("output").innerHTML = "";
    MathService.ExtendedDivideNumbers(
      parseInt(f.elements["a"].value),
      parseInt(f.elements["b"].value),
      callComplete,
      callError);
  }

  function callComplete(result) {
    document.getElementById("c").innerHTML =
```

```
        result.result +
        " (calculated at " +
        result.calculationTime +
        ")";
    }

    function callError(result) {
      document.getElementById("output").innerHTML =
        "<b>" +
        result.get_exceptionType() +
        "</b>: " +
        result.get_message() +
        "<br />" +
        result.get_stackTrace();
    }
    </script>

  </head>
  <body>
    <form id="form1" runat="server">
        <asp:ScriptManager ID="ScriptManager1" runat="server">
          <Services>
            <asp:ServiceReference Path="MathService.asmx" />
          </Services>
        </asp:ScriptManager>
        <div>
          <nobr>
          <input type="text" id="a" name="a" size="2" />
          /
          <input type="text" id="b" name="b" size="2" />
          =
          <span id="c" style="width: 50px;"></span>
        </nobr>
        <br />
        <input type="button" value="Divide Numbers"
               onclick="callService(this.form);" />
        <br />
        <div id="output" style="width: 600px; height: 300px;">
        </div>
      </div>
    </form>
  </body>
  </html>
```

Результат работы сценария, который выводит сразу оба значения – результат деления и время, приводится на рис. 5.5.

Если перехватить трафик HTTP, который генерируется этим сценарием, можно увидеть, как выглядят эти данные в формате JSON (рис. 5.6).

К настоящему моменту вы узнали о специфических особенностях веб-служб, предлагаемых платформой ASP.NET AJAX, которые весьма сложно было бы реализовать исключительно средствами JavaScript.

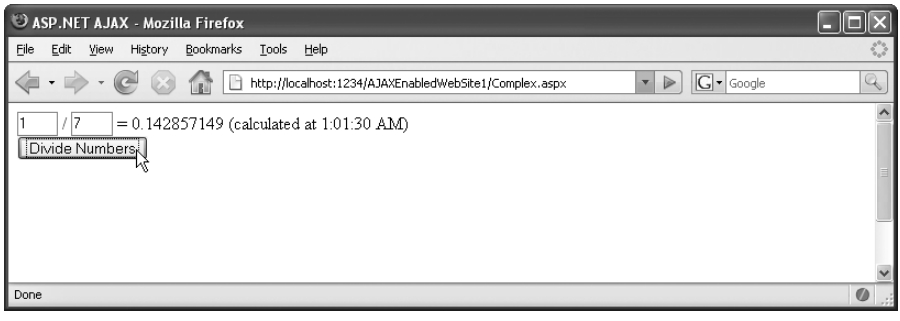


Рис. 5.5. *В окне броузера отображаются полученные от сервера данные со сложной структурой*

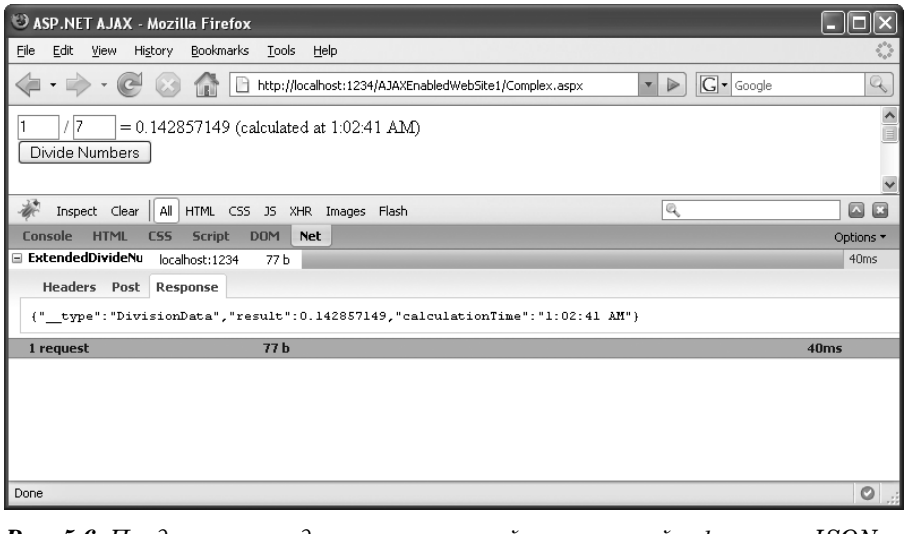


Рис. 5.6. *Представление данных со сложной структурой в формате JSON*

Платформа ASP.NET AJAX прекрасно интегрируется с веб-службами .NET, обеспечивая удобный способ организации взаимодействий между клиентской (JavaScript) и серверной (ASP.NET) технологиями.

Доступ к веб-службам из JavaScript

Автоматические механизмы, которые предоставляет ASP.NET AJAX для доступа к веб-службам, использовать легко и удобно, так как они принимают на себя основной объем работ. Однако существуют ситуации, когда эти механизмы не работают. Например, представьте себе веб-службу (расположенную в том же домене), которая написана не на основе .NET, а с использованием какой-либо другой серверной технологии, такой как PHP или Java. Или представьте себе ситуацию, когда невозможно использовать ASP.NET AJAX по каким-либо причинам (напри-

мер, из-за особой политики компании в отношении программных модулей от третьих фирм или из-за несогласия с лицензионными соглашениями). Так как эта книга не ограничивается использованием ASP.NET AJAX для написания Ajax-приложений на основе ASP.NET, в этом разделе будут рассмотрены альтернативные способы обращения к удаленным веб-службам из JavaScript.

Прежде чем перейти к обсуждению деталей, следует напомнить еще раз, что модель безопасности JavaScript делает невозможным междоменный скриптинг. Это означает, что JavaScript не имеет возможности обращаться к внешним сайтам (при неявном использовании `XMLHttpRequest`).

Существует два возможных способа обратиться к веб-службе из сценария JavaScript. Можно либо сделать ставку на объект `XMLHttpRequest`, либо написать соответствующий SOAP HTTP-запрос и затем выполнить интерпретацию данных, полученных от сервера. Реализовать это достаточно сложно, и при таком подходе легко допустить ошибку. Более надежный подход основан на использовании встроенных технологий или официально распространяемых дополнений к броузерам, которые возьмут решение этой проблемы на себя.

К сожалению, два основных типа броузеров – Internet Explorer и Mozilla (включая Firefox, Epiphany, Camino и другие броузеры) – используют совершенно разные подходы к работе с веб-службами. Таким образом, нам придется исследовать оба пути и рассмотреть эти броузеры по отдельности. В конце этого раздела мы объединим обе модели и создадим сценарий, более или менее независимый от типа броузера.

Веб-службы и Internet Explorer

Несколько лет тому назад в Microsoft были начаты работы по созданию сценария, который позволил бы обращаться к веб-службам из броузера. По сути, этот сценарий создает объект `XMLHttpRequest`, настраивает HTTP-заголовки, необходимые для оформления запроса SOAP, создает тело запроса, ожидает ответ и преобразует его обратно в представление, которое может использовать JavaScript. Кроме того, сценарий в состоянии выполнять интерпретацию описаний веб-служб в формате WSDL (Web Services Description Language – язык описания веб-служб) и генерировать локальный объект-посредник.

Идея довольно проста, чего нельзя сказать о реализации. Окончательная версия сценария (версия 1.0.1.1120) содержала порядка 2300 строк программного кода. К сожалению, в 2002 году Microsoft забросила работу над компонентом. Тем более жаль, так как он до сих пор неплохо работает. К счастью, сценарий можно найти в архивах MSDN, по адресу: *http://msdn.microsoft.com/archive/en-us/samples/internet/behaviors/library/webservice/default.asp*.

Загрузите файл *webservice.htc* и сохраните его в каталоге со сценариями примеров. Расширение *.htc* происходит от «HTML control» (элемент

управления HTML), известный также, как аспект поведения Internet Explorer. Загрузить файл в свое приложение можно с помощью стиля CSS, который поддерживается только в Internet Explorer.

```
<div id="WebService" style="behavior:url(webservice.htc);"></div>
```

Имя, указанное с помощью атрибута id, может затем использоваться в сценарии JavaScript для доступа как к самому элементу управления, так и для обращения к веб-службе, с которой он будет связан.

В данном случае под словом «связывание» подразумевается предоставление ссылки на описание (в формате WSDL) веб-службы, которую предполагается использовать. Для этой цели используется метод useService() из сценария *.htc*. Кроме того, необходимо завести уникальный идентификатор, который позднее будет использоваться для доступа к конкретной веб-службе.

```
WebService.useService("MathService.asmx?WSDL", "MathService");
```

После этого можно вызвать веб-службу. Порядок следования параметров, которые передаются методу callService(), несколько отличается от порядка следования параметров, передаваемых объекту-посреднику, созданному платформой ASP.NET AJAX. Обязательными параметрами являются:

- Ссылка на метод обратного вызова
- Имя вызываемого веб-метода
- Параметры, передаваемые веб-службе

Следует учитывать, что в данном случае не поддерживается возможность обработки ошибок (в отличие от ASP.NET AJAX, где клиентский сценарий получает информацию об исключении).

Вызов, приведенный ниже, выполняет операцию деления с использованием службы MathService:

```
WebService.MathService.callService(
  callComplete,
  "DivideNumbers",
  6, 7);
```

После этого функция обратного вызова принимает результат в виде объекта, в котором свойство value содержит значение, возвращаемое веб-службой:

```
function callComplete(result) {
  document.getElementsById("c").innerHTML = result.value;
}
```

В примере 5.7 приводится полный исходный текст этого примера.

Пример 5.7. Вызов веб-службы из Internet Explorer

```
MathServiceInternetExplorer.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function callService(f) {
    document.getElementById("c").innerHTML = "";
    WebService.useService("MathService.asmx?WSDL", "MathService");
    WebService.MathService.callService(
      callComplete,
      "DivideNumbers",
      f.elements["a"].value, f.elements["b"].value);
  }

  function callComplete(result) {
    document.getElementById("c").innerHTML = result.value;
  }
  </script>

</head>
<body>
  <div id="WebService" style="behavior:url(webservice.htc);">
  </div>
  <form method="post" onsubmit="return false;">
  <div>
    <nobr>
      <input type="text" id="a" name="a" size="2" />
      :
      <input type="text" id="b" name="b" size="2" />
      =
      <span id="c" style="width: 50px;"></span>
    </nobr>
    <br />
    <input type="button" value="Divide Numbers"
           onclick="callService(this.form);" />
    </div>
  </form>
</body>
</html>
```

Если элемент с описанием поведения веб-службы поместить не в начале элемента `<body>`, вы будете получать обескураживающие сообщения об ошибках, вплоть до сообщений о том, что объект с именем `WebService` не определен (хотя при этом вызов `window.alert(WebService)` будет работать без ошибок).

Веб-службы и броузеры Mozilla

Относительно недавние версии броузеров Mozilla также обладают поддержкой веб-служб в виде встроенного в броузер расширения. К сожалению, похоже, что компоненту для работы с веб-службами давно не

уделялось должного внимания со стороны сообщества, но тем не менее он неплохо справляется со своей задачей. Однако он практически недокументирован, а совет по работе с компонентом, который вам удастся отыскать, выглядит достаточно странно. Подход, который будет представлен в этом разделе, делает свою работу, но его реализация требует некоторого объема дополнительного программного кода.

В броузере Mozilla имеется класс SOAPCall, который обслуживает весь комплекс взаимодействий с удаленной службой. Так как он использует протокол SOAP 1.1, вам потребуется настроить заголовок SOAPAction (который для удобства оформлен в виде свойства класса SOAPCall) и определить URL файла веб-службы. Ниже приводится фрагмент, который выполняет эти действия:

```
var soapcall = new SOAPCall( );
soapcall.actionURI = "http://hauser-wenz.de/AspNetAJAX/DivideNumbers";
soapcall.transportURI =
    "http://localhost:1234/AJAXEnabledWebSite1/MathServiceDocEnc.asmx";
```

Значением свойства transportURI должен быть абсолютный адрес URL. Убедитесь, что правильно определили строку URI в своей локальной системе (особенно это относится к номеру порта, если вы используете сервер разработки из Visual Studio/Visual Web Developer).

Все параметры, которые передаются веб-службе, имеют тип SOAPParameter. В конструктор класса первым аргументом передается значение, а вторым – его имя.

```
var p1 = new SOAPParameter(6, "a");
var p2 = new SOAPParameter(7, "b");
```

Теперь самое интересное. Если пропустить следующий шаг, вызов SOAP будет отправлен (и будет получено возвращаемое значение), но служба на стороне сервера получит пустые параметры. В случае нашей службы, выполняющей операцию деления, это приведет к неожидаемой исключительной ситуации «деления на ноль».

Вся хитрость состоит в том, чтобы вручную указать правильную кодировку для целочисленных значений. Для этого нужно загрузить пространство имен, соответствующее типу данных SOAP – integer. Затем параметрам, отправляемым веб-службе, записать в свойство schemaType получившийся тип данных. Ниже приводится фрагмент, который выполняет все эти действия:

```
var senc = new SOAPEncoding( );
assenc = senc.getAssociatedEncoding(
  "http://schemas.xmlsoap.org/soap/encoding/",
  false);
var scoll = assenc.schemaCollection;
var stype = scoll.getType(
  "integer",
  "http://www.w3.org/2001/XMLSchema");
```

```
p1.schemaType = stype;
p2.schemaType = stype;
```

Затем необходимо собрать запрос к веб-службе. Делается это с помощью метода `encode()`, но только после подготовки всех входных параметров, как показано в следующем фрагменте:

```
soapcall.encode(
  0,                            // значение по умолчанию для SOAP 1.1
  "DivideNumbers",              // имя веб-метода
  "http://hauser-wenz.de/AspNetAJAX/", // Пространство имен
  0,                            // количество дополнительных заголовков
  new Array(),                  // дополнительные заголовки
  2,                            // количество параметров
  new Array(p1, p2)             // параметры
);
```

Наконец, нужно выполнить асинхронный вызов веб-службы с помощью метода `asyncInvoke()`. В качестве параметра этому методу передается ссылка на функцию обратного вызова.

```
soapcall.asyncInvoke(callComplete);
```

Функция обратного вызова принимает три параметра:

- Результат работы веб-службы в формате XML
- Объект `SOAPCall` (на тот случай, если вас заинтересует содержимое SOAP-заголовков)
- Код HTTP-статуса вызова

Единственное, что осталось сделать, – это извлечь информацию из полученного XML-документа. Для начала рассмотрим пример XML-документа, полученного в результате вызова веб-службы `MathService` – получить эти данные можно с помощью дополнительного программного обеспечения, такого как инструмент Fiddler (*http://www.fiddler-tool.com/fiddler*) или расширение для Mozilla – Live HTTP Headers (*http://livehttpheaders.mozdev.org/*):

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://
schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <DivideNumbersResponse xmlns="http://hauser-wenz.de/AspNetAJAX/">
      <DivideNumbersResult>0.857142866</DivideNumbersResult>
    </DivideNumbersResponse>
  </soap:Body>
</soap:Envelope>
```

Дополнительные сведения о доступе к HTTP-запросам, которые отправляются Ajax-приложениями, и об отладке Ajax-приложений вообще вы найдете в приложении А.

Исходя из имеющегося XML-представления данных можно заметить, что для доступа к фактическому результату, 0.857142866, необходимо выполнить следующие действия:

- С помощью свойства body получить доступ к элементу <soap:Body>
- С помощью свойства firstChild получить доступ к элементу <DivideNumbersResponse>
- Получить доступ к элементу <DivideNumbersResult>, воспользовавшись свойством firstChild еще раз
- Использовав свойство firstChild в третий раз, получить доступ к следующему текстовому узлу элемента <DivideNumbersResult>
- С помощью свойства data получить доступ к тексту внутри текстового узла

Ниже приводится фрагмент JavaScript, который извлекает результат из ответа, полученного от веб-службы:

```
function callComplete(result, soapcall, status) {
  document.getElementById("c").innerHTML =
    result.body.firstChild.firstChild.firstChild.data;
}
```

Объединив все приведенные фрагменты, вы получите сценарий, который приводится в примере 5.8. Обратите внимание, что для его работы у вас должно быть установлено соединение с Интернетом, чтобы броузер Mozilla мог получить доступ к информации о схемах SOAP.

Пример 5.8. Вызов веб-службы из броузера Mozilla

```
MathServiceMozilla.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function callService(f) {
    document.getElementById("c").innerHTML = "";
    var soapcall = new SOAPCall( );
    soapcall.actionURI = "http://hauser-wenz.de/AspNetAJAX/DivideNumbers";
    soapcall.transportURI =
      "http://localhost:1234/AJAXEnabledWebSite1/MathService.asmx";

    var p1 = new SOAPParameter(parseInt(f.elements["a"].value), "a");
    var p2 = new SOAPParameter(parseInt(f.elements["b"].value), "b");

    var senc = new SOAPEncoding( );
    assenc = senc.getAssociatedEncoding(
      "http://schemas.xmlsoap.org/soap/encoding/",
      false);
    var scoll = assenc.schemaCollection;
```

```
    var stype = scoll.getType(
      "integer",
      "http://www.w3.org/2001/XMLSchema");
    p1.schemaType = stype;
    p2.schemaType = stype;

    soapcall.encode(
      0,                          // значение по умолчанию для SOAP 1.1
      "DivideNumbers",            // имя веб-метода
      "http://hauser-wenz.de/AspNetAJAX/", // Пространство имен
      0,                          // количество дополнительных заголовков
      new Array(),                // дополнительные заголовки
      2,                          // количество параметров
      new Array(p1, p2)           // параметры
    );
    soapcall.asyncInvoke(callComplete);
  }
  function callComplete(result, soapcall, status) {
    document.getElementById("c").innerHTML =
      result.body.firstChild.firstChild.firstChild.data;
  }
  </script>

</head>
<body>
  <form method="post" onsubmit="return false;">
    <div>
      <nobr>
        <input type="text" id="a" name="a" size="2" />
```

Удаленные веб-службы и Mozilla

Модель системы безопасности броузера Mozilla позволяет обращаться к удаленным службам. Однако сценарий будет запрашивать у пользователя разрешение на получение дополнительных привилегий (рис. 5.7). Привилегия, требуемая в данном случае, – `UniversalBrowserRead`; она означает, что броузер получает доступ на чтение по любому адресу (будь то удаленный сервер или локальная файловая система).

```
netscape.security.PrivilegeManager.enablePrivilege(
  "UniversalBrowserRead");
```

С настройками по умолчанию Mozilla, Firefox и другие броузеры этого семейства ограничивают эту привилегию возможностью доступа только к локальным файлам (с использованием протокола *file://*), поэтому такой подход по большей части применим только к Интранет-приложениям. На рис. 5.7 показано сообщение броузеров Mozilla, которое выводится при необходимости получить дополнительные привилегии.

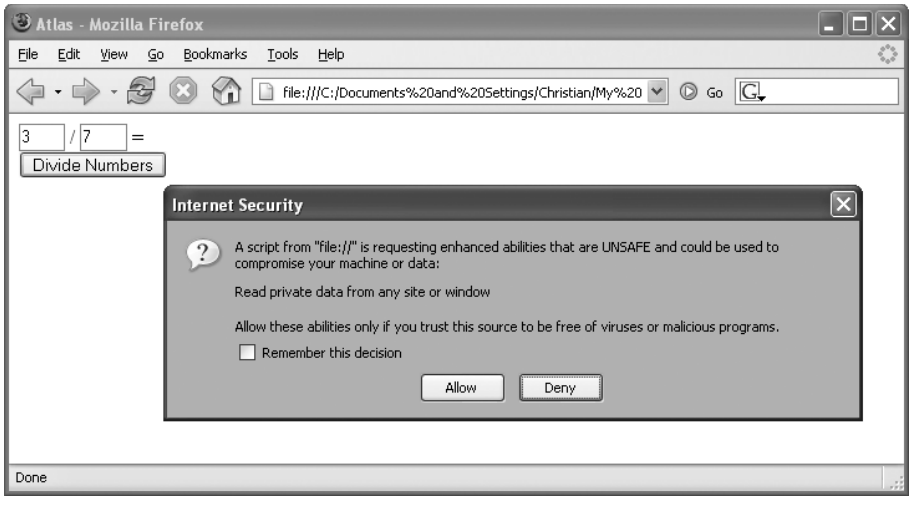


Рис. 5.7. *Броузер Firefox запрашивает дополнительные привилегии для вызова удаленной службы*

```
          :
          <input type="text" id="b" name="b" size="2" />
          =
          <span id="c" style="width: 50px;"></span>
        </nobr>
        <br />
        <input type="button" value="Divide Numbers"
               onclick="callService(this.form);" />
      </div>
    </form>
  </body>
  </html>
```

Веб-службы и броузеры обоих типов

Чтобы завершить обзор методик доступа к веб-службам из JavaScript в Internet Explorer и в броузерах семейства Mozilla, попробуем объединить оба подхода в одной странице. Для начала нужно реализовать способ определения типа броузера. Как говорилось в главе 2, лучший способ заключается в проверке функциональных возможностей броузера, который не зависит от типа броузера. В примере 5.9 выбран подход, использовавшийся в главе 2, когда мы учились создавать объект `XMLHttpRequest`. Суть этого подхода состоит в том, чтобы создать объект, характерный для одного типа броузеров. Если попытка увенчается успехом, можно продолжать выполнять запланированные операции. В случае неудачи используется метод, который работает в других броузерах. Вызовы будут производиться в двух вложенных конструкциях `try...catch`.

В примере 5.9 приводится полный текст разметки и сценарий, необходимые для решения поставленной задачи. Проверьте работоспособность этого примера в разных броузерах и не забудьте установить корректное значение адреса URL используемого сайта для свойства `soapcall.transportURI` (и номер порта, если это необходимо).

Пример 5.9. Вызов веб-службы из Internet Explorer или из Mozilla

```
MathService.htm

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
  function callService(f) {
    document.getElementById("c").innerHTML = "";
    try {
      WebService.useService("MathService.asmx?WSDL", "MathService");
      WebService.MathService.callService(
        callComplete,
        "DivideNumbers",
        parseInt(f.elements["a"].value), parseInt(f.elements["b"].value));
    } catch (e) {
      try {
        var soapcall = new SOAPCall( );
        soapcall.actionURI = "http://hauser-wenz.de/AspNetAJAX/DivideNumbers";
        soapcall.transportURI =
          "http://localhost:1234/AJAXEnabledWebSite1/MathService.asmx";

        var p1 = new SOAPParameter(parseInt(f.elements["a"].value), "a");
        var p2 = new SOAPParameter(parseInt(f.elements["b"].value), "b");

        var senc = new SOAPEncoding();
        assenc = senc.getAssociatedEncoding(
          "http://schemas.xmlsoap.org/soap/encoding/",
          false);
        var scoll = assenc.schemaCollection;
        var stype = scoll.getType(
          "integer",
          "http://www.w3.org/2001/XMLSchema");
        p1.schemaType = stype;
        p2.schemaType = stype;
        soapcall.encode(
          0,                    // значение по умолчанию для SOAP 1.1
          "DivideNumbers",      // имя веб-метода
          "http://hauser-wenz.de/AspNetAJAX/", // Пространство имен
          0,                    // количество дополнительных заголовков
          new Array(),          // дополнительные заголовки
          2,                    // количество параметров
          new Array(p1, p2)     // параметры
```

```
        );
        soapcall.asyncInvoke(callComplete);
      } catch (e) {
        window.alert("Your browser is not supported.");
      }
    }
  }

  function callComplete(result, soapcall, status) {
    if (result.value != null) {
      document.getElementById("c").innerHTML = result.value;
    } else {
      document.getElementById("c").innerHTML =
        result.body.firstChild.firstChild.firstChild.data;
    }
  }
  </script>

</head>
<body>
  <div id="WebService" style="behavior: url(webservice.htc);">
  </div>
  <form method="post" onsubmit="return false;">
    <div>
      <nobr>
        <input type="text" id="a" name="a" size="2" />
        :
        <input type="text" id="b" name="b" size="2" />
        = <span id="c" style="width: 50px;" ></span>
      </nobr>
      <br />
      <input type="button" value="Divide Numbers"
             onclick="callService(this.form);" />
    </div>
  </form>
</body>
</html>
```

Как показано на рис. 5.8 и 5.9[1], пример 5.9 работает в обоих основных броузерах.

[1] Рисунки 5.7, 5.8, 5.9, воспроизведенные из оригинального издания, неточно соответствуют коду примеров 5.8, 5.9 и, видимо, были перенесены из предыдущей книги автора, посвященной Atlas. Впрочем, досадные неточности не затрагивают сути обсуждаемых в этих примерах технологических приемов. Строка заголовка на рис. 5.7, 5.8 и 5.9 вместо «Atlas» должна содержать «ASP.NET AJAX», как указано в теге `<title>` примеров 5.8 и 5.9. В адресной строке броузера на рис. 5.8 и 5.9 предполагается адрес *http://localhost:1234/AJAXEnabledWebSite1/MathService.htm*. На рис. 5.8 отсутствует кнопка «Divide Numbers». – *Примеч. науч. ред.*

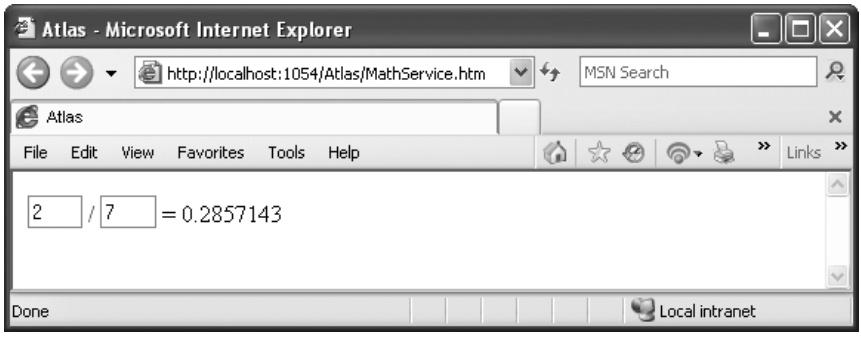


Рис. 5.8. *Теперь сценарий работает в Internet Explorer*

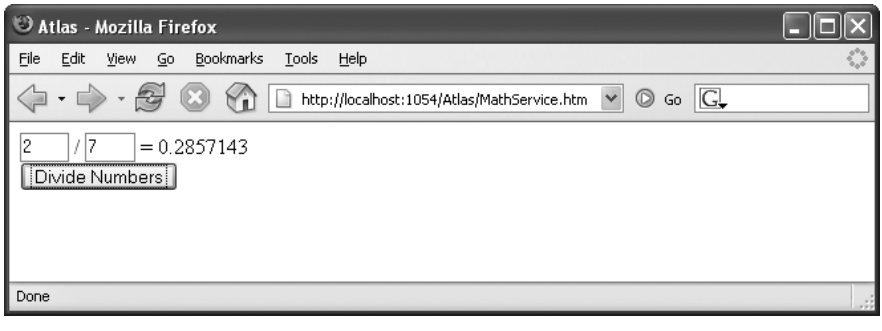


Рис. 5.9. *Сценарий работает и в броузерах семейства Mozilla, таких как Firefox*

Подведение итогов

В этой главе был продемонстрирован ряд приемов работы с веб-службами: сначала мы рассмотрели вопросы обработки ошибок и управления состоянием сеанса, затем мы произвели обмен данными со сложной структурой между клиентом и веб-службой. Наконец, мы рассмотрели способы обращения к веб-службам, построенным не на основе ASP.NET, из JavaScript.

Для дополнительного чтения

http://msdn.microsoft.com/archive/en-us/samples/internet/behaviors/library/webservice/default.asp

Версия файла *webservice.htc* в архиве

http://ajax.asp.net/docs/tutorials/ASPNETAJAXWebServicesTutorials.aspx

Учебное руководство по веб-службам в документации к Microsoft ASP.NET AJAX

6

UpdatePanel: обновление только части страницы

В описание основных достоинств Ajax наверняка можно было бы включить такую формулировку: «изменение части веб-страницы без необходимости обновления ее целиком». В предыдущих главах говорилось о том, как организовать получение данных со стороны сервера и затем с помощью JavaScript и DOM использовать эти данные для заполнения элементов страницы.

Одна из самых замечательных особенностей платформы ASP.NET AJAX заключается в возможности выполнять обновление отдельных частей страницы. Это означает возможность обновления фрагмента без выполнения полного цикла отправки формы и обновления страницы. Дополнительное преимущество состоит в том, что для реализации такого поведения не требуется (от разработчика) создавать программный код JavaScript – все, что необходимо, делается средствами ASP.NET AJAX.

Такое необычное поведение достигается за счет использования элемента управления ASP.NET AJAX – `UpdatePanel`, который позволяет отправлять на сервер содержимое отдельных частей страницы, таких как поля ввода формы. Элемент управления `UpdateControl` может, например, получать от веб-служб биржевые котировки или прогноз погоды и периодически обновлять результаты.

Кроме того, использование этого элемента позволит вам сэкономить время на программировании и отладке. Это, пожалуй, одна из самых ценных особенностей платформы ASP.NET AJAX.

В этой главе вы узнаете, как использовать элемент управления `UpdatePanel`, чтобы избежать необходимости обновления всей страницы и повысить отзывчивость вашего приложения.

Создание обновляемой области страницы

Неудивительно, что главной темой этой главы является элемент управления ASP.NET AJAX – UpdatePanel. Содержимое этого элемента действует как страница в странице и обновляется данными, получаемыми от сервера (разумеется, используя XMLHttpRequest). Однако с точки зрения сервера этот процесс практически не отличается от обычного обновления страницы. При обращении к свойству Page.IsPostBack вы будете получать значение true в случае, когда часть страницы, доступная для изменения, была обновлена данными, полученными от сервера. Все события, которые возбуждаются при обычном обновлении страницы, также возбуждаются и при обновлении содержимого элемента управления UpdatePanel.

Элемент UpdatePanel можно представить себе как плавающий фрейм внутри страницы (внутренний фрейм в странице, который создается с помощью HTML-элемента <iframe>). Содержимое плавающего фрейма загружается и обновляется независимо от остального содержимого страницы. Однако главное преимущество UpdatePanel перед плавающими фреймами состоит в том, что при обновлении продолжают возбуждаться события ASP.NET жизненного цикла страницы, поэтому программно у вас всего одна страница, а не две. Это упрощает программирование и делает архитектуру приложения менее сложной.

Обновление части страницы

В элементе управления UpdatePanel имеется шаблон содержимого (<ContentTemplate>), который, в свою очередь, содержит элементы, образующие панель. Прекрасной иллюстрацией может служить компонент ASP.NET 2.0 – GridView (наследник компонента DataGrid в ASP.NET 1.0). В настройке таких параметров компонента GridView, как порядок сортировки и возможность редактирования с помощью Visual Studio 2005 (включая Visual Web Developer Express Edition), нет ничего сложного. Однако, когда с таблицей выполняются какие-либо операции – сортировка, листание, переключение в режим редактирования или выход из него – производится отправка данных на сервер с последующим принудительным обновлением всей страницы. Поместив компонент GridView в раздел <ContentTemplate> элемента управления UpdatePanel, вы получите ту же функциональность, но в этом случае обновления всей страницы происходить уже не будет. Все необходимое будет сделано платформой ASP.NET AJAX и объектом XMLHttpRequest. (На самом деле компонент GridView поддерживает свойство EnableSortingAndPagingCallbacks, которое реализует аналогичную функциональность. Кроме того, позднее вы узнаете, насколько незначителен объем программного кода, который нужно добавить, чтобы избежать полного обновления страницы при работе с любым компонентом ASP.NET.)

Технически ASP.NET AJAX встраивает содержимое `<ContentTemplate>` в элемент `<div>`. Если вместо блочных элементов (`<div>`) вы предпочитаете использовать внутристрочные HTML-элементы (`<span>`), установите значение `"Inline"` в свойстве `RenderMode` элемента `UpdatePanel` (по умолчанию установлено значение `"Block"`).

Ниже приводится пример размещения компонента `GridView` внутри элемента управления `UpdatePanel` (здесь снова предполагается использование базы данных *AdventureWorks*). Обратите внимание: текст разметки требует, чтобы строка подключения к базе данных *AdventureWorks* была сохранена в файле *Web.config*. При использовании Visual Studio и Visual Web Developer это делается автоматически – при перетаскивании таблицы из окна Database Explorer (Server Explorer в Visual Studio) на страницу в режиме проектирования. Обратите внимание, что в данном примере в окне Database Explorer имя `Purchasing.Vendor` отображается как `Vendor (Purchasing)`.

```
<asp:UpdatePanel id="UpdatePanel1" runat="server">
  <ContentTemplate>
    <asp:GridView ID="GridView1" runat="server"
      AllowPaging="True"
      AllowSorting="True"
      AutoGenerateColumns="False"
      DataKeyNames="VendorID" DataSourceID="SqlDataSource1"
      EmptyDataText="There is no data to display.">
      [...]
    </asp:GridView>
    <asp:SqlDataSource ID="SqlDataSource1"
      runat="server"
      ConnectionString=
        "<%$ ConnectionStrings:AdventureWorksConnectionString1 %>"...>
    [...]
    </asp:SqlDataSource>
  </ContentTemplate>
</asp:UpdatePanel>
```

После передачи значения атрибута `id` элементу управления `UpdatePanel` вы получаете возможность использовать преимущества поддержки SmartTag в режиме проектирования. Хотя единственное доступное в настоящий момент действие SmartTag – это добавление элемента `ScriptManager` в страницу. Главное же удобство заключается в возможности перетаскивать таблицы базы данных из окна Database Explorer в элемент `UpdatePanel`. На рис. 6.1 показан элемент `UpdatePanel` в режиме проектирования.

Для демонстрации того, что полного обновления страницы действительно не происходит, мы добавим на страницу элемент управления `Label`.

```
<asp:Label ID="CurrentTime" runat="server" />
```

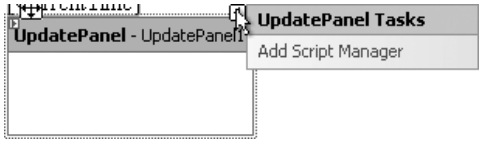


Рис. 6.1. *Элемент управления UpdatePanel в режиме проектирования*

Этот компонент будет отображать текущее время на сервере. Если будет происходить полное обновление страницы, то следующий фрагмент будет обновлять содержимое компонента `Label`.

```
protected void Page_Load(object sender, EventArgs e)
{
  CurrentTime.Text = DateTime.Now.ToLongTimeString();
}
```

Полный исходный текст данного примера приводится в примере 6.1.

Пример 6.1. Компонент GridView, обновление которого происходит без перезагрузки страницы

```
UpdatePanel.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

  protected void Page_Load(object sender, EventArgs e)
  {
    CurrentTime.Text = DateTime.Now.ToLongTimeString();
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <asp:Label ID="CurrentTime" runat="server" />
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>
        <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
                      AllowSorting="True"
                       AutoGenerateColumns="False" DataKeyNames="VendorID"
                      DataSourceID="SqlDataSource1"
                      EmptyDataText="there is no data to display.">
          <Columns>
            <asp:CommandField ShowEditButton="True" />
            <asp:BoundField DataField="VendorID" HeaderText="VendorID"
```

```
                                ReadOnly="True" SortExpression="VendorID" />
            <asp:BoundField DataField="AccountNumber"
                            HeaderText="AccountNumber"
                            SortExpression="AccountNumber" />
            <asp:BoundField DataField="Name" HeaderText="Name"
                            SortExpression="Name"/>
             <asp:BoundField DataField="CreditRating" HeaderText="CreditRating"
                           SortExpression="CreditRating" />
            <asp:CheckBoxField DataField="PreferredVendorStatus"
                               HeaderText="PreferredVendorStatus"
                               SortExpression="PreferredVendorStatus" />
             <asp:CheckBoxField DataField="ActiveFlag" HeaderText="ActiveFlag"
                              SortExpression="ActiveFlag" />
            <asp:BoundField DataField="PurchasingWebServiceURL"
                            HeaderText="PurchasingWebServiceURL"
                            SortExpression="PurchasingWebServiceURL" />
             <asp:BoundField DataField="ModifiedDate" HeaderText="ModifiedDate"
                           SortExpression="ModifiedDate" />
          </Columns>
        </asp:GridView>
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
          ConnectionString=
            "<%$ ConnectionStrings:AdventureWorksConnectionString1 %>"
          DeleteCommand=
          "DELETE FROM [Purchasing].[Vendor] WHERE [VendorID] = @VendorID"
          ProviderName="<%$ ConnectionStrings:AdventureWorksConnection-
                       String1.ProviderName %>"
          SelectCommand="SELECT [VendorID], [AccountNumber], [Name],
                             [CreditRating], [PreferredVendorStatus],
                             [ActiveFlag], [PurchasingWebServiceURL],
                             [ModifiedDate] FROM [Purchasing].[Vendor]"
          UpdateCommand="UPDATE [Purchasing].[Vendor] SET [AccountNumber] =
              @AccountNumber, [Name] = @Name, [CreditRating] = @CreditRating,
[PreferredVendorStatus] = @PreferredVendorStatus, [ActiveFlag] = @ActiveFlag,
[PurchasingWebServiceURL] = @PurchasingWebServiceURL, [ModifiedDate] =
@ModifiedDate WHERE [VendorID] = @VendorID">
          <UpdateParameters>
            <asp:Parameter Name="AccountNumber" Type="String" />
            <asp:Parameter Name="Name" Type="String" />
            <asp:Parameter Name="CreditRating" Type="Byte" />
            <asp:Parameter Name="PreferredVendorStatus" Type="Boolean" />
            <asp:Parameter Name="ActiveFlag" Type="Boolean" />
             <asp:Parameter Name="PurchasingWebServiceURL" Type="String" />
            <asp:Parameter Name="ModifiedDate" Type="DateTime" />
            <asp:Parameter Name="VendorID" Type="Int32" />
          </UpdateParameters>
          <DeleteParameters>
            <asp:Parameter Name="VendorID" Type="Int32" />
          </DeleteParameters>
        </asp:SqlDataSource>
      </ContentTemplate>
    </asp:UpdatePanel>
```

```
    </form>
  </body>
  </html>
```

В предыдущем примере предполагается, что строка подключения к базе данных *AdventureWorks* хранится в файле с настройками *Web.config*, в параметре с именем `AdventureWorksConnectionString1`. Среда Visual Studio выбирает это имя автоматически при перетаскивании таблицы из базы данных *AdventureWorks* на страницу, и эта строка подключения содержится в файле *Web.config* в загружаемых примерах для этой книги. Однако вам при опробовании этого примера в своей системе следует проверить, действительно ли используется именно такое имя.

На рис. 6.2 и 6.3 показано, что компонент `GridView` функционирует, как и ожидалось, но компонент `Label` при этом не изменяется. Это доказывает, что все взаимодействия действительно происходят в фоновом режиме.

Когда в Visual Studio вы, находясь в режиме проектирования, будете перетаскивать таблицу `Vendor` на страницу, в броузере возможно появление сообщения об ошибке от ASP.NET (особенно при использовании версии Visual Studio или VWD, не обновленной до версии SP1). Несмотря на то, что имя `Vendor` является уникальным именем таблицы, оно определено только в пространстве имен базы данных. Правильное имя таблицы будет `Purchasing.Vendor`. Поэтому вам может потребоваться просмотреть весь автоматически сгенерированный код и заменить все вхождения подстроки `[Vendor]` на `[Purchasing].[Vendor]`.

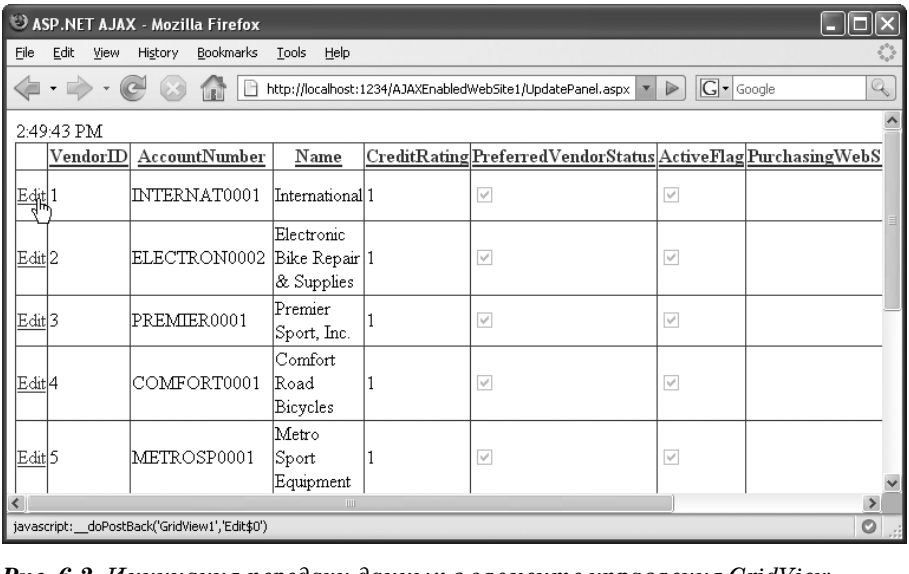


Рис. 6.2. *Инициация передачи данных в элементе управления GridView*

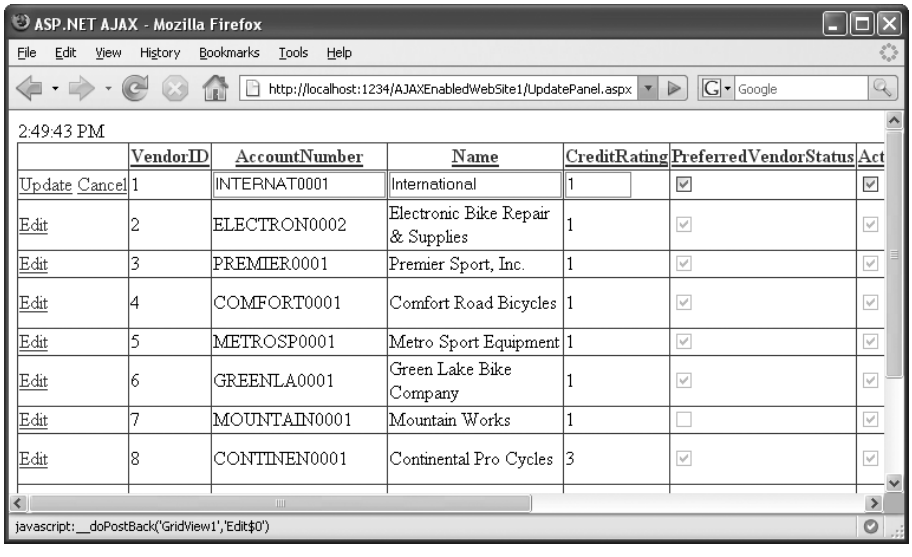


Рис. 6.3. *Передача данных не привела к изменению значения времени над таблицей*

Используйте эти функциональные возможности с осторожностью и всегда помните о возможных побочных эффектах. Например, операция загрузки (upload) файла при исполнении внутри элемента управления `UpdatePanel` сопряжена с некоторыми проблемами.

Если для нужд приложения достаточно будет HTML-элемента `<iframe>`, не следует прибегать к помощи ASP.NET AJAX, поскольку это осложнит отладку приложения, если что-то пойдет не так. В разделе «Для дополнительного чтения» в конце главы приводится ссылка на список компонентов, несовместимых с элементом управления `UpdatePanel`.

Обновление части страницы через определенные интервалы времени

Иногда может потребоваться обновлять содержимое элемента управления `UpdatePanel` не в ответ на действия пользователя, а через определенные интервалы времени. Мне запомнился один сетевой чат, посвященный теме ASP.NET, которым я руководил несколько лет тому назад. Один из посетителей спросил, как можно использовать в страницах ASP.NET компонент `Timer`, который он нашел в Visual Studio. Я ответил на вопрос, объяснив суть модели клиент-сервер, и описал средства, которыми обладает JavaScript для организации временных задержек.

Теперь, обладая опытом работы с платформой ASP.NET AJAX в течение нескольких лет, я мог бы дать другой ответ. Компонент `Timer`, входящий в состав ASP.NET AJAX, переводит работу со связанными

с ним методами JavaScript – window.setTimeout() и window.setInterval() – на уровень абстракции. Вы определяете длительность интервала (в миллисекундах), как того требуют методы JavaScript, и после этого через заданные интервалы времени начинает возникать событие Tick. Ниже приводится объявление элемента Timer, который генерирует событие Tick каждые пять секунд:

```
<asp:Timer Interval="5000" runat="server" />
```

Теперь, с помощью этого таймера вы можете запускать обновление UpdatePanel всякий раз, когда возникает событие Tick, другими словами, – через регулярные интервалы времени. Это может быть сделано программным способом, но, как обычно, существует и декларативный способ.

Внутри элемента управления UpdatePanel для определения реакции на события можно использовать элемент <Triggers>. Всякий раз, когда будет происходить определенное событие, UpdatePanel будет запускать цикл регенерации. Для этого должны быть установлены следующие два свойства:

ControlID

Имя элемента управления, который запускает событие

EventName

Имя события, которое используется для запуска регенерации

Существует два режима обработки события:

AsyncPostBackTrigger

Обработка выполняется асинхронно – предпочтительный вариант

PostBackTrigger

Обработка выполняется синхронно – такого способа следует по возможности избегать

Для демонстрации использования таймера с элементом управления UpdatePanel мы воспользуемся компонентом Label из предыдущего примера для отображения текущего времени в элементе UpdatePanel. Это положение влечет за собой следующее: когда страница будет загружаться первый раз, компонент Label будет отображать текущее время.

Каждые пять секунд компонент Timer будет генерировать событие Tick, которое будет приводить к обновлению содержимого элемента управления UpdatePanel (это будет производиться автоматически, средствами ASP.NET AJAX).

Полный исходный текст этого примера приводится в примере 6.2.

Пример 6.2. Обновление содержимого панели через заданные интервалы времени

UpdatePanelTimer.aspx

```
<%@ Page Language="C#" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

  protected void Page_Load(object sender, EventArgs e)
  {
    CurrentTime.Text = DateTime.Now.ToLongTimeString();
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <asp:Timer ID="FiveSeconds" Interval="5000" runat="server" />
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>
        <asp:Label ID="CurrentTime" runat="server" />
      </ContentTemplate>
      <Triggers>
       <asp:AsyncPostBackTrigger ControlID="FiveSeconds" EventName="Tick" />
      </Triggers>
    </asp:UpdatePanel>
  </form>
</body>
</html>
```

На рис. 6.4 показан результат работы примера, когда после загрузки страницы происходит обновление показаний времени через заданные интервалы.

Программное обновление части страницы через определенные интервалы времени

Из всех методов, которыми обладает элемент управления `UpdatePanel`, наиболее важным является `Update()`. Как следует из названия, этот метод производит обновление панели. Одно из возможных применений этого метода – обработка события `Tick` элемента управления `Timer`:

```
<asp:Timer ID="FiveSeconds" Interval="5000"
  OnTick="UpdateContents"
  runat="server" />
```

После этого остается только написать на стороне сервера самый обычный обработчик события ASP.NET, который будет вызывать метод `Update()` элемента управления `UpdatePanel`:

```
protected void UpdateContents(object o, EventArgs e)
```

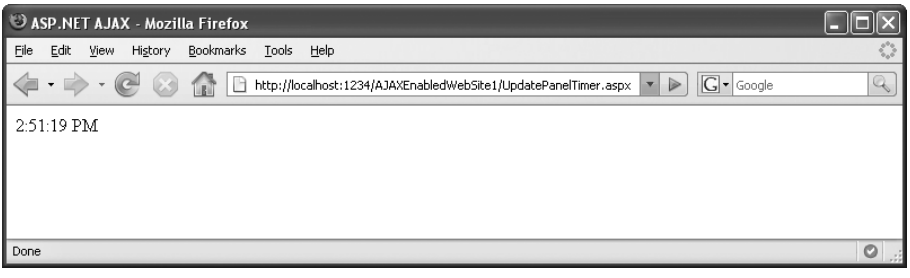


Рис. 6.4. *Показания времени обновляются каждые пять секунд*

```
{
  if (new Random().Next(0, 4) == 1) {
    UpdatePanel1.Update();
  }
}
```

Этот фрагмент обновляет панель (и изображение на экране, которое видит пользователь) в среднем при каждом четвертом запросе. Компонент `Timer` вызывает обработчик события каждые пять секунд. После этого программный код случайным образом определяет, следует ли выполнить фактическое обновление панели в текущем вызове. В реальности вы бы скорее сначала проверили, изменились ли определенные данные (в базе данных или в файле), а затем при необходимости включали обновление.

Элемент управления ASP.NET AJAX – `UpdatePanel` поддерживает два режима работы, установить которые можно с помощью атрибута `UpdateMode`:

`Always`

Содержимое элемента управления `UpdatePanel` будет обновляться после каждой передачи данных (поведение по умолчанию).

`Conditional`

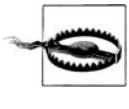
Содержимое элемента управления `UpdatePanel` будет обновляться только при обработке событий, то есть при обращении к методу `Update()` (как в данном примере) или при обновлении родительской панели, в которую вложен данный элемент `UpdatePanel`.

Вообще в режиме `Conditional` между сервером и клиентом передается меньше данных, что способствует повышению производительности элемента `UpdatePanel`. По этой причине всегда, когда это возможно (то есть когда обновление панели происходит в результате обработки событий или при непосредственном обращении к методу `Update()`), используйте `UpdateMode="Conditional"`.

В примере 6.3 демонстрируется полный программный код, управляющий обновлением элемента `UpdatePanel`, обновляемый через случайные интервалы времени.

Пример 6.3. Программное обновление панели

```
UpdatePanelTimerCode.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  private void UpdateContents(object o, EventArgs e)
  {
    if (new Random().Next(0, 4) == 1)
    {
      UpdatePanel1.Update();
    }
  }

  protected void Page_Load(object sender, EventArgs e)
  {
    CurrentTime.Text = DateTime.Now.ToLongTimeString();
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <asp:UpdatePanel ID="UpdatePanel1" runat="server"
         UpdateMode="Conditional">
      <ContentTemplate>
        <asp:Timer ID="FiveSeconds" Interval="5000"
             OnTick="UpdateContents"
             runat="server" />
        <asp:Label ID="CurrentTime" runat="server" />
      </ContentTemplate>
    </asp:UpdatePanel>
  </form>
</body>
</html>
```



Не забывайте, что каждое событие `Tick` порождает сетевой трафик, который может привести к существенному увеличению нагрузки на ваш сервер. Это одно из отличий между элементом управления `Timer` платформы ASP.NET AJAX и элементом управления `Timer` в WinForms. Поэтому проявляйте благоразумие при работе с `<asp:Timer>` и используйте его не чаще, чем это необходимо.

Отображение экранной заставки в период ожидания

Еще одна приятная особенность UpdatePanel заключается в возможности отображать экранную заставку во время ожидания получения новых данных от сервера, особенно если генерирование данных на сервере занимает продолжительное время (например, при выполнении сложных операций в базе данных). Простой баннер с надписью «loading» (загрузка) сообщит пользователю, что идет обработка запроса и в настоящий момент невозможно выполнить повторную отправку данных формы.

В следующем примере имитируется медленная работа сценария на стороне сервера, что вынуждает ASP.NET AJAX вывести экранную заставку до того момента, пока сценарий не завершит работу.

Для начала создадим медленный серверный сценарий. Этот сценарий просто ничего не делает в течение пяти секунд.

```
void WaitFiveSeconds(object o, EventArgs e)
{
  System.Threading.Thread.Sleep(5000);
}
```

Запуск сценария производится щелчком на кнопке, расположенной в элементе управления UpdatePanel. После щелчка на кнопке сценарий будет работать в течение пяти секунд.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    <asp:Button ID="Button1" runat="server"
         Text="Do something" OnClick="WaitFiveSeconds" />
  </ContentTemplate>
</asp:UpdatePanel>
```

В завершение реализуем экранную заставку. Для этого в ASP.NET AJAX имеется элемент управления UpdateProgress. Внутри этого компонента в элемент <ProgressTemplate> следует поместить разметку HTML (или ASP.NET). После этого всякий раз, когда будет происходить обновление UpdatePanel, на странице будет отображаться содержимое шаблона <ProgressTemplate> элемента UpdateProgress. По окончании процесса обновления UpdatePanel содержимое <ProgressTemplate> опять станет невидимым. На одних сайтах в качестве экранной заставки используется изображение песочных часов, на других обычный текст, например: «loading...» («загрузка...»).

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server">
  <ProgressTemplate>
    <div style="position: absolute; left: 200px; top: 150px; border: solid 2px
black;">
      Loading ... Please stand by ...
    </div>
  </ProgressTemplate>
</asp:UpdateProgress>
```

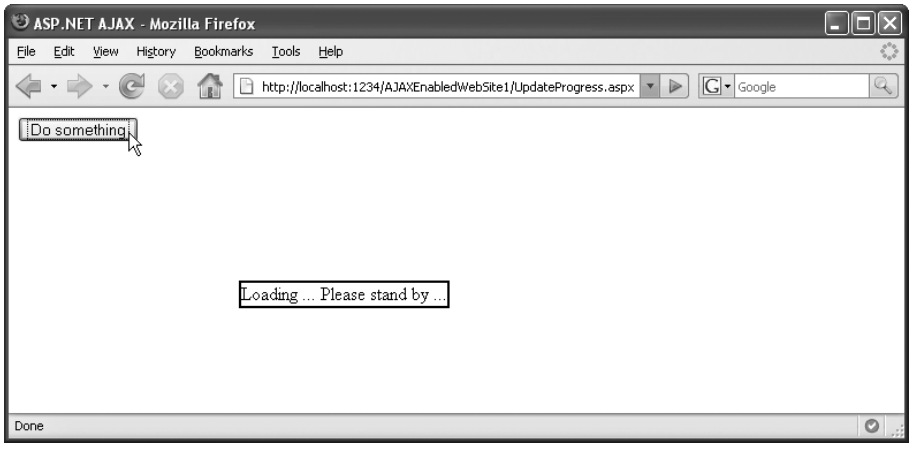


Рис. 6.5. *Экранная заставка, которая появляется на время обновления содержимого элемента управления UpdatePanel*

Для организации совместной работы элементы `UpdatePanel` и `UpdateProgress` необходимо связать между собой, для этой цели в последнем из них имеется свойство `AssociatedUpdatePanelID`. В примере 6.4 содержится полный программный код этого примера. На рис. 6.5 демонстрируется страница сразу после щелчка на кнопке, что вызывает пятисекундную задержку обновления `UpdatePanel`.

Пример 6.4. Экранная заставка в элементе UpdatePanel

```
UpdateProgress.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  void WaitFiveSeconds(object o, EventArgs e)
  {
    System.Threading.Thread.Sleep(5000);
    Label1.Text = DateTime.Now.ToLongTimeString();
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>
```

```
        <asp:Button ID="Button1" runat="server"
             Text="Do something" OnClick="WaitFiveSeconds" /><br />
        <asp:Label ID="Label1" runat="server" />
      </ContentTemplate>
    </asp:UpdatePanel>
    <asp:UpdateProgress ID="UpdateProgress1" runat="server"
         AssociatedUpdatePanelID="UpdatePanel1">
      <ProgressTemplate>
        <div style="position: absolute; left: 200px; top: 150px;
                              border: solid 2px black; padding:4px;">
          Loading, please stand by ...
        </div>
      </ProgressTemplate>
    </asp:UpdateProgress>
  </form>
</body>
</html>
```

Когда асинхронная операция передачи данных выполняется достаточно быстро, элемент UpdateProgress появляется на экране на очень короткое время, что вызывает раздражающий эффект мерцания. В таких случаях желательно использовать свойство DisplayAfter компонента UpdateProgress. В этом свойстве устанавливается время в миллисекундах, которое должно пройти перед выводом содержимого элемента <ContentTemplate> компонента UpdateProgress. Чтобы избежать неприятного эффекта мерцания, устанавливайте с помощью этого свойства разумное время задержки, например одну секунду. Значение по умолчанию – 500 миллисекунд.

Управление асинхронными запросами

При использовании элемента управления UpdatePanel обработку асинхронных запросов ASP.NET AJAX выполняет класс PageRequestManager. Наиболее важная его особенность – это способность прерывать затянувшееся ожидание ответа от сервера. Помните элемент UpdateProgress из предыдущего раздела? Экранная заставка во время ожидания – это очень неплохо, но если запрос исполняется слишком долго, то через некоторое время необходимо что-то предпринимать.

Чтобы прервать затянувшийся запрос, для начала необходимо получить доступ к соответствующему экземпляру PageRequestManager. На каждой странице может быть только один такой экземпляр. Для обращения к нему используется следующий синтаксис:

```
Sys.WebForms.PageRequestManager.getInstance();
```

К экземпляру класса PageRequestManager можно присоединять свои обработчики событий. Поддерживаются следующие события:

beginRequest

Перед отправкой запроса серверу.

endRequest

После того как запрос будет завершен.

initializeRequest

Запрос был инициализирован.

pageLoaded

Содержимое страницы было обновлено в результате запроса.

pageLoading

Данные от сервера были получены, но страница еще не обновлялась.

Эти события возникают в следующем порядке: initializeRequest, beginRequest, pageLoading, pageLoaded и endRequest.

Следующий пример предоставляет пользователю возможность щелчком на кнопке прервать затянувшийся запрос. Кнопка инициирует отправку на сервер своих собственных данных. А для того, чтобы инициализировать новый запрос, необходимо прервать исполнение текущего. Чтобы обеспечить такое поведение, в обработчике события initializeRequest необходимо выполнить некоторые действия:

```
var manager = Sys.WebForms.PageRequestManager.getInstance();
manager.add_initializeRequest(abortPendingPostback);
```

Функция JavaScript abortPendingPostback() автоматически получает два аргумента (подобно методам обработки событий в .NET): объект, который породил событие, и дополнительные аргументы, если они имеются. В случае с классом PageRequestManager дополнительные аргументы имеют чрезвычайно важное значение. Для доступа к элементу, инициировавшему отправку запроса, можно использовать метод get_postBackElement().

Кроме того, могут пригодиться следующие два метода PageRequestManager:

abortPostback()

Прерывает исполнение запроса

get_isInAsyncPostBack()

Проверяет, занят ли в настоящее время PageRequestManager обработкой асинхронного запроса

Эти методы делают возможным прерывание затянувшихся запросов. Воспользуемся примером 6.4 в качестве отправной точки и добавим новую кнопку для отмены запроса:

```
<asp:Button ID="Button2" runat="server"
            Text="Abort postback" Style="display:none;" />
```

Когда произойдет щелчок на кнопке, которая запускает программный код на сервере, должна сделаться видимой кнопка отмены:

```
<asp:Button ID="Button1" runat="server"
            Text="Do something" OnClick="WaitFiveSeconds"
            OnClientClick="$get('Button2').style.display='';" />
```

Теперь пришло время заняться логикой сценария. Сначала функция abortPendingPostback() определяет, не занят ли в настоящее время PageRequestManager обработкой запроса. После этого она отыскивает атрибут id отправителя, чтобы узнать, какая из кнопок была нажата. (Метод abortPendingPostback() вызывается по щелчку на любой из двух кнопок, поскольку обе кнопки инициируют отправку запроса.)

Если щелчок был произведен на кнопке «отмены», производится прерывание запроса:

```
function abortPendingPostback(sender, eventArgs) {
  var manager = Sys.WebForms.PageRequestManager.getInstance();
  if (manager.get_isInAsyncPostBack() &&
    eventArgs.get_postBackElement().getAttribute("id") == "Button2") {
    manager.abortPostBack();
  }
}
```

Единственное, что остается сделать после отправки запроса, – скрыть вторую кнопку:

```
Button2.Style["display"] = "none";
```

В примере 6.5 демонстрируется полный программный код страницы, в которой предусмотрена возможность отмены асинхронного HTTP-запроса. На рис. 6.6 показано, как выглядит эта страница в броузере.

Пример 6.5. Экранная заставка для UpdatePanel с возможностью отмены

```
UpdateProgressAbort.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  void WaitFiveSeconds(object o, EventArgs e)
  {
    System.Threading.Thread.Sleep(5000);
    Label1.Text = DateTime.Now.ToLongTimeString();
    Button2.Style["display"] = "none";
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
  <script type="text/javascript">
    function pageLoad() {
      var manager = Sys.WebForms.PageRequestManager.getInstance();
```

```
        manager.add_initializeRequest(abortPendingPostback);
      }

      function abortPendingPostback(sender, eventArgs) {
        var manager = Sys.WebForms.PageRequestManager.getInstance();
        if (manager.get_isInAsyncPostBack() &&
            eventArgs.get_postBackElement().getAttribute("id") == "Button2") {
          manager.abortPostBack();
        }
      }
    </script>
  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server" />
      <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
          <asp:Button ID="Button1" runat="server"
                      Text="Do something" OnClick="WaitFiveSeconds"
                    OnClientClick="$get('Button2').style.display='';" /><br />
          <asp:Button ID="Button2" runat="server"
                      Text="Abort postback" Style="display:none;" />
          <asp:Label ID="Label1" runat="server" />
        </ContentTemplate>
      </asp:UpdatePanel>
      <asp:UpdateProgress ID="UpdateProgress1" runat="server"
                          AssociatedUpdatePanelID="UpdatePanel1">
        <ProgressTemplate>
          <div style="position: absolute; left: 200px; top: 150px;
                              border: solid 2px black; padding: 4px;">
              Loading, please stand by ...<br />
          </div>
        </ProgressTemplate>
      </asp:UpdateProgress>
    </form>
  </body>
  </html>
```

Подведение итогов

В этой главе была представлена одна из самых значимых возможностей ASP.NET AJAX: элемент управления `UpdatePanel`. Кроме того, здесь были описаны сопутствующие ему компоненты `Timer` и `UpdateProgress`, из которых последний реализует экранную заставку, появляющуюся на время ожидания, – эффект, обычно используемый в Ajax-приложениях.

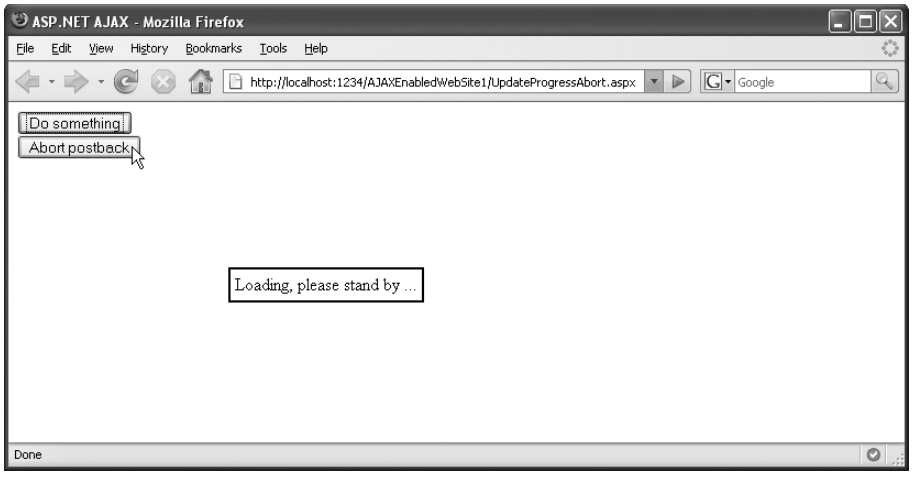


Рис. 6.6. *Теперь есть возможность отменить запрос щелчком на кнопке*

Для дополнительного чтения

Gibbs, Matt and Bertrand LeRoy. ASP.NET AJAX UpdatePanel Control (O'Reilly)

Подробное описание компонента UpdatePanel.

http://ajax.asp.net/docs/overview/UpdatePanelOverview.aspx

Документация компании Microsoft с описанием `UpdatePanel`; здесь, в разделе «Controls that Are Not Compatible with UpdatePanel Controls» (элементы управления, несовместимые с компонентом UpdatePanel), вы найдете перечень элементов управления, которые не могут работать в паре с `UpdatePanel`.

7

Использование службы управления профилями в ASP.NET AJAX

Профили пользователей – это механизмы хранения, которые позволяют хранить на стороне сервера данные как для зарегистрированных, так и для незарегистрированных (анонимных) пользователей. Платформа ASP.NET 2.0 обладает поддержкой профилей, предоставляющей прикладной интерфейс (API) с типовой безопасностью, упрощающий операции чтения и записи данных на стороне сервера.

Разумеется, понятие профиля пользователя не является чем-то новым или особенным. Например, HTTP-cookie обеспечивают возможность управлять данными на стороне клиента. Хотя, чтобы не терять в производительности, в большинстве схем управления параметрами пользователя не производится сохранение всей пользовательской информации на стороне клиента, а используется лишь уникальный идентификатор (ID). Этот ID постоянно передается туда и обратно между сервером и клиентом и выступает в качестве (первичного) ключа доступа к хранимым данным при извлечении необходимой информации. В Microsoft этот механизм был введен для облегчения продвижения Windows NT и ASP (Active Server Pages – активные серверные страницы). А реализация ASP.NET 2.0 вобрала в себя лучшие черты предыдущих реализаций, включая вышеупомянутый API с типовой безопасностью.

В состав платформы ASP.NET AJAX включены функции JavaScript, которые позволяют клиентским сценариям получать доступ к данным из профиля на стороне сервера. В результате JavaScript имеет возможность читать и сохранять данные профиля, что позволяет создавать динамические веб-страницы, учитывающие личные предпочтения пользователей и требующие минимального количество обновлений. За кулисами программный код JavaScript использует поддержку веб-служб ASP.NET AJAX для обращения к серверному компоненту, который за-

тем получает возможность доступа к данным из профиля. Как и в случае со многими другими компонентами ASP.NET AJAX, такое решение нельзя назвать революционным, но оно действительно позволяет существенно сократить время разработки приложений. Помимо службы управления профилями ASP.NET AJAX предоставляет доступ и к другим прикладным службам на стороне сервера, таким как ASP.NET Membership (глава 8) и веб-службы ASP.NET (глава 5).

Подготовка веб-сайта

Чтобы пользоваться преимуществами, которые дает поддержка профилей в ASP.NET AJAX, ее сначала необходимо активировать. Для этого в файле *Web.config* необходимо определить дополнительные элементы. В узел `<system.web>` (по умолчанию уже имеется в файлах *Web.config* сайтов, созданных из шаблона ASP.NET AJAX в среде Visual Studio) необходимо добавить элемент `<profile>`, где будет определяться набор свойств, необходимых для работы вашего приложения. К слову – это особенность не ASP.NET AJAX, а ASP.NET 2.0.

По умолчанию поддержка профилей в ASP.NET 2.0 действует только для зарегистрированных пользователей. Однако, воспользовавшись элементом `<anonymousIdentification>`, можно генерировать уникальные ключи и для незарегистрированных пользователей с целью идентифицировать их данные профиля на стороне сервера:

```
<configuration>
  [...]
  <system.web>
    <anonymousIdentification enabled="true" />
```

Объявить свойства для фактических данных профиля можно двумя способами: в виде отдельных свойств или в виде групп свойств. В следующем фрагменте показаны оба варианта: `userName` – это отдельное свойство, и `UserData` – групповое свойство. Свойство `UserData` включает в себя параметры `myUserName` и `myPassword`.

```
  <profile>
    <properties>
      <add name="userName" allowAnonymous="true" />
      <group name="UserData">
        <add name="myUserName" allowAnonymous="true" />
        <add name="myPassword" allowAnonymous="true" />
      </group>
    </properties>
  </profile>
  [...]
</system.web>
[...]
```

Атрибут `allowAnonymous` свойства `userName` позволяет пользоваться этим свойством незарегистрированным пользователям. Однако для групп свойств в целом (элемент `<group>`) такая возможность недоступна – этот атрибут необходимо устанавливать для каждого свойства в отдельности.

В заключение необходимо активизировать компонент, который даст возможность обращаться к информации в профиле из сценариев JavaScript. Для этого нужно добавить элемент `<system.web.extensions>` (специфичный для ASP.NET AJAX!) в конец файла *Web.config*, непосредственно перед закрывающим тегом `</configuration>`, и настроить его, как показано ниже:

```
    <system.web.extensions>
      <scripting>
        <webServices>
          <profileService
     enabled="true"
     readAccessProperties="userName,UserData.myUserName,UserData.myPassword"
     writeAccessProperties="userName,UserData.myUserName,UserData.myPassword" />
        </webServices>
      </scripting>
    </system.web.extensions>
  </configuration>
```

В элементе `<profileService>` необходимо определить перечень свойств, доступных приложению для чтения и для записи. Свойства, которые объявляются доступными клиентским сценариям, могут составлять лишь часть всего множества свойств профиля приложения. Кроме того, ASP.NET различает доступ к данным для чтения и для записи. Обратите внимание на точечную нотацию (*<группа>.<свойство>*) описания сгруппированной информации в профиле.

Теперь можно приступать к выполнению операций чтения и записи данных в профиле с помощью JavaScript.

Доступ к данным в профиле

Функциональность, имеющая отношение к профилям, в ASP.NET AJAX определена в виде класса `ProfileService`. На первом этапе любое приложение, использующее поддержку доступа к профилям, должно загрузить информацию из профиля с помощью метода `Sys.Services.ProfileService.load()`, который ожидает получить четыре параметра:

`propertyNames`

Имена загружаемых свойств. Если в этом аргументе передать значение null или пустую строку, с сервера будут загружены все доступные свойства.

`loadCompletedCallback`

Метод, который необходимо вызвать в случае успешного окончания загрузки профиля.

`failedCallback`

Метод, который необходимо вызвать в случае неудачного окончания загрузки профиля (напоминает функцию обратного вызова, предназначенную для обработки ошибок обращений к веб-службам).

`userContext`

Необязательная информация, которая будет передана функциям обратного вызова.

После того как данные профиля будут загружены, доступ к этой информации выполняется достаточно просто. Читать *и* записывать значения свойств можно с использованием синтаксиса `Sys.Services.ProfileService.properties.<имя свойства>` – при условии, что настройки в файле *Web.config* позволяют это. В нашем примере для доступа к свойству `userName` в профиле используется конструкция:

```
Sys.Services.ProfileService.properties.userName
```

Однако установка значения свойства в профиле на самом деле не приводит к сохранению этой информации на сервере – она доступна лишь сценарию JavaScript. Чтобы сохранить эту информацию, необходимо вызвать метод `save()`, который ожидает получить четыре параметра, как и метод `load()`:

`propertyNames`

Имена сохраняемых свойств. Если в этом аргументе передать значение null или пустую строку, на сервер будут выгружены все свойства, доступные клиентскому сценарию.

`loadCompletedCallback`

Метод, который необходимо вызвать в случае успешного сохранения профиля.

`failedCallback`

Метод, который необходимо вызвать в случае неудачного сохранения профиля.

`userContext`

Необязательная информация, которая будет передана функциям обратного вызова.

Ниже демонстрируется пример использования этого прикладного интерфейса.

Форма регистрации, где вводится имя пользователя (и, возможно, пароль), остается неизменной. Как обычно, сценарий начинается с элемента `ScriptManager`:

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
```

Затем необходимо создать форму регистрации, содержащую три обычных для этого случая элемента – поле ввода имени пользователя, поле ввода пароля и кнопку. Кроме того, на форме будет находиться метка HTML, в которую сценарий JavaScript будет выводить информацию о состоянии обращения к службе управления профилями:

```
User name: <input type="text" id="txtUsername" runat="server" /><br />
Password: <input type="password" id="txtPassword" runat="server" /><br />
<input type="button" id="Button1" runat="server" value="Login"
  onclick="alert('not implemented!');" /><br />
<span id="statusText" runat="server"> </span>
```

Теперь можно приступить к реализации клиентского сценария. Когда страница и библиотеки ASP.NET AJAX будут загружены, сценарий извлекает все данные из профиля с помощью метода `load()`:

```
function pageLoad() {
  Sys.Services.ProfileService.load(
    "",
    profileLoaded,
    profileError,
    "load");
}
```

Функция `profileError()` просто выводит в метку сообщение, содержащее полезную (надеюсь) информацию об ошибке:

```
function profileError(result, context) {
  $get("statusText").firstChild.nodeValue =
    "Could not "+ context + " profile (" +
    result.get_message() +
    "). Check the configuration in web.config!";
}
```

Обратите внимание на то, как используется информация из необязательного аргумента `userContext` функции `load()` в функции обратного вызова, чтобы сообщение об ошибке начиналось со слов «*Could not load profile*» (невозможно загрузить настройки). Как вы уже наверняка догадались, это позволяет повторно использовать метод `profileError()` для обработки ошибки сохранения параметров пользователя, но с иным значением аргумента `context`.

Если информация из профиля будет благополучно загружена, становится доступным имя пользователя; оно записывается в текстовое поле:

```
function profileLoaded() {
  $get("statusText").firstChild.nodeValue = "Profile data loaded.";
  if (Sys.Services.ProfileService.properties.userName != null) {
    $get("txtUsername").value =
      Sys.Services.ProfileService.properties.userName;
  }
}
```

Начиная с этого момента пути назад уже не будет. Когда пользователь введет в форму другое имя, эти данные необходимо вернуть на сервер и сохранить в профиле. Этот процесс запускает функция `saveProfile()`:

```
function saveProfile() {
  Sys.Services.ProfileService.properties.userName = $get("txtUsername").value;
  Sys.Services.ProfileService.save(
    null,
    profileSaved,
    profileError,
    "save");
}
```

Здесь функция `profileError()` используется повторно, а функция `profileSaved()` – новая. Ее цель – вывести новую информацию о состоянии:

```
function profileSaved() {
  $get("statusText").firstChild.nodeValue = "Profile data saved.";
}
```

Все, что осталось сделать в приложении, – это вызвать функцию `saveProfile()`. Данная функция должна вызываться при изменении содержимого поля ввода с именем пользователя, то есть как обработчик события изменения текстового поля ввода. (Можно предусмотреть и другую возможность, например, сохранять имя пользователя только по нажатию кнопки регистрации.) Это поведение легко реализуется средствами JavaScript. Один из возможных способов заключается в использовании метода addHandler(), предоставляемого платформой ASP.NET AJAX:

```
$addHandler(
  $get("txtUsername"),
  "change",
  saveProfile);
```

В примере 7.1 представлен полный программный код страницы. Не забудьте добавить в *Web.config* элементы, описанные ранее, чтобы активировать свойства профиля и сделать их доступными для клиентских приложений.

Пример 7.1. Чтение и запись данных профиля

```
Profile.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
  <script type="text/javascript">
    function pageLoad() {
      $addHandler(
        $get("txtUsername"),
```

```
        "change",
        saveProfile);
      Sys.Services.ProfileService.load(
        null,
        profileLoaded,
        profileError,
        "load");
    }

    function profileLoaded() {
      $get("statusText").firstChild.nodeValue = "Profile data loaded.";
      if (Sys.Services.ProfileService.properties.userName != null) {
        $get("txtUsername").value =
          Sys.Services.ProfileService.properties.userName;
      }
    }

    function profileError(result, context) {
      $get("statusText").firstChild.nodeValue =
        "Could not " + context + " profile (" +
        result.get_message() +
        "). Check the configuration in web.config!";
    }

    function saveProfile() {
      Sys.Services.ProfileService.properties.userName =
        $get("txtUsername").value;
      Sys.Services.ProfileService.save(
        null,
        profileSaved,
        profileError,
        "save");
    }

    function profileSaved() {
      $get("statusText").firstChild.nodeValue = "Profile data saved.";
    }

  </script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      User name: <input type="text" id="txtUsername" runat="server" /><br />
      Password: <input type="password" id="txtPassword" runat="server" />
      <br />
      <input type="button" id="Button1" runat="server" value="Login"
             onclick="alert('not implemented!');" /><br />
      <span id="statusText" runat="server"> </span>
    </div>
  </form>
</body>
</html>
```

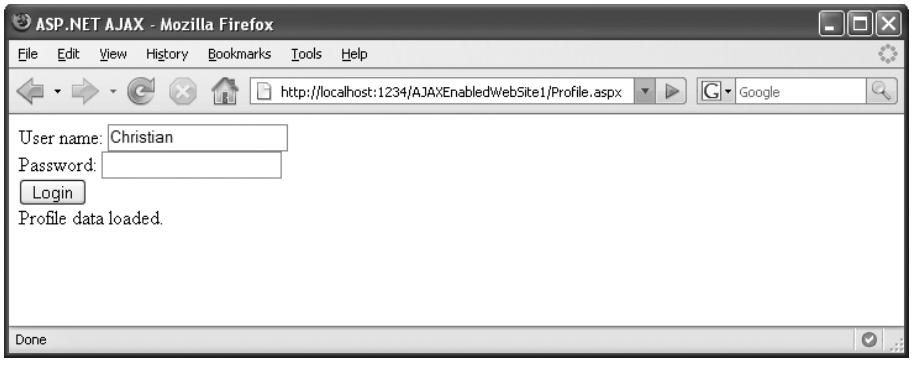


Рис. 7.1. *Теперь в поле User name сразу же загружается ранее введенное имя пользователя*

При запуске этого примера поля формы регистрации сначала пустые. (В самый первый раз можно заметить, что запуск примера длится несколько дольше, чем обычно – это связано с настройкой базы данных службы управления профилями в ASP.NET.) После ввода имени пользователя и нажатия клавиши табуляции введенное имя будет сохранено. С этого момента всякий раз, когда будет загружаться страница (при принудительном обновлении страницы из броузера или при повторном запуске броузера и примера), имя пользователя, введенное ранее, будет загружено в текстовое поле с меткой «User name» (имя пользователя), как показано на рис. 7.1.

Опробовав этот пример, загляните в каталог `App_Data` на веб-сайте. Здесь вы найдете созданную базу данных профилей с именем *ASPNETDB.MDF*. Если открыть ее, в базе данных *aspnet_Profile* можно будет найти запись с именем пользователя (рис. 7.2.).

Доступ к сгруппированным данным в профиле

Если в профиле используются сгруппированные данные, порядок доступа к такой информации несколько отличается от доступа к отдельным свойствам. Чтобы получить доступ к свойству в группе, необходимо указать имя группы и самого свойства, разделенные точкой:

```
Sys.Services.ProfileService.properties.<имя группы>.<имя свойства>
```

Еще раз напомню, что невозможно в файле *Web.config* объявить доступной для чтения или записи всю группу целиком, но это можно сделать для всех элементов в группе по отдельности:

```
<profileService
  enabled="true"
  readAccessProperties="UserData.myUserName,UserData.myPassword"
  writeAccessProperties="UserData.myUserName,UserData.myPassword" />
```

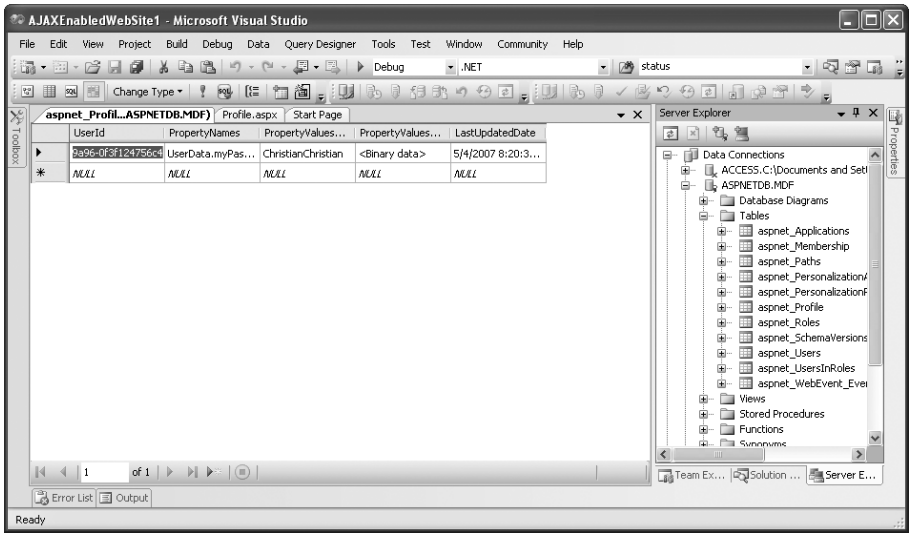


Рис. 7.2. *В MDF-файле содержатся данные профиля*

Теперь можно дополнить пример 7.1 и включить в него возможность сохранения в профиле пароля и имени пользователя. Для этого нужно совсем немного – добавить две функции сохранения, по одной для каждого свойства.

```
function saveProfile1() {
  Sys.Services.ProfileService.properties.UserData.myUserName =
$get("txtUsername").value;
  Sys.Services.ProfileService.save(
    null,
    profileSaved,
    profileError,
    {"operation": "save", "property": "username"});
}

function saveProfile2() {
  Sys.Services.ProfileService.properties.UserData.myPassword =
    $get("txtPassword").value;
  Sys.Services.ProfileService.save(
    null,
    profileSaved,
    profileError,
    {"operation": "save", "property": "password"});
}
```

Как видите, в данном случае снова используется тот же контекст, но на этот раз вместо строки используется объект. Свойство `operation` этого объекта будет содержать либо строку `"save"`, либо строку `"load"` (разумеется, первое значение будет использоваться только в функции со-

Профили в ASP.NET AJAX: за кулисами

Если в процессе работы примера наблюдать за HTTP-трафиком, можно проследить, как ASP.NET AJAX производит обращение к веб-службе для получения информации из профиля и как происходит ее сохранение на сервере. На рис. 7.3 приводится содержимое типичного HTTP-трафика, который был получен с помощью популярного расширения Fireburg для броузера Firefox (*http://www.getfirebug.com/*).

хранения), а свойство `property` содержит информацию о том, какие данные были записаны в профиль. В результате удалось убить сразу двух зайцев: в сценарии присутствует всего одна функция обработки ошибок и всего одна функция, обрабатывающая успешное завершение записи данных в профиль. Приведем сначала последний обработчик:

```
function profileSaved(success, context) {
  $get("statusText").firstChild.nodeValue =
    "Profile data (" + context.property + ") saved.";
}
```

Конкретная контекстная информация также должна приниматься во внимание при вызове метода `load()`:

```
Sys.Services.ProfileService.load(
  "",
```

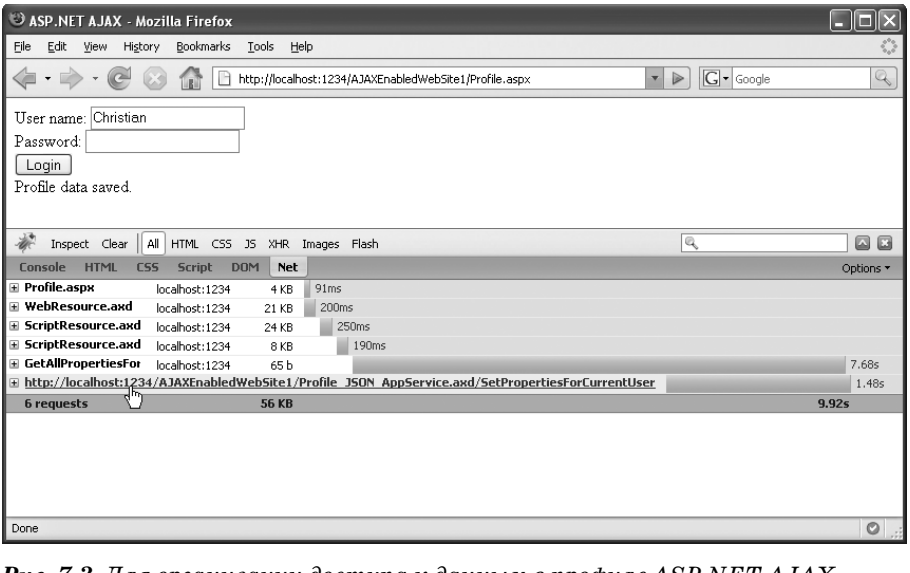


Рис. 7.3. *Для организации доступа к данным в профиле ASP.NET AJAX использует свои собственные веб-службы*

```
  profileLoaded,
  profileError,
  {"operation": "load"});
```

В заключение необходимо изменить функцию `profileError()`:

```
function profileError(result, context) {
  $get("statusText").firstChild.nodeValue =
    "Could not " + context.operation + " profile (" +
    result.get_message() +
    "). Check the configuration in web.config!";
}
```

Полный программный код этого приложения приводится в примере 7.2, а на рис. 7.4 показан возможный результат.

Пример 7.2. Чтение и запись сгруппированных данных профиля

```
ProfileGroup.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
  <script type="text/javascript">
    function pageLoad() {
      $addHandler(
        $get("txtUsername"),
        "change",
        saveProfile1);
      $addHandler(
        $get("txtPassword"),
        "change",
        saveProfile2);
      Sys.Services.ProfileService.load(
        null,
        profileLoaded,
        profileError,
        {"operation": "load"});
    }

    function profileLoaded() {
      if (Sys.Services.ProfileService.properties.UserData != null) {
        $get("statusText").firstChild.nodeValue = "Profile data loaded.";
        $get("txtUsername").value =
          Sys.Services.ProfileService.properties.UserData.myUserName;
        $get("txtPassword").value =
          Sys.Services.ProfileService.properties.UserData.myPassword;
      } else {
```

```
        $get("statusText").firstChild.nodeValue = "No data available.";
      }
  }

  function profileError(result, context) {
    $get("statusText").firstChild.nodeValue =
      "Could not " + context.operation + " profile (" +
      result.get_message() +
      "). Check the configuration in web.config!";
  }

  function saveProfile1() {
    Sys.Services.ProfileService.properties.UserData.myUserName =
      $get("txtUsername").value;
    Sys.Services.ProfileService.save(
      null,
      profileSaved,
      profileError,
      {"operation": "save", "property": "username"});
  }

  function saveProfile2() {
    Sys.Services.ProfileService.properties.UserData.myPassword =
      $get("txtPassword").value;
    Sys.Services.ProfileService.save(
      null,
      profileSaved,
      profileError,
      {"operation": "save", "property": "password"});
  }

  function profileSaved(success, context) {
    $get("statusText").firstChild.nodeValue = "Profile data (" +
      context.property
      + ") saved.";
  }

  </script>
</head>
```

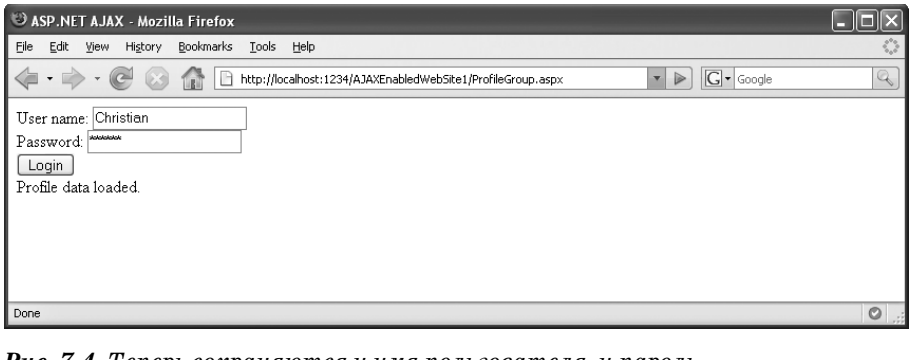


Рис. 7.4. *Теперь сохраняются и имя пользователя, и пароль*

```
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      User name: <input type="text" id="txtUsername" runat="server" /><br />
      Password: <input type="password" id="txtPassword" runat="server" />
      <br />
      <input type="button" id="Button1" runat="server" value="Login"
             onclick="alert('not implemented!');" /><br />
      <span id="statusText" runat="server"> </span>
    </div>
  </form>
</body>
</html>
```

Примите к сведению: пароли никогда не должны храниться в открытом виде. Механизм аутентификации ASP.NET 2.0 (который будет рассматриваться в следующей главе), например, по умолчанию хранит пароли в зашифрованном виде. И хотя на стороне клиента пароли не хранятся в открытом виде, на стороне сервера они могут оказаться незащищенными. Использование протокола HTTPS позволит обезопасить хотя бы передачу паролей.

Подведение итогов

В этой главе была использована служба управления профилями – ASP.NET 2.0 Profile Service, причем без применения серверных сценариев. Вместо этого был использован прикладной интерфейс JavaScript, предоставляемый ASP.NET AJAX для доступа к данным профиля с применением исключительно клиентского сценария.

Для дополнительного чтения

http://ajax.asp.net/docs/tutorials/UsingProfileInformationTutorial.aspx

Описание JavaScript Profile API в документации ASP.NET AJAX компании Microsoft

http://www.ondotnet.com/pub/a/dotnet/2004/10/25/libertyonwhidbey.html

Электронная статья о профилях в ASP.NET 2.0

8

Использование службы аутентификации в ASP.NET AJAX

Одна из целей при разработке ASP.NET 2.0 заключалась в том, чтобы уменьшить время, требуемое на реализацию типовых задач, выполняемых веб-сайтами. Среди этих задач – задача управления пользователями и правами доступа, включая регистрацию пользователей, завершение сеансов, создание ролей и тому подобное. Действительно, создать защищенный веб-сайт на основе ASP.NET 2.0 очень просто: добавить пользователей и затем использовать систему ASP.NET управления такими операциями, как вход в систему, – и дело сделано.

Некоторые из этих функций управления доступом могут также быть реализованы на JavaScript средствами службы аутентификации ASP.NET AJAX. Набор этих поддерживаемых функциональных возможностей невелик, но они очень удобны. Платформа ASP.NET AJAX поддерживает механизмы аутентификации ASP.NET 2.0. В результате сценарии JavaScript могут выполнять проверку прав доступа пользователей. Однако защита самого содержимого *должна* выполняться на самом сервере. Никогда не забывайте, что поддержку JavaScript можно дезактивировать, что делает наличие такой поддержки недостаточным условием для обеспечения надежной защиты частной информации.

Подготовка приложения

Чтобы иметь возможность использовать службу аутентификации ASP.NET AJAX, необходимо добавить пользователей в приложение. Сделать это можно с помощью такого инструмента, как ASP.NET Web Application Administration Tool, который поставляется в составе Visual Studio и Visual Web Developer. Сначала запустите команду ASP.NET Configuration (настройка ASP.NET) в меню Web Site, как показано на рис. 8.1.

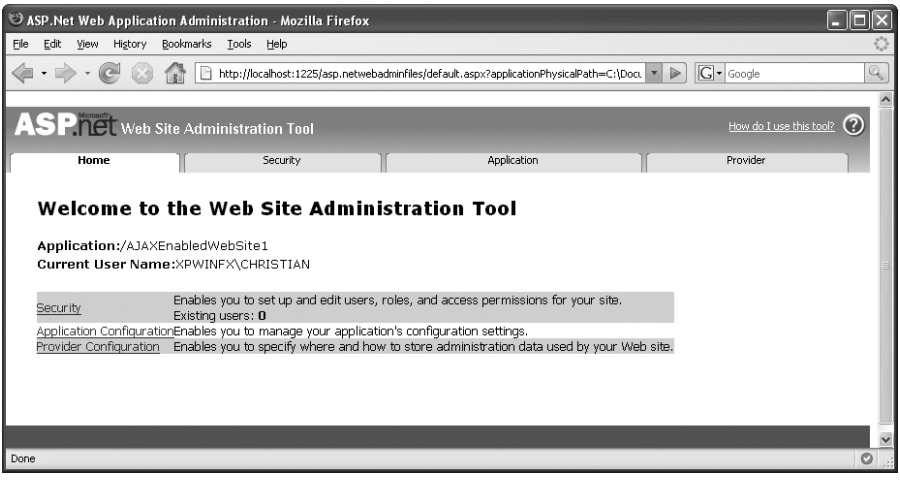


Рис. 8.1. *Инструмент ASP.NET Configuration Tool (хотя в Internet Explorer он выглядит лучше)*

Щелкните на вкладке Security (безопасность), а на следующей странице щелкните на ссылке «Select authentication type» (выбор типа аутентификации). Далее измените текущий тип аутентификации, выбрав радиокнопку «From the internet» (из Интернета) (рис. 8.2), что означает переход на использование механизма аутентификации ASP.NET вместо механизма аутентификации Windows. После этого можно щелкнуть на кнопке Done (завершить).

Затем щелкните на ссылке «Create user» (создать пользователя) и введите информацию хотя бы об одном пользователе (рис. 8.3).

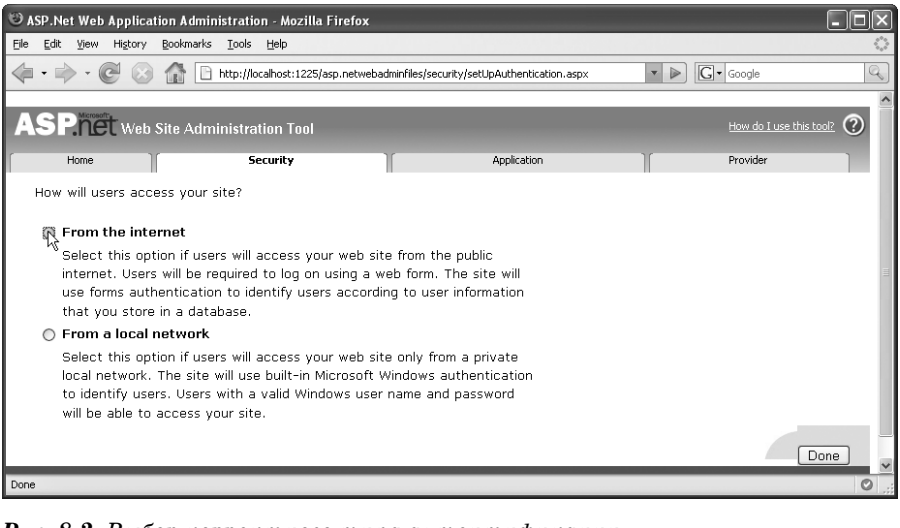


Рис. 8.2. *Выбор корректного типа аутентификации*

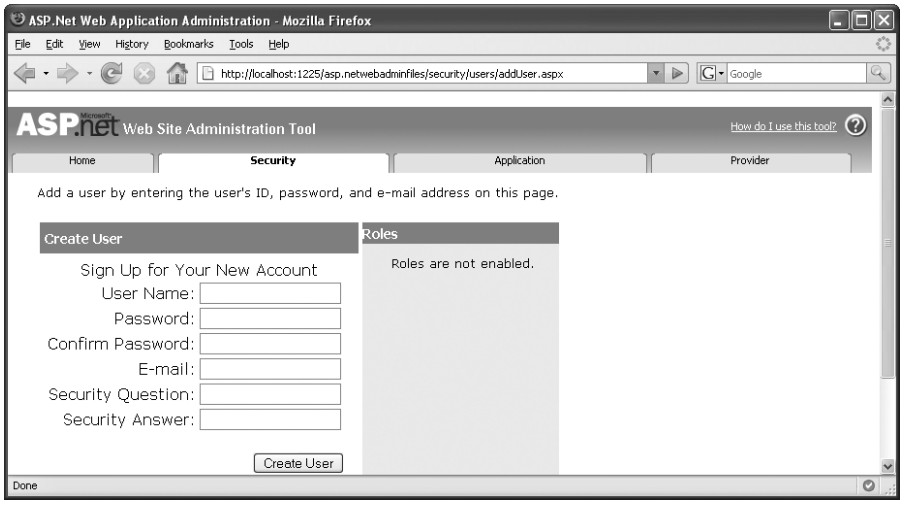


Рис. 8.3. *Создайте нового пользователя (или двух, или трех)*

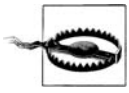
Если перед запуском ASP.NET Configuration Tool файл *Web.config* был открыт в интегрированной среде разработки, то Visual Studio (или Visual Web Developer) предложит перезагрузить файл. Дело в том, что ваши действия привели к появлению в файле *Web.config* (в узле `<system.web>`) следующей строки:

```
<authentication mode="Forms" />
```

Теперь в файл *Web.config* необходимо добавить еще один параметр. К сожалению, сделать это придется без помощи инструмента с графическим интерфейсом, вручную. В предыдущей главе в файл *Web.config* был добавлен узел `<system.web.extensions>`. В этот узел следует добавить элемент `<authenticationService>` и установить значение `true` в атрибуте `enabled`:

```
<configuration>
  [...]
  <system.web.extensions>
    <scripting>
      <webServices>
        <profileService
          enabled="true"
readAccessProperties="userName,UserData.myUserName,UserData.myPassword"
writeAccessProperties="userName,UserData.myUserName,UserData.myPassword" />
        <authenticationService enabled="true" />
      </webServices>
    </scripting>
  </system.web.extensions>

</configuration>
```

Тем самым будет гарантирована передача программного кода JavaScript, который подключает клиента к серверному API.



Убедитесь, что вы настроили свое приложение в соответствии с инструкциями, приведенными в начале этой и предыдущей главы. Тем самым гарантируется поддержка прикладного интерфейса профилей, аутентификации и поддержка описанных ранее свойств профиля.

Вход и выход

Поддержка механизма ASP.NET AJAX Forms Authentication осуществляется двумя методами класса `Sys.Services.AuthenticationService`: `login()` и `logout()`. Начнем с операции входа пользователя. При вызове этому методу должно быть передано не менее восьми параметров:

`userName`

Имя пользователя

`password`

Пароль

`isPersistent`

Должен ли cookie системы аутентификации постоянно храниться на стороне клиента (по умолчанию `false`)

`redirectUrl`

Адрес URL, куда должен быть перенаправлен броузер после входа пользователя, или `null`, если перенаправление не требуется (по умолчанию)

`customInfo`

В настоящее время не используется

`loginCompletedCallback`

Функция, которая вызывается после успешного входа

`failedCallback`

Функция, которая вызывается в случае неудачной попытки входа

`userContext`

Данные, которые передаются функциям обратного вызова

Существуют два варианта реализации процедуры аутентификации:

1. Вызвать метод `login()` и разрешить JavaScript выполнить перенаправление на другую страницу, установив значение аргумента `redirectUrl`.
2. Вызвать метод `login()` и с помощью функций обратного вызова `loginCompletedCallback()` и `failedCallback()` обработать результаты аутентификации.

Чаще на практике используется второй вариант. Перенаправление упирается в проблему безопасности данных в JavaScript, который, как уже говорилось, не обеспечивает достаточный уровень безопасности.

Выход пользователя производится с помощью метода `logout()`. Он поддерживает «только» четыре аргумента:

`redirectUrl`

Адрес URL, куда должен быть перенаправлен броузер после выхода пользователя, или `null`, если перенаправление не требуется (по умолчанию)

`logoutCompletedCallback`

Функция, которая вызывается после успешного выхода

`failedCallback`

Функция, которая вызывается в случае неудачной попытки выхода

`userContext`

Данные, которые передаются функциям обратного вызова

Теперь на основе файла *ProfileGroup.aspx* (представленного в главе 7) создадим новый файл *Authentication.aspx* и добавим в него эти две функции. Напомню, что страница *ProfileGroup.aspx* содержит форму регистрации пользователя и позволяет сохранять имя пользователя и пароль в переменных профиля. В предыдущих примерах кнопка на форме не имела никакой функциональности. Поэтому, перед тем как добавить функциональность, сначала удалим фиктивный код JavaScript, связанный с этой кнопкой, чтобы разметка приняла следующий вид:

```
<input type="button" id="Button1" runat="server" value="Login" />
```

В функцию `pageLoad()` вставляем обработчик событий от кнопки:

```
$addHandler(
  $get("Button1"),
  "click",
  doClick);
```

Функция `doClick()`, на которую ссылается вышеприведенный фрагмент, проверяет надпись на кнопке. Если это строка `"Login"`, выполняется попытка входа пользователя, если `"Logout"` — производится попытка выхода.

Теперь необходимо соответствующим образом установить аргументы методов `login()` и `logout()`. Поскольку информация о пользователях постоянно хранится в профиле, нам не требуется сохранять информацию о вошедшем пользователе. Кроме того, мы не будем выполнять перенаправление пользователя после входа или после выхода. Наконец, мы воспользуемся приемом, о котором узнали в главе 7, и определим значения для аргумента `userContext`, чтобы можно было повторно использовать одни и те же функции обратного вызова для операций входа и выхода.

```
function doClick( ) {
  switch ($get("Button1").value) {
    case "Login":
      Sys.Services.AuthenticationService.login(
```

```
          $get("txtUsername").value,
          $get("txtPassword").value,
          false,
          null,
          null,
          loginComplete,
          loginError,
          "login");
        break;
      case "Logout":
        Sys.Services.AuthenticationService.logout(
          null,
          loginComplete,
          loginError,
          "logout");
        break;
    }
  }
```

Функция `loginError()` обрабатывает такие ошибки при входе и выходе, как неверное имя пользователя или пароль, отсутствие имени пользователя или неправильные настройки на сервере. Соответствующее сообщение об ошибке будет отображаться в метке HTML:

```
function loginError(result, context) {
  $get("statusText").firstChild.nodeValue =
    "Could not " + context + " (" +
    result.get_message( ) + ").";
}
```

Наконец, функция `loginComplete()` выводит сообщение о состоянии операции. Кроме того, она изменяет форму регистрации. После успешного входа пользователя оба текстовых поля делаются недоступными для ввода, а надпись `"login"` на кнопке изменяется на `"logout"`. После того как пользователь совершит выход, надпись на кнопке опять изменится на `"login"`, а оба текстовых поля станут доступными для ввода.

```
function loginComplete(result, context) {
  switch (context) {
    case "login":
      if (result == true) {
        $get("txtUsername").disabled = true;
        $get("txtPassword").disabled = true;
        $get("Button1").value = "Logout";
        $get("statusText").firstChild.nodeValue = "Logged in";
      } else {
        $get("statusText").firstChild.nodeValue = "Login failed";
      }

      break;
    case "logout":
      $get("txtUsername").disabled = false;
      $get("txtPassword").disabled = false;
```

```
        $get("Button1").value = "Login";
        $get("statusText").firstChild.nodeValue = "Logged out";
        break;
    }
}
```

Полный программный код этого примера приводится в примере 8.1. Не забудьте внести в файл *Web.config* необходимые изменения, прежде чем опробовать этот пример.

Пример 8.1. Аутентификация пользователя с помощью JavaScript

```
Authentication.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/ xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script type="text/javascript">
    function pageLoad( ) {
      $addHandler(
        $get("Button1"),
        "click",
        doClick);
      $addHandler(
        $get("txtUsername"),
        "change",
        saveProfile1);
      $addHandler(
        $get("txtPassword"),
        "change",
        saveProfile2);
      Sys.Services.ProfileService.load(
        "",
        profileLoaded,
        profileError,
        {"operation": "load"});
    }

    function profileLoaded( ) {
      if (Sys.Services.ProfileService.properties.UserData != null) {
        $get("statusText").firstChild.nodeValue = "Profile data loaded.";
        $get("txtUsername").value =
          Sys.Services.ProfileService.properties.UserData.myUserName;
        $get("txtPassword").value =
          Sys.Services.ProfileService.properties.UserData.myPassword;
      } else {
        $get("statusText").firstChild.nodeValue = "No data available.";
      }
    }
```

```
function profileError(result, context) {
  $get("statusText").firstChild.nodeValue =
    "Could not " + context.operation + " profile (" +
    result.get_message( ) +
    ". Check the configuration in web.config!";
}

function saveProfile1( ) {
  Sys.Services.ProfileService.properties.UserData.myUserName =
    $get("txtUsername").value;
  Sys.Services.ProfileService.save(
    null,
    profileSaved,
    profileError,
    {"operation": "save", "property": "username"});
}

function saveProfile2( ) {
  Sys.Services.ProfileService.properties.UserData.myPassword =
    $get("txtPassword").value;
  Sys.Services.ProfileService.save(
    null,
    profileSaved,
    profileError,
    {"operation": "save", "property": "password"});
}

function profileSaved(success, context) {
  $get("statusText").firstChild.nodeValue = "Profile data (" +
    context.property + ") saved.";
}

function doClick( ) {
  switch ($get("Button1").value) {
    case "Login":
      Sys.Services.AuthenticationService.login(
        $get("txtUsername").value,
        $get("txtPassword").value,
        false,
        null,
        null,
        loginComplete,
        loginError,
        "login");
      break;
    case "Logout":
      Sys.Services.AuthenticationService.logout(
        null,
        loginComplete,
        loginError,
        "logout");
      break;
  }
}
```

```
      function loginComplete(result, context) {
        switch (context) {
          case "login":
            if (result == true) {
              $get("txtUsername").disabled = true;
              $get("txtPassword").disabled = true;
              $get("Button1").value = "Logout";
              $get("statusText").firstChild.nodeValue = "Logged in";
            } else {
              $get("statusText").firstChild.nodeValue = "Login failed";
            }
            break;
          case "logout":
            $get("txtUsername").disabled = false;
            $get("txtPassword").disabled = false;
            $get("Button1").value = "Login";
            $get("statusText").firstChild.nodeValue = "Logged out";
            break;
        }
      }

      function loginError(result, context) {
        $get("statusText").firstChild.nodeValue =
          "Could not " + context + " (" +
        result.get_message( ) + ").";
      }
    </script>
  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server" />
      <div>
        User name: <input type="text" id="txtUsername" runat="server" /><br />
        Password: <input type="password" id="txtPassword" runat="server" />
        <br />
       <input type="button" id="Button1" runat="server" value="Login" /><br />
        <span id="statusText" runat="server"> </span>
      </div>
    </form>
  </body>
  </html>
```

На рис. 8.4 показана форма перед входом пользователя. На рис. 8.5 видны неактивные поля ввода и изменившаяся надпись на кнопке после успешного входа.

Подведение итогов

В этой главе было продемонстрировано использование механизма аутентификации ASP.NET 2.0 без применения какого-либо программного кода на стороне сервера. Все, что было нами использовано, – это Java-

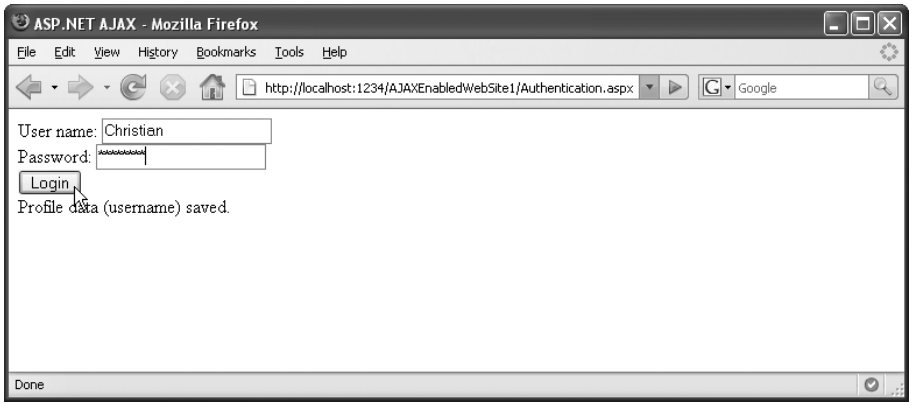


Рис. 8.4. *Перед входом*

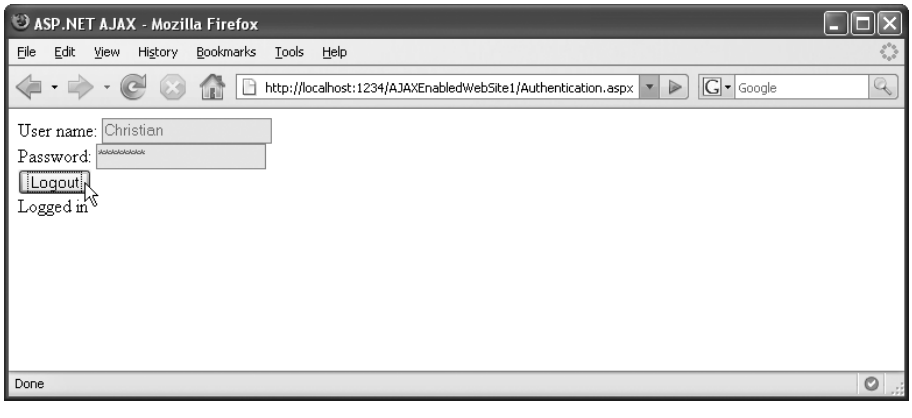


Рис. 8.5. *Перед выходом*

Script и ASP.NET AJAX. Кроме всего прочего, это позволило произвести вход пользователя без полного обновления страницы.

Для дополнительного чтения

http://ajax.asp.net/docs/tutorials/UsingFormsAuthenticationTutorial.aspx

Информация о Forms Authentication JavaScript API в документации компании Microsoft к платформе ASP.NET AJAX

http://msdn2.microsoft.com/en-us/library/ms998310.aspx

Статья о ASP.NET 2.0 Forms Authentication в MSDN

9

Локализация и глобализация приложений

Я хочу рассказать вам одну поучительную историю…

Когда издатели присылают мне авторские экземпляры моих книг, я обычно раздаю их своим друзьям, разыгрываю их в своем блоге или просто складываю у себя в подвале. Одним словом, я не стремлюсь коллекционировать свои книги. Однако из этого правила есть одно исключение: мне ужасно нравится приобретать их иностранные переводы. Надо сказать, что издатели не всегда получают мои изданные за рубежом книги, и в таких случаях я вовсе не претендую на дарственные экземпляры. Но даже если к издателю и попадает какое-нибудь количество таких переводных изданий, то порой проходит целая вечность, прежде чем они достаются мне. Поэтому, когда до меня доходит информация, что очередная моя книга издана в другой стране, я запускаю броузер и начинаю «охоту».

Обычно поиски приводят меня на сайт какого-нибудь книжного Интернет-магазина, где вся информация выводится на незнакомом для меня языке, и потому с большими трудностями мне удается вписать свой адрес и номер кредитной карты в соответствующие поля. Я считаю себя счастливчиком, потому что мои анкетные данные пока еще не были перехвачены злоумышленниками (у меня есть специальная кредитная карта для таких «серых заказов», как я их называю). В результате всего этого время от времени я получаю посылки, которые совершают чуть ли не кругосветное путешествие, прежде чем попадают ко мне.

Но почему, по какой причине я должен испытывать подобные сложности? В век глобализации владельцы веб-сайтов могут существенно расширить круг своих посетителей, если будут общаться на одном с ними языке. На мой взгляд, есть две основные причины, почему основная масса веб-сайтов остаются одноязычными. Первая: перевод сайтов –

удовольствие дорогостоящее, а в зависимости от целевой аудитории, часто выгода от наличия иноязычной версии сайта не покрывает затрат на ее поддержку. Вторая: препятствия технического характера. Если при создании многоязычного сайта вы хотите избежать использования «дизайн-метода», основанного на операциях копирования и вставки, вам потребуется некоторый механизм, который позволит выполнять перевод с минимальными усилиями. Благодаря растущей значимости и быстрому распространению приложений, построенных на базе JavaScript и Ajax, реализация перевода также становится все более значимой и востребованной.

Как всегда, можно выработать свое собственное решение этой задачи, но платформа ASP.NET AJAX уже имеет некоторую поддержку локализации и глобализации. Это позволяет создавать многоязычные версии веб-сайтов, обладающие поддержкой ASP.NET AJAX, и создавать сайты, которые могут предоставлять пользователю ту или иную свою версию, в зависимости от региональных настроек броузера. (Если раньше вы не знали, что броузер может передавать серверу список предпочитаемых языков, вы сможете прочитать об этом ниже, в этой же главе.)

Локализация

Локализация веб-сайта – это процесс адаптации содержимого к региональным настройкам системы, чаще всего – к региональным настройкам пользователя. Часто локализация как таковая обозначается аббревиатурой l10n, которая происходит от обозначения «*l*, затем 10 символов, затем *n*»[1] (так называемый *нумероним*).

На любом веб-сайте имеются самые разные разделы, которые могли бы быть локализованы. И сам текст на сайте, и название валюты, и формат представления даты и времени – все это может быть локализовано. В платформе ASP.NET имеется ряд возможностей по обеспечению локализации (некоторые ссылки вы найдете в разделе «Для дополнительного чтения» в конце главы). Некоторые из этих возможностей используются платформой ASP.NET AJAX для организации поддержки локализации в программном коде JavaScript.

Локализация сценариев

Самый простой способ локализации состоит в том, чтобы написать специализированный сценарий, который будет решать поставленную задачу. Например, этот сценарий мог бы определять региональные настройки и затем загружать ту или иную библиотеку. Половину этой задачи можно решить средствами ASP.NET, а вторую половину – средствами ASP.NET AJAX.

---

[1] Слово «локализация» в английском языке записывается как localization, то есть «l», затем 10 символов («ocalizatio»), затем символ «n». – *Прим. перев.*

Ниже приводится пример короткого сценария JavaScript, который выводит текущую дату в локализованном формате. В файле JavaScript *Dayname.js* определяются две переменные:

`dateformat`

Строка с форматом представления даты, где используются следующие заполнители: `ss` (день недели), `dd` (число месяца), `mm` (месяц) и `yyyy` (год)

`daynames`

Массив с локализованными названиями всех семи дней недели

В примере 9.1 приводится содержимое файла *Dayname.js*.

Пример 9.1. Информация о дате, локализованная для английского языка

```
Dayname.js

var dateformat = "ss, yyyy-mm-dd";

var daynames = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
                "Friday", "Saturday"];
```

Перевести содержимое этого файла на другой язык не представляет труда. В примере 9.2 приводится тот же самый файл, но на этот раз вся информация переведена на немецкий язык. В обоих этих языках используются различные форматы представления дат и названия дней недели.

Пример 9.2. Информация о дате, локализованная для немецкого языка

```
Dayname.de-DE.js

var dateformat = "ss, dd.mm.yyyy";

var daynames = ["Sonntag", "Montag", "Dienstag", "Mittwoch", "Donnerstag",
                "Freitag", "Samstag"];
```

Для локализованных версий сценариев очень важную роль играют имена их файлов. Непосредственно перед расширением файла (*.js*) вставляется информация о языке и, возможно, о регионе, в формате язык-регион, где язык обозначается двухсимвольным кодом языка, а регион обозначается двухсимвольным кодом региона. Код языка, само собой разумеется, определяет язык, на котором написан текст. Код региона определяет форматы представления тех или иных данных (в различных регионах или странах могут говорить на одном и том же языке, но использовать разные соглашения по форматированию – как в случае Соединенных Штатов и Великобритании). Например, американский английский обозначается как en-US, британский английский – en-UK, немецкий (Германия) – de-DE, немецкий (Австрия) – de-AT и так далее. Коды определяются международной организацией по стандартизации ISO, и, в соответствии с соглашениями, код языка записывается строчными символами, а код региона – прописными. В примере 9.2 файл сценария имеет имя *Dayname.de-DE.js*, указывая на немецкий язык и Германию.

Теперь создадим новую страницу *.aspx*, которая будет использовать эти локализованные файлы. В начале страницы расположим элемент `<span>`, который будет использоваться для динамического вывода локализованной даты. Обратите внимание на символ неразрывного пробела (` `) внутри элемента `<span>` – он очень важен для программного кода JavaScript. Обычный пробел не даст желаемого эффекта из-за особенностей поведения Internet Explorer.

```
<span id="date"> </span>
```

Теперь можно добавить программный код JavaScript, который будет использовать переменные `dateformat` и `daynames`, определения которых находятся во внешних файлах JavaScript для вывода даты в локализованном формате:

```
<script type="text/javascript">
  function pageLoad() {
    var d = new Date();
    var datestring = dateformat.replace("ss", daynames[d.getDay()])
                               .replace("dd", d.getDate())
                               .replace("mm", d.getMonth() + 1)
                               .replace("yyyy", d.getFullYear());
    $get("date").firstChild.nodeValue = datestring;
  }
</script>
```

Пока что нет ничего необычного. Но дальше ASP.NET и ASP.NET AJAX начинают свои магические манипуляции. Первое: платформа ASP.NET AJAX должна загрузить внешний файл JavaScript и она (это просто чудо) отыскивает файл, соответствующий текущим языку и региону, основываясь на имени файла. Для загрузки файла *Dayname.js* внутри элемента управления `ScriptManager` используется элемент `<Scripts>`. Основную функцию на этом этапе выполняют два атрибута:

`ResourceUICultures` (*атрибут элемента* `<asp:ScriptReference>`)

Содержит перечень всех поддерживаемых культур (разделенных запятыми), для которых существует перевод.

`EnableScriptLocalization` (*атрибут элемента* `<asp:ScriptManager>`)

Если этот атрибут установить в значение `true`, для текущей страницы будет активирована поддержка механизма локализации ASP.NET AJAX:

```
<asp:ScriptManager ID="ScriptManager1" runat="server"
  EnableScriptLocalization="true">
  <Scripts>
    <asp:ScriptReference Path="Dayname.js"
                                      ResourceUICultures="de-DE,fr-FR" />
  </Scripts>
</asp:ScriptManager>
```

По умолчанию используется файл *Dayname.js*, однако если будет использована одна из культур, перечисленных в атрибуте `ResourceUICul-`

tures (то есть если броузер запросит одну из культур, основываясь на своих настройках), то будет загружен соответствующий локализованный файл.

До сих пор страница еще не установила свою культуру в соответствии с региональными настройками броузера клиента. В этом месте на сцене появляется ASP.NET. Следующая ниже директива Page автоматически установит значение культуры в соответствии с настройками броузера:

```
<%@ Page Language="C#" UICulture="auto" %>
```

Броузер, настроенный на предпочтительное использование французского языка, загрузит файл *Dayname.fr-FR.js*, с другой стороны, броузер, настроенный на предпочтительное использование итальянского языка, загрузит файл *Dayname.js*, так как для итальянского языка нет локализованного сценария. То есть файл *Dayname.js* используется по умолчанию в случае отсутствия локализации для данного языка.

В примере 9.3 приводится полный программный код страницы, которая использует механизм локализации ASP.NET AJAX, который мы только что рассмотрели. Для опробования примера вам необходимо переписать файл *Dayname.js* в корневой каталог вашего веб-сайта.

На рис. 9.1 показано, как выглядит страница в броузере, настроенном на использование английского языка. На рис. 9.2 видно, что в броузере, настроенном на использование немецкого языка, дата отображается в формате, типичном для Германии.

Пример 9.3. Локализация сценария

```
Localization-Inline.aspx

<%@ Page Language="C#" UICulture="auto" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
  <script type="text/javascript">
    function pageLoad() {
      var d = new Date();
      var datestring = dateformat.replace("ss", daynames[d.getDay()])
                                 .replace("dd", d.getDate())
                                 .replace("mm", d.getMonth() + 1)
                                 .replace("yyyy", d.getFullYear());
      $get("date").firstChild.nodeValue = datestring;
    }
  </script>
</head>
<body>
```

Рис. 9.1. *Страница на английском языке*

Рис. 9.2. *Страница на немецком языке*

```
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server"
                         EnableScriptLocalization="true">
        <Scripts>
          <asp:ScriptReference Path="Dayname.js"
                               ResourceUICultures="de-DE,fr-FR" />
        </Scripts>
      </asp:ScriptManager>
      <div>
        <span id="date"> </span>
      </div>
    </form>
  </body>
  </html>
```

Использование сопутствующих ресурсов из ASP.NET AJAX

Другой способ локализации приложений ASP.NET заключается в использовании так называемых *сопутствующих ресурсов* (иногда их называют *сопутствующими сборками*). Под ними подразумеваются скомпилированные внешние файлы ресурсов, которые загружаются, только

если это необходимо для текущей культуры. Платформа ASP.NET AJAX допускает возможность добавления файлов JavaScript (*.js*) в виде ресурсов в сопутствующие сборки, а платформа Ajax обеспечивает механизм использования данных из программного кода JavaScript.

В примере 9.4 используются возможности ASP.NET AJAX по локализации, заключенные в компонент ASP.NET, который может использоваться в веб-странице ASP.NET. На примере этого компонента иллюстрируется решение двух задач. Во-первых, загрузка корректной сопутствующей сборки с помощью ASP.NET AJAX. Во-вторых, доступ к корректным локализованным данным для вывода текста на языке пользователя броузера из сценария, размещенного в сборке.

Для создания компонентов необходимо иметь Visual Studio 2005, так как потребуется создавать скомпилированные сборки. Если вы пользуетесь Visual Web developer Express Edition, то эта среда разработки не поддерживает создание сборок. Однако вы можете установить Visual C# Express Edition (еще один бесплатный инструмент, доступный для загрузки) и пользоваться им.

Для начала создайте новый проект (рис. 9.3). Дайте проекту имя `LocalizedDate`. Вообще это имя не является обязательным, но оно будет использоваться повсюду в этом примере, и поэтому вам предлагается использовать это имя, чтобы следовать за дальнейшим описанием.

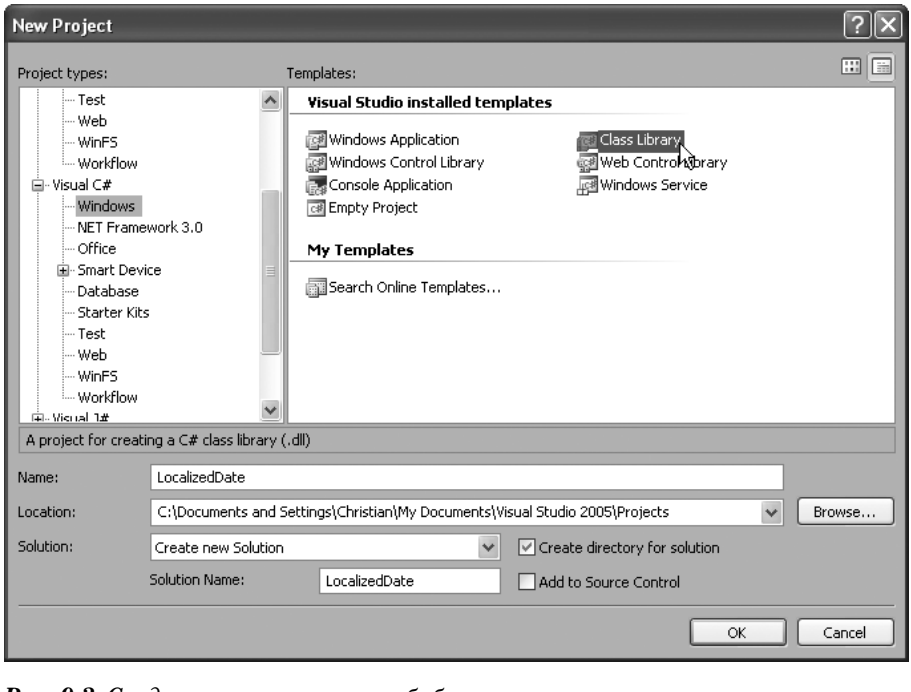


Рис. 9.3. *Создание нового проекта библиотеки классов*

Если вы пользуетесь Visual C# Express Edition, вам придется создавать новый проект с нуля. Пользователи Visual Studio 2005 могут просто добавить новый проект к существующему веб-сайту ASP.NET AJAX. В любом случае это должен быть проект библиотеки классов, как показано на рис. 9.3.

Прежде всего, в проект необходимо добавить ссылки на `System.Web` и `System.Web.Extensions`. Затем добавить некоторые файлы ресурсов. Как показано на рис. 9.4, мы начнем с файла ресурсов английской локализации *DateResources.resx*, который одновременно будет использоваться как исходный. Фактически на рис. 9.4 показан лишь инструмент с графическим интерфейсом для редактирования файла *DateResources.resx*. Внутри файлы ресурсов содержат текст в формате XML. В примере 9.4 показана разметка XML для файла ресурсов с немецкой локализацией. Как можно заметить, применение инструмента с графическим интерфейсом облегчает работу и снижает вероятность появления ошибок. (Чтобы перейти в режим редактирования файла ресурсов в формате XML, перейдите в окно Visual Studio Solution Explorer (обозреватель решений), щелкните правой кнопкой мыши на файле и выберите пункт контекстного меню View Code (просмотр кода).) Создайте немецкую версию файла ресурсов и дайте ему имя *DateResources.de.resx*.

Пример 9.4. Немецкий файл ресурсов

```
DateResources.de.resx

<?xml version="1.0" encoding="utf-8"?>
<root>
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

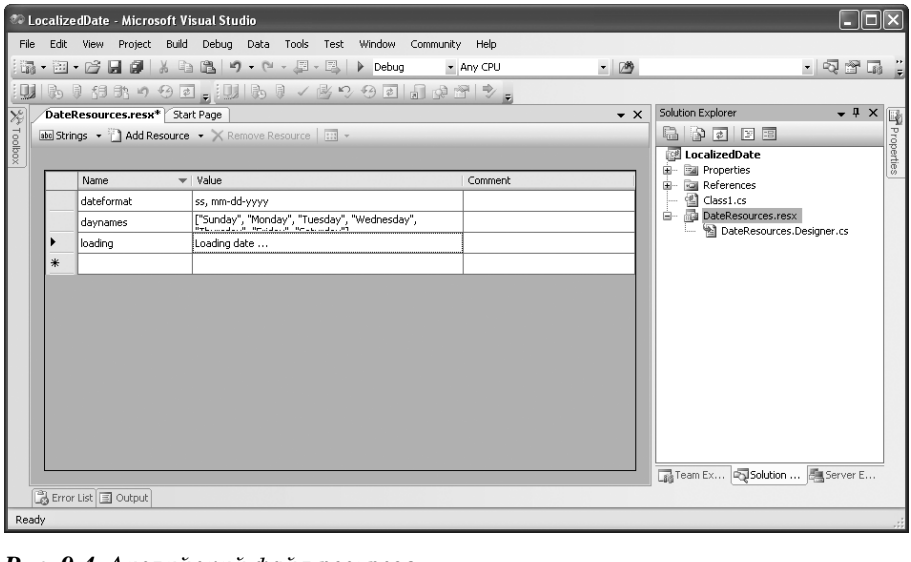


Рис. 9.4. *Английский файл ресурсов*

```
          xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
<xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
<xsd:element name="root" msdata:IsDataSet="true">
  <xsd:complexType>
    <xsd:choice maxOccurs="unbounded">
      <xsd:element name="metadata">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="value" type="xsd:string" minOccurs="0" />
          </xsd:sequence>
          <xsd:attribute name="name" use="required" type="xsd:string" />
          <xsd:attribute name="type" type="xsd:string" />
          <xsd:attribute name="mimetype" type="xsd:string" />
          <xsd:attribute ref="xml:space" />
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="assembly">
        <xsd:complexType>
          <xsd:attribute name="alias" type="xsd:string" />
          <xsd:attribute name="name" type="xsd:string" />
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="data">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="value" type="xsd:string" minOccurs="0"
                msdata:Ordinal="1" />
            <xsd:element name="comment" type="xsd:string" minOccurs="0"
                msdata:Ordinal="2" />
```

Установка культуры в ASP.NET

В примере 9.1 приводится вариант автоматического определения используемой культуры. Однако платформа ASP.NET AJAX обеспечивает механизм динамического изменения культуры. Вы можете дать пользователю возможность изменять культуру страницы, например, с помощью элемента `LinkButton` или раскрывающегося списка. (Нередко используются национальные флаги, но такую практику нельзя признать удачной, поскольку на одном и том же языке могут говорить в нескольких странах, а в некоторых странах официальными признаны несколько языков.)

Для изменения культуры текущей страницы используется свойство `System.Threading.Thread.CurrentThread.CurrentUICulture`. В пространстве имен `System.Globalization` существуют вспомогательные методы создания подходящей культуры для свойства `CurrentUICulture` в текущем потоке. Дополнительная информация об этой особенности ASP.NET приводится в разделе «Для дополнительного чтения» в конце главы.

```
                </xsd:sequence>
                <xsd:attribute name="name" type="xsd:string" use="required"
                     msdata:Ordinal="1" />
                <xsd:attribute name="type" type="xsd:string"
                     msdata:Ordinal="3" />
                <xsd:attribute name="mimetype" type="xsd:string"
                     msdata:Ordinal="4" />
                <xsd:attribute ref="xml:space" />
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="resheader">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="value" type="xsd:string" minOccurs="0"
                      msdata:Ordinal="1" />
                </xsd:sequence>
                <xsd:attribute name="name" type="xsd:string" use="required" />
              </xsd:complexType>
            </xsd:element>
          </xsd:choice>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
    <resheader name="resmimetype">
      <value>text/microsoft-resx</value>
    </resheader>
    <resheader name="version">
      <value>2.0</value>
    </resheader>
    <resheader name="reader">
      <value>System.Resources.ResXResourceReader, System.Windows.Forms,
  Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <resheader name="writer">
      <value>System.Resources.ResXResourceWriter, System.Windows.Forms,
  Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <data name="dateformat" xml:space="preserve">
      <value>ss, dd.mm.yyyy</value>
    </data>
    <data name="daynames" xml:space="preserve">
      <value>["Sonntag", "Montag", "Dienstag", "Mittwoch", "Donnerstag",
               "Freitag", "Samstag"]</value>
    </data>
    <data name="loading" xml:space="preserve">
      <value>Lade Datum ...</value>
    </data>
  </root>
```

Из этого XML-файла видно, какая информация помещается в файлы ресурсов:

`dateformat`

Формат представления даты с использованием символов-заполнителей

`daynames`

Локализованные названия дней недели

`loading`

Текст «Loading date...» (загрузка даты), переведенный на отдельные языки

Вам необходимо создать как минимум два файла ресурсов (для английского и немецкого языков). По своему желанию вы можете добавить локализации и для других языков. Для определения названий дней недели используйте синтаксис массивов в формате JSON (глава 3). Будьте внимательны и старайтесь не допускать опечаток, так как приложения, состоящие из серверных и клиентских сценариев, а также ресурсов, очень сложно отлаживать.

В окне Solution Explorer (обозреватель решений) в текущем проекте откройте папку *Properties* (свойства). Внутри нее вы найдете файл *AssemblyInfo.cs*, в котором содержится дополнительная информация о проекте. Если вы не видите этот файл, щелкните на кнопке Show All Files (показать все файлы), как показано на рис. 9.5.

В конец файла *AssemblyInfo.cs* добавьте следующие строки:

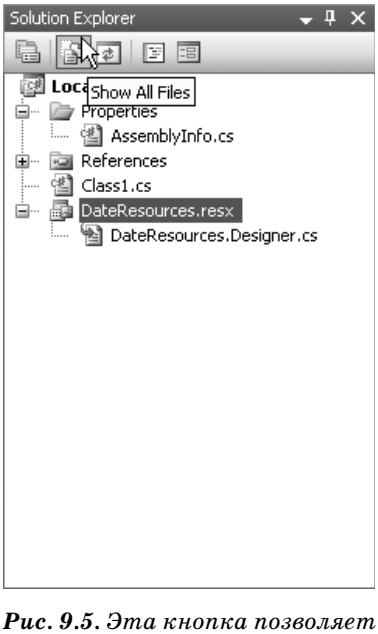


Рис. 9.5. *Эта кнопка позволяет увидеть файл AssemblyInfo.cs*

```
[assembly: System.Web.UI.WebResource("LocalizedDate.Dayname.js",
"application/xjavascript")]
[assembly: System.Web.UI.ScriptResource("LocalizedDate.Dayname.js",
"LocalizedDate.DateResources", "LocData")]
```

Обратите внимание на значение LocData в последнем аргументе. Это имя позволит обращаться ко всем данным в файле ресурсов из программного кода JavaScript посредством объекта LocData. Имя может выбираться совершенно произвольно, но это имя будет использоваться в следующих примерах.

Остальные значения в предыдущем фрагменте содержат данные, имеющие отношение к приложению. LocalizedDate – это имя проекта, *Dayname.js* – это файл JavaScript, который будет создан на следующем шаге, а DateResources будет использоваться позже для организации доступа к данным в файле ресурсов (из программного кода на C#).

Файл *Dayname.js* содержит программный код JavaScript, который определяет текущую дату и форматирует ее в соответствии с текущими региональными настройками. Как уже говорилось выше, объект LocData используется в JavaScript для доступа к информации из файла ресурсов. Свойство LocData.dateformat содержит строку формата даты, а свойство LocData.daynames – массив с названиями дней недели. Последнее значение хранится в формате JSON (глава 3), поэтому нам придется использовать функцию eval() для преобразования строки в объект JavaScript:

```
var daynames = eval("(" + LocData.daynames + ")");
```

Программный код в *Dayname.js*, подобный тому, что приводился в примере 9.3, будет замещать символы-заполнители в строке dateformat и отображать результат в окне броузера. Обратите внимание на односекундную задержку, которая была добавлена для того, чтобы сделать эффект более наглядным. В примере 9.5 приводится полный программный код файла JavaScript. Добавьте его в свой проект.

Пример 9.5. Программный код JavaScript в библиотеке классов

```
Dayname.js

Sys.Application.add_load(function() {
  var d = new Date();
  var daynames = eval("(" + LocData.daynames + ")");
  var datestring = LocData.dateformat.replace("ss", daynames[d.getDay()])
                                     .replace("dd", d.getDate())
                                     .replace("mm", d.getMonth() + 1)
                                     .replace("yyyy", d.getFullYear());
  setTimeout('$get("date").firstChild.nodeValue = "'
             + datestring + '"', 1000);
});
```

В заключение файл *Dayname.js* необходимо встроить в сборку. Щелкните на имени файла в окне Solution Explorer (обозреватель решений).

В окне Properties (свойства) установите параметр Build Action (действие по сборке) в значение Embedded Resource (встроенный ресурс), как показано на рис. 9.6. Это приведет к тому, что во время компиляции проекта файл JavaScript будет внедрен непосредственно в файл DLL библиотеки.

В программном коде из *Dayname.js* вы можете заметить вызов функции `$get()` для доступа к еще не определенному элементу с именем `"date"`. Этот элемент будет объявлен как часть управляющего кода в файле с классом на C#.

В окне Solution Explorer (обозреватель решений) переименуйте файл *Class1.cs* в *LocDateControl.cs* (для этого нужно щелкнуть правой кнопкой мыши на имени файла и выбрать в контекстном меню пункт Rename (переименовать)) и выберите вариант обновления всех ссылок. В этом файле будет располагаться программный код компонента ASP.NET на языке C#. Этот компонент будет наследовать свойства и методы класса `Sys.Web.UI.Control` платформы .NET.

В классе `LocDateControl` мы переопределим метод `CreateChildControls()` базового класса и добавим свой собственный HTML-элемент – метку `<span>` с именем `"date"`:

```
hgc = new HtmlGenericControl();
hgc.TagName = "span";
hgc.ID = "date";
```

В элементе `<span>` будет находиться текст «Loading date...» или тот, который задан для выбранного языка. Чтобы получить нужную строку,

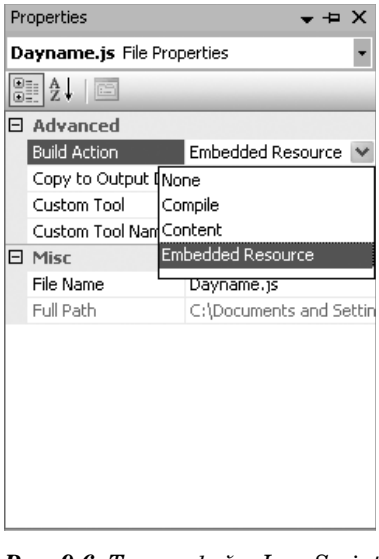


Рис. 9.6. *Теперь файл JavaScript будет встроен в файл библиотеки*

компонент должен обратиться к встроенным ресурсам. Эту операцию выполняет следующий фрагмент:

```
System.Resources.ResourceManager r = new System.Resources.ResourceManager(
  "LocalizedDate.DateResources",
  this.GetType().Assembly);
hgc.InnerHtml = r.GetString("loading");
```

И наконец, новый компонент с локализованным текстом добавляется на страницу:

```
Controls.Add(hgc);
```

В примере 9.6 приводится полный программный код для файла библиотеки классов.

Пример 9.6. Программный код для библиотеки классов

```
LocDateControl.cs

using System;
using System.Collections.Generic;
using System.Text;
using System.Web.UI;
using System.Web.UI.HtmlControls;

namespace LocalizedDate
{
  public class LocDateControl : Control
  {
    private HtmlGenericControl hgc;

    protected override void CreateChildControls()
    {
      base.CreateChildControls();

      hgc = new HtmlGenericControl();
      hgc.TagName = "span";
      hgc.ID = "date";

      System.Resources.ResourceManager r = new
      System.Resources.ResourceManager(
          "LocalizedDate.DateResources",
          this.GetType().Assembly);
      hgc.InnerHtml = r.GetString("loading");

      Controls.Add(hgc);
    }
  }
}
```

Теперь можно выполнить сборку проекта библиотеки классов. В результате будут созданы два файла. Файл *LocalizedDate.dll*, содержащий реализацию компонента в библиотеке классов, и файл сборки, содержащий файл ресурсов по умолчанию (из *DateResources.resx*). Для

каждого дополнительного языка будет создаваться каталог с сопутствующей сборкой. Например, немецкий перевод будет находиться в каталоге *de*, содержащем сборку *LocalizedDate.resources.dll*.

При использовании Visual Studio 2005 новые сборки, как правило, сразу становятся доступны в приложениях ASP.NET AJAX (если это не так, добавьте явно ссылку на библиотеку классов в проекте веб-сайта). Если вы используете Visual Web Developer, создайте каталог *Bin* в каталоге с приложением ASP.NET AJAX, а затем поместите в этот каталог копии обоих файлов *LocalizedDate.dll* и всех подкаталогов (*de*, ...).

Теперь можно импортировать компонент в любую страницу ASP.NET с помощью следующей директивы:

```
<%@ Register TagPrefix="OReilly" Assembly="LocalizedDate" Namespace="
LocalizedDate" %>
```

Поскольку этот компонент не объявляет никаких общедоступных свойств, он может быть включен в страницу с минимальными усилиями:

```
<OReilly:LocDateControl ID="ldc1" runat="server" />
```

Компонент содержит весь программный код JavaScript, необходимый для отображения экранной заставки («Loading data...») и локализованной даты. Все, что осталось сделать для ASP.NET AJAX, – это загрузить сборку в элемент управления `ScriptManager`. В элементе ScriptReference должны быть указаны оба имени – имя сборки (`localizedDate`) и каноническое имя встроенного файла JavaScript (`LocalizedDate.Dayname.js`). Не забудьте установить атрибут `EnableScriptLocalization` элемента `ScriptManager`!

```
<asp:ScriptManager ID="ScriptManager1" runat="server"
                   EnableScriptLocalization="true">
  <Scripts>
    <asp:ScriptReference Assembly="LocalizedDate"
                         Name="LocalizedDate.Dayname.js" />
  </Scripts>
</asp:ScriptManager>
```

Наконец, настроим страницу так, чтобы она автоматически определяла корректную культуру:

```
<%@ Page Language="C#" UICulture="auto" %>
```

В примере 9.7 приводится полный код страницы *.aspx*. На рис. 9.7 и 9.8 показаны различные состояния приложения.

Пример 9.7. Использование сопутствующей сборки в ASP.NET AJAX

```
Localization-Satellite.aspx

<%@ Page Language="C#" UICulture="auto" %>

<%@ Register TagPrefix="OReilly" Assembly="LocalizedDate" Namespace="
LocalizedDate" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server"
                       EnableScriptLocalization="true">
      <Scripts>
        <asp:ScriptReference Assembly="LocalizedDate"
                             Name="LocalizedDate.Dayname.js" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <OReilly:LocDateControl ID="ldc1" runat="server" />
    </div>
  </form>
```

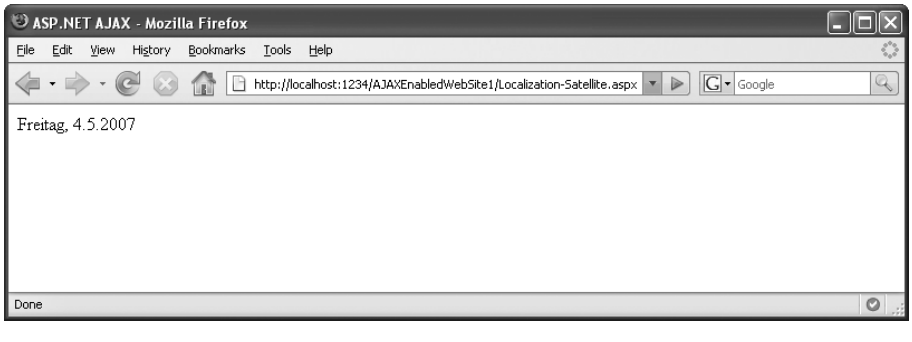


Рис. 9.7. *Страница, локализованная для немецкого языка*

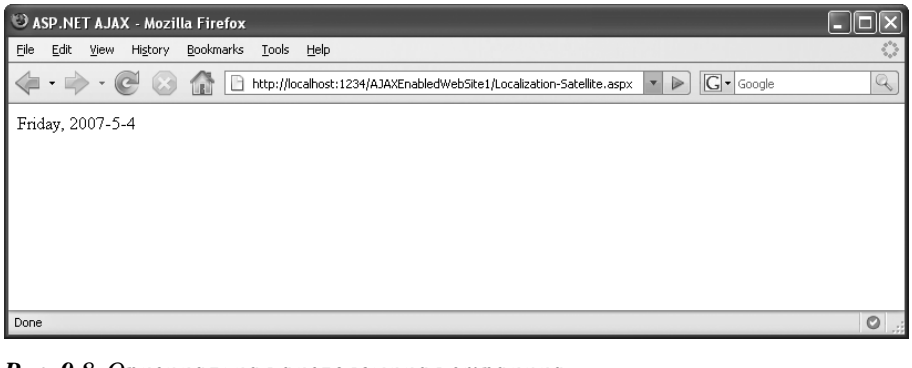


Рис. 9.8. *Оригинальная англоязычная страница*

```
</body>
</html>
```

Глобализация и интернационализация

В дополнение к поддержке локализации, ASP.NET AJAX предоставляет поддержку глобализации, которую иногда еще называют как интернационализация (или i18n, если вам нравится использовать нумеронимы). В элементе управления `ScriptManager` имеется свойство `EnableScriptGlobalization`. Если в этом свойстве установить значение `true`, ASP.NET AJAX будет в состоянии локализовать значения дат. Для этого библиотека Ajax расширяет возможности объекта `Date` из JavaScript (и других объектов – подробности в главе 4 и в приложении D), благодаря чему в этом объекте появился новый метод с именем `localeFormat()`. Этот метод форматирует значение даты в соответствии с региональными настройками в броузере. Эти настройки передаются в HTTP-заголовке `Accept-Language`. На рис. 9.9 показаны HTTP-заголовки, отправляемые броузером Firefox с настройками для немецкого языка. Обратите внимание: заголовок `Accept-Language` включает перечень поддерживаемых языков в сответствии с уровнем их предпочтения. На этой иллюстрации видно, что немецкий язык является наиболее предпочтительным, вслед за ним идет английский язык. В англоязычных броузерах обычно предпочтительным является только английский язык, иногда учитываются различия между американским,

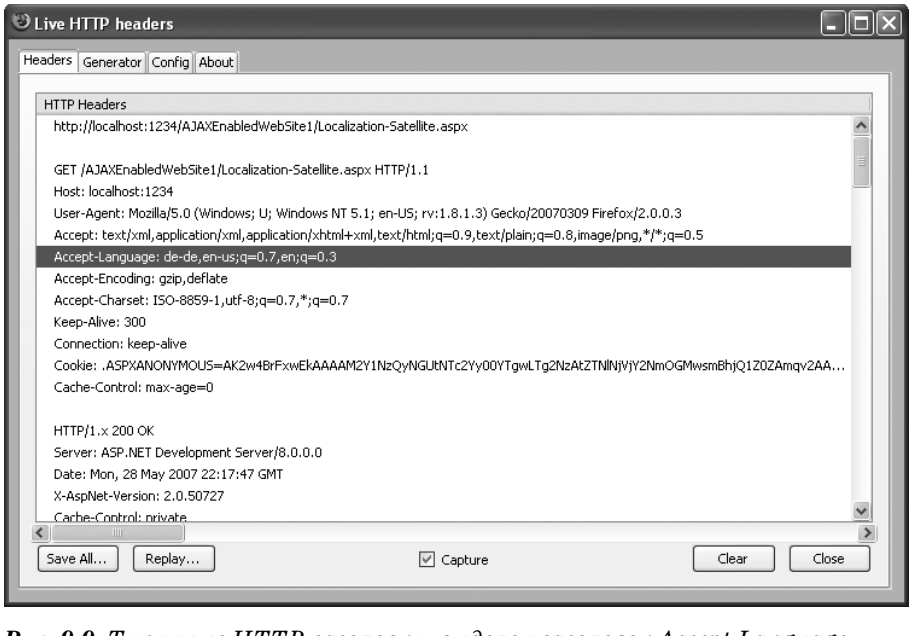


Рис. 9.9. *Типичные HTTP-заголовки, выделен заголовок Accept-Language*

британским и канадским английским. Разумеется, название языка, которое отправляется броузером, не всегда соответствует действительности, поэтому наилучшим вариантом остается возможность явного выбора языка пользователем.

Предпочтительный язык броузера может быть изменен пользователем. В Firefox для этого нужно выбрать пункт меню Правка→Настройки (Tools→Options), в открывшемся диалоге выбрать вкладку Дополнительно (Advanced) и щелкнуть на кнопке Выбрать (Choose). В диалоге Languages (Языки) (рис. 9.10) можно изменить порядок предпочтений, добавить дополнительные языки или удалить существующие.

В Internet Explorer нужно выбрать пункт меню Сервис→Свойства обозревателя (Tools→Internet Options) и в открывшемся диалоге на вкладке Общие (General) щелкнуть на кнопке Языки (Language), в результате откроется диалог, как показано на рис. 9.11.

В других броузерах также имеются похожие средства изменения языковых настроек.

Но вернемся к теме ASP.NET AJAX и глобализации значений дат. Метод `Date.localFormat()` замещает символы-заполнители локализованными названиями дней недели и месяцев. Следующий фрагмент выведет нечто похожее на «Wednesday, 1. May 2007» в зависимости от текущих языковых настроек броузера:

```
<script type="text/javascript">
  function pageLoad() {
    $get("date").firstChild.nodeValue =
      (new Date()).localeFormat("dddd, dd. MMMM yyyy");
  }
</script>
```

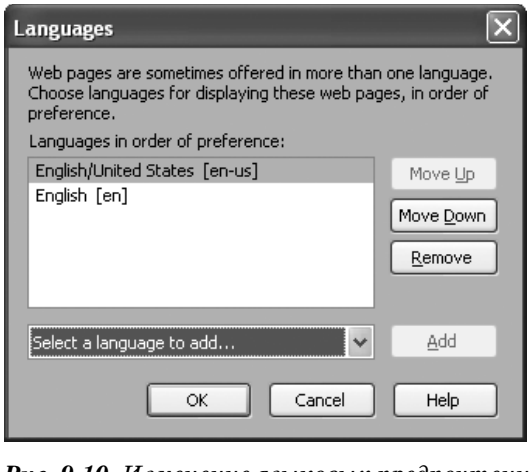


Рис. 9.10. *Изменение языковых предпочтений в броузере Firefox*

Необходимо выполнить еще один шаг. Приложение ASP.NET должно правильно установить культуру. Культура может быть установлена в файле *Web.config*, программно или с помощью разметки в странице. Ссылки на дополнительную информацию о возможных вариантах вы найдете в разделе «Для дополнительного чтения» в конце этой главы.

В следующем примере культура устанавливается с помощью директивы `@ Page`. Мы могли бы установить какую-то конкретную культуру, но предпочитаем доверить платформе ASP.NET AJAX определять корректные настройки из HTTP-заголовка `Accept-Language`. Делается это с помощью следующего объявления:

```
<%@ Page Language="C#" Culture="auto" %>
```

В примере 9.8, который на этот раз получился достаточно коротким, приводится полный код, который делает все что нужно.

Пример 9.8. Глобализация даты

```
Globalization.aspx

<%@ Page Language="C#" Culture="auto" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

Рис. 9.11. *Изменение языковых предпочтений в Internet Explorer*

```
    <title>ASP.NET AJAX</title>
    <script type="text/javascript">
    function pageLoad() {
      $get("date").firstChild.nodeValue =
        (new Date()).localeFormat("dddd, dd. MMMM yyyy");
    }
    </script>
  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server"
                         EnableScriptGlobalization="true">
      </asp:ScriptManager>
      <div>
        <span id="date"> </span>
      </div>
    </form>
  </body>
  </html>
```

На рис. 9.12 показаны результаты работы этой страницы на немецком языке, а на рис. 9.13 – на французском. В обоих случаях эти результаты

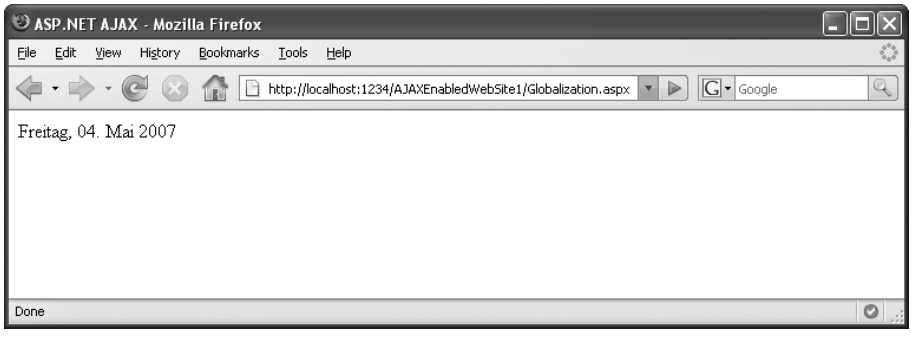


Рис. 9.12. *Дата на немецком языке*

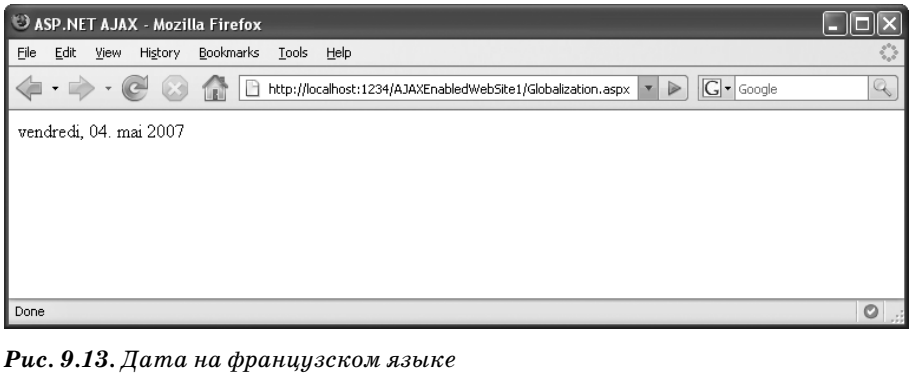


Рис. 9.13. *Дата на французском языке*

были получены простым изменением языковых настроек в броузере. На рисунках видно, что названия дня недели (символ-заполнитель `dddd`) и месяца (`MMMM`) корректно переведены на выбранный язык. Однако формат представления даты локализован не полностью: точка после числа месяца – обычное дело в немецком языке, но не во французском. Для устранения подобных проблем необходимо вручную выполнять локализацию информации о дате, как это было продемонстрировано в примерах 9.3 и 9.7.

Подведение итогов

В век глобализации веб-сайты должны обеспечивать возможность взаимодействия с пользователями из разных стран, которые говорят на разных языках. В этой главе были продемонстрированы некоторые приемы создания многоязычных веб-сайтов с использованием ASP.NET AJAX.

Для дополнительного чтения

http://msdn2.microsoft.com/en-us/library/76091f86-f967-4687-a40f-de87bd8cc9a0.aspx

Информация в MSDN о настройке языковых параметров из ASP.NET

http://ajax.asp.net/docs/tutorials/GlobalizingDateUsingClientScript.aspx

Руководство по глобализации приложений в документации компании Microsoft к ASP.NET AJAX

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-122-3, название «Программирование в ASP.NET AJAX» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

III

ASP.NET AJAX Control Toolkit

Глава 10. Использование Control Toolkit

Глава 11. Добавление анимационных эффектов в веб-страницу

Глава 12. Автодополнение ввода пользователя, борьба со спамом и не только

Глава 13. Создание собственных элементов управления и помощь сообществу

10

Использование Control Toolkit

Некоторые критики платформы ASP.NET 2.0 AJAX утверждают, что она представляет собой не более чем комбинацию элемента управления `UpdatePanel` и нескольких расширений JavaScript. Действительно, из окончательной версии были исключены некоторые функциональные возможности, которые существовали в Atlas (предварительной версии ASP.NET AJAX), с целью уменьшить размер библиотеки ASP.NET AJAX настолько, насколько это возможно (чтобы сделать платформу более приемлемой для построения крупных сайтов). Разумеется, не все возможности были удалены, и ASP.NET AJAX по-прежнему остается отличным выбором.

Пакет ASP.NET AJAX Control Toolkit был создан для того, чтобы дать возможность и Microsoft, и сообществу ASP.NET без труда добавлять в платформу дополнительные функциональные возможности вне официальной поддержки Microsoft и независимо от цикла обновлений ASP.NET AJAX. Программное обеспечение, включая открытые исходные тексты, выпускается под лицензией совместного доступа (shared source license) (Microsoft Permissive License, известной так же как MS-PL). Кроме того, в Microsoft был создан сайт, на котором разработчики компании и члены сообщества могут добавлять новые функциональные возможности. (Ссылки приводятся в разделе «Для дополнительного чтения» в конце этой главы.)

В этой главе будет показано, как установить и использовать пакет ASP.NET AJAX Control Toolkit, кроме того вы познакомитесь с некоторыми наиболее полезными компонентами, содержащимися в этом пакете. Однако, учитывая, что пакет продолжает развиваться и ежемесячно в него добавляются новые компоненты и новые возможности, информация о пакете устаревает очень быстро. Вам обязательно нужно будет проверить наличие последних обновлений и справочной информации на сайте проекта ASP.NET AJAX Control Toolkit. (Ссылки приводятся в разделе «Для дополнительного чтения» в конце этой главы.)

Установка пакета Control Toolkit

Прежде чем можно будет использовать компоненты ASP.NET AJAX, необходимо добавить пакет Control Toolkit в свою среду разработки. Загрузить пакет можно с домашней страницы ASP.NET AJAX по адресу *http://ajax.asp.net/toolkit/default.aspx?tabid=47*. Последнюю версию документации можно найти по адресу *http://ajax.asp.net/ajaxtoolkit*. Домашняя страница проекта размещается на сайте CodePlex (*http://www.codeplex.com/AtlasControlToolkit/*), откуда можно загрузить пакет в виде ZIP-архива. В действительности, пакет распространяется в виде двух архивов: в одном из них содержится сам пакет с исходными текстами; другой, меньший по размеру, не содержит исходных текстов. Для простого использования пакета достаточно будет загрузить архив, в имени которого содержится текст «NoSource» (без исходных текстов). Если вам будет интересно взглянуть на реализацию компонентов, выбирайте для загрузки архив с исходными текстами.

На рис. 10.1 показано содержимое ZIP-архива без исходных текстов.

В дополнение к файлам EULA (пользовательское соглашение) и README в архиве содержатся две папки:

SampleWebSite

Веб-сайт ASP.NET AJAX с примерами использования всех компонентов пакета и с документацией

AjaxControlExtender

Установочный пакет VSI (Visual Studio Integration – пакет интеграции с Visual Studio), который содержит несколько шаблонов для работы с пакетом в Visual Studio.

Пример веб-сайта, поставляемый в составе пакета, – это документация к пакету. Сайт построен на основе платформы ASP.NET 2.0 и позволяет увидеть компоненты пакета в действии. На рис. 10.2 показан веб-сайт документации в окне броузера. В ней не только перечислены компоненты пакета с их свойствами и информацией об использовании, но также демонстрируется каждый компонент в действии.

Прежде чем приступать к исследованию сайта, найдите минуту, чтобы запустить мастер установки *AjaxControlExtender.vsi*. На рис. 10.3 видно, что он может установить до шести шаблонов. Из них только последние два обязательны для работы с компонентами пакета. Однако вам следует установить весь набор шаблонов, так как они будут использоваться в главе 14. Предупреждения о том, что устанавливаемые компоненты не имеют цифровой подписи, можно просто игнорировать.

Мастер установки VSI содержит шаблон веб-сайта, который генерирует соответствующий файл *Web.config* для сайтов, управляемых пакетом Control Toolkit, подобно шаблонам ASP.NET AJAX и ASP.NET Futures.

Первые четыре шаблона образуют инфраструктуру для создания собственных компонентов пакета. Подробнее о том, как это делается, будет рассказываться в главе 14.

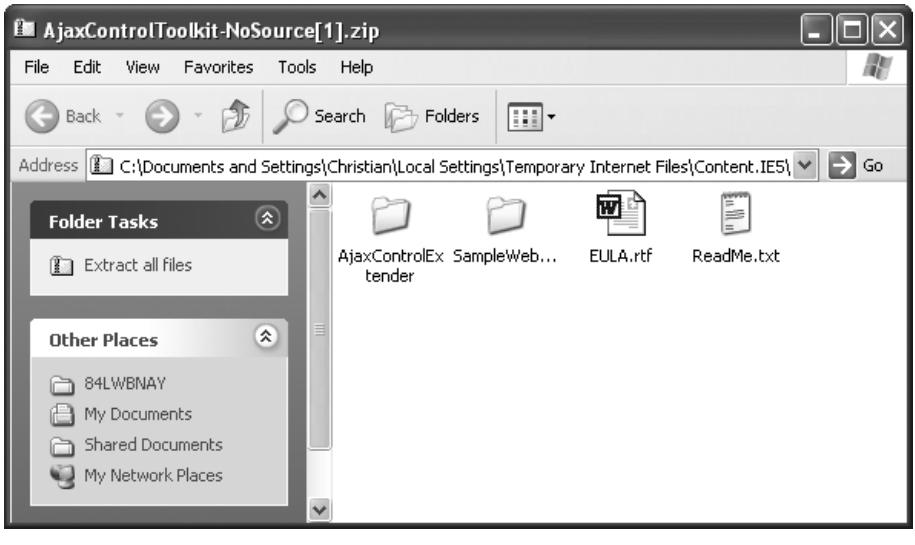


Рис. 10.1. *ZIP-архив пакета ASP.NET AJAX Control Toolkit*

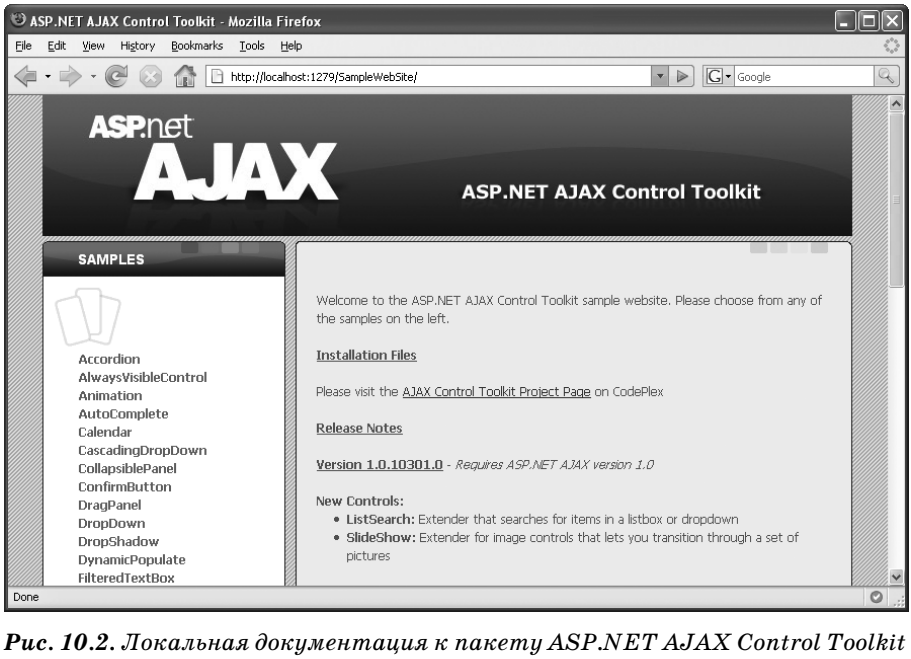


Рис. 10.2. *Локальная документация к пакету ASP.NET AJAX Control Toolkit*

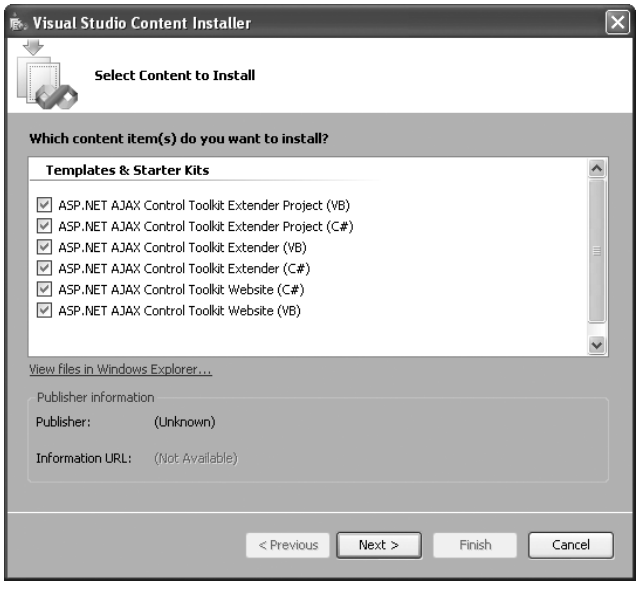


Рис. 10.3. *Мастер установки пакета интеграции VSI для работы с пакетом Control Toolkit*

Теперь создадим новый веб-сайт в среде Visual Studio или Visual Web Developer Express Edition, используя для этого шаблон AJAX Control Toolkit Web Site (рис. 10.4).

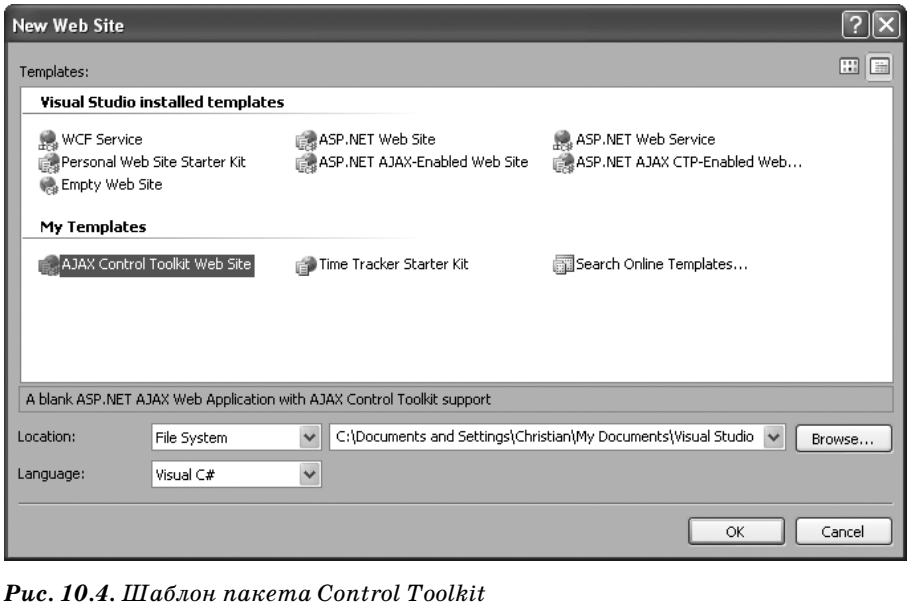


Рис. 10.4. *Шаблон пакета Control Toolkit*

Затем добавим компоненты в палитру инструментов Visual Studio. Выведите Toolbox (палитру компонентов) в режиме проектирования в среде разработки. Щелкните правой кнопкой мыши на палитре и выберите пункт контекстного меню Add Tab (добавить вкладку). Дайте новой вкладке имя *ASP.NET AJAX Control Toolkit* (имя не обязательно должно быть именно таким, можно дать любое другое имя).

Щелкните правой кнопкой мыши на вновь созданной вкладке и выберите пункт контекстного меню Choose Toolbox (выбрать палитру компонентов) (в Visual Web Developer этот пункт отображается как Choose Items (выбрать элементы)). Добавьте сборку *ASP.NET AJAX Control Toolkit*, *AjaxControlToolkit.dll*. Эта сборка находится в каталоге *Bin* текущего веб-сайта, созданного на основе шаблона AJAX Control Toolkit Web Site. После этого в палитре появится несколько новых записей, как показано на рис. 10.5. (Обратите внимание: компоненты в палитре

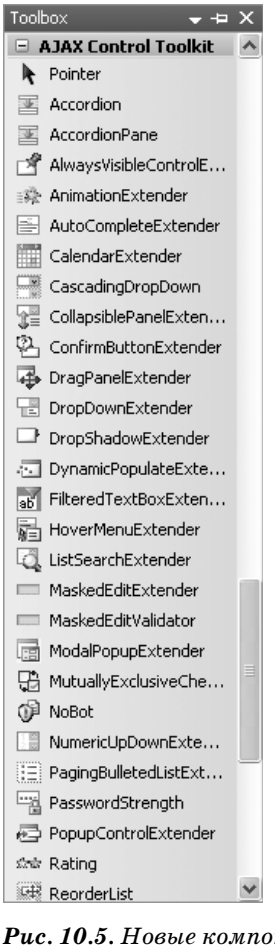


Рис. 10.5. *Новые компоненты в палитре*

не будут видны, пока не открыт ни один документ, например страница ASP.NET, в котором могут быть использованы эти компоненты.)

Теперь компоненты готовы к использованию (главы 12 и 13). Вы даже можете оказать содействие в их разработке (глава 14)!

Использование пакета Control Toolkit

После того как компоненты будут добавлены в проект, вы можете использовать их при создании своего веб-сайта. Теперь продемонстрируем, как это можно сделать, добавив в страницу один из самых простых компонентов – элемент управления `ConfirmButton`. Компонент `ConfirmButton` отображает диалог подтверждения JavaScript (с помощью функции `window.confirm()`, разумеется), который запрашивает у пользователя подтверждение на выполнение текущей операции. Если пользователь щелкает на кнопке No (нет), операция отменяется. Это бывает удобно, когда выполняется отправка формы щелчком на кнопке `LinkButton` или на обычной кнопке: если выполняется щелчок на кнопке No (нет), JavaScript отменяет щелчок на кнопке, препятствуя отправке формы.

Прежде чем в странице можно будет использовать какие-либо компоненты из пакета, необходимо сначала зарегистрировать пакет, добавив следующую разметку в страницу. После этого можно будет избежать лишнего ввода с клавиатуры, так как необходимая разметка будет добавляться автоматически при перетаскивании выбранного компонента на страницу в режиме проектирования или в режиме кодирования.

```
<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
TagPrefix="ajaxToolkit" %>
```

Всякий раз, перетаскивая компонент из палитры, используйте имя, которое вы присвоили свойству `TagPrefix`. Если вы не задали значение `TagPrefix`, то при каждом перетаскивании расширителя из палитры компонентов на проектируемую страницу среда разработки автоматически будет присваивать ему префикс `cc1`. Префикс `ajaxToolkit` выглядит более наглядно. Кроме того, в страницу нужно добавить компонент `ScriptManager`, чтобы обеспечить функционирование компонентов пакета.

Большая часть компонентов пакета ASP.NET AJAX Control Toolkit обеспечивают свою функциональность за счет расширения функциональности других элементов управления в странице. (В главе 1 приводится обсуждение пакетов расширений, которые поставляются в составе ASP.NET AJAX.) Перечень доступных свойств расширителя зависит от используемого компонента, но общий принцип работы остается неизменным: вы добавляете компонент в страницу, затем добавляете расширитель и устанавливаете в свойстве `TargetControlID` значение атрибута ID целевого HTML-элемента или элемента управления ASP.NET.

Компонент `ConfirmButton` обладает одним дополнительным свойством с именем `ConfirmText`. Это свойство содержит текст сообщения, которое будет отображаться по щелчку на кнопке `LinkButton`.

Запустите страницу и щелкните на компоненте, с которым был ассоциирован расширитель ConfirmButton. Перед вами появится диалог, запрашивающий подтверждение на продолжение операции. Если будет выбран ответ Yes (да), будет выполнено действие, предусмотренное элементом управления LinkButton, то есть произойдет отправка формы по адресу, указанному в LinkButton. После этого появится текст, подтверждающий отправку формы. Если же будет выбран ответ No (нет), действие будет отменено. Полный программный код этого примера приводится в примере 10.1.

Пример 10.1. Использование расширителя ConfirmButton

```
ConfirmButton.aspx

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
TagPrefix="ajaxToolkit" %>

<script runat="server">
  void Page_Load( )
  {
    if (Page.IsPostBack)
    {
      Label1.Text = "You have been warned!";
    }
  }
</script>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      <asp:LinkButton ID="LinkButton1"
                      runat="server">LinkButton</asp:LinkButton>
      <ajaxToolkit:ConfirmButtonExtender ID="ConfirmButtonExtender1"
        runat="server" ConfirmText="Are you sure?! "
        TargetControlID="LinkButton1" />
      <br />
      <asp:Label ID="Label1" runat="server" />
    </div>
  </form>
</body>
</html>
```

На рис. 10.6 показан результат, отображаемый броузером. После щелчка на компоненте LinkButton появляется окно диалога. Если щелкнуть на кнопке No (нет), форма не будет отправлена на сервер.

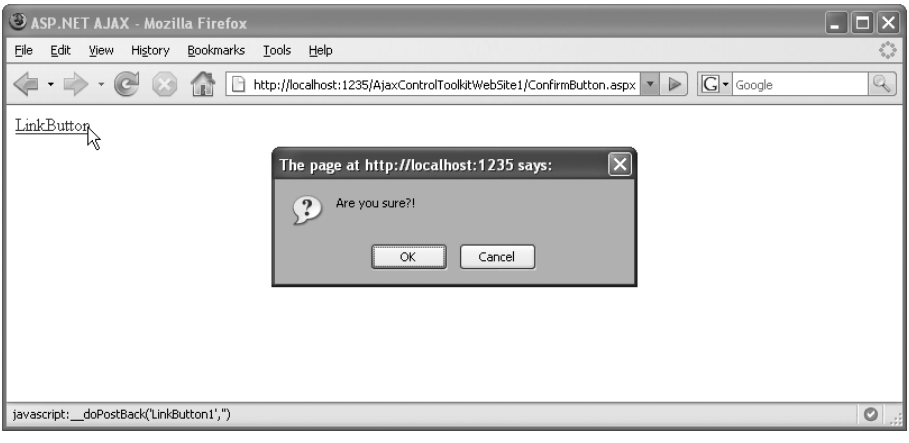


Рис. 10.6. *После щелчка на кнопке отображается текст, запрашивающий подтверждение*

Остальные расширители, входящие в состав пакета, работают аналогичным образом. Просто добавьте расширитель элемента управления (создайте элемент `<ajaxToolkit:controlExtender>`) в страницу и установите значения свойств расширителя.

С точки зрения JavaScript `ConfirmButtonExtender` реализует достаточно тривиальный эффект. Чтобы добавить запрос на подтверждение к обычной гиперссылке HTML, достаточно будет вставить следующий фрагмент программного кода JavaScript (то, что для компонента `ConfirmButtonExtender` пока недоступно):

```
<a href="http://atlas.asp.net/"
  onclick="return window.confirm('Are you
sure?!');">Go to the ASP.NET AJAX homepage</a>
```

На примере этого расширителя становится понятно, что платформа ASP.NET AJAX представляет собой не только набор инструментальных средств Ajax, но, благодаря пакету Control Toolkit, это еще и набор инструментальных средств JavaScript.

Подведение итогов

В этой главе мы познакомились с пакетом ASP.NET AJAX Control Toolkit, установили его и рассмотрели первый пример. В следующих главах будет рассказываться о наиболее интересных компонентах и особенностях, реализованных в этом проекте с открытыми исходными текстами.

Для дополнительного чтения

http://www.codeplex.com/AtlasControlToolkit

Домашняя страница проекта ASP.NET AJAX Control Toolkit на сайте CodePlex

http://ajax.asp.net/ajaxtoolkit/

Страница, где можно опробовать компоненты из пакета Control Toolkit

11

Добавление анимационных эффектов в веб-страницу

Одним из ключевых компонентов ASP.NET AJAX Control Toolkit является богатая своими возможностями платформа создания анимационных эффектов, которая обеспечивает разработчика инструментами для создания сложной анимации без необходимости писать большие фрагменты программного кода JavaScript. В этой главе мы исследуем сильные стороны данной платформы. Мы также коснемся такой родственной возможности, как поддержка технологии перетащить-и-отпустить (drag-and-drop).

Примите к сведению: для работы с примерами из этой главы необходимо установить пакет ASP.NET AJAX Control Toolkit. Вам также потребуется веб-сайт, настроенный на использование этого пакета. Подробнее о пакете Control Toolkit говорится в главе 10.

Платформа создания анимационных эффектов

Платформа ASP.NET AJAX предлагает два способа создания анимационных эффектов (оба способа не поддерживаются официально). Это либо средства пакета ASP.NET AJAX Futures, который обладает некоторыми анимационными возможностями (более подробно пакет рассматривается в главе 19), либо средства пакета ASP.NET AJAX Control Toolkit. В состав этого пакета входит не просто несколько веб-компонентов с поддержкой анимации, а целая платформа, возможности которой мы будем исследовать на примерах этой главы. Трудно сказать, какой из двух способов «победит» в конечном итоге, но я лично готов поставить на Control Toolkit, как более активно развиваемый сообществом, чем пакет Futures.

Основы анимации

Платформа создания анимационных эффектов пакета Control Toolkit содержит элемент AnimationExtender. Самое первое свойство, которое вам потребуется, – это TargetControlID, в нем должно содержаться значение атрибута ID анимируемого элемента:

```
<ajaxToolkit:AnimationExtender ID="AnimationExtender1" runat="server"
  TargetControlID="myTargetElement">
  <!-- ... -->
</ajaxToolkit:AnimationExtender>
```

Фактическое определение анимационного эффекта располагается внутри элемента AnimationExtender. Здесь вы размещаете разметку XML, которая будет управлять анимацией. Корневым элементом этой разметки является элемент <Animations>. Внутри этого узла должна быть определена следующая информация:

События

Когда запускать анимацию, то есть – событие

Типы и свойства анимации

Какие анимационные эффекты будут использоваться: плавное исчезновение, перемещение, изменение размеров или какие-то еще (подробнее о них будет говориться чуть ниже)

Пример 11.1 демонстрирует создание анимационного эффекта. Анимационный эффект представлен в виде узла <OnLoad>, расположенного внутри узла <Animation>, который будет запускаться по окончании полной загрузки страницы. В данном случае в элементе <OnLoad> описывается единственный анимационный эффект – элемент <FadeOut>. В результате целевой элемент будет постепенно исчезать в течение трех секунд (атрибут Duration), минуя 25 градаций уровня прозрачности в секунду (атрибут Fps [frames per second – кадров в секунду]). На рис. 11.1 показано окно броузера в середине воспроизведения анимационного эффекта.

Пример 11.1. Простейший эффект плавного исчезновения

```
AnimationFade.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
```

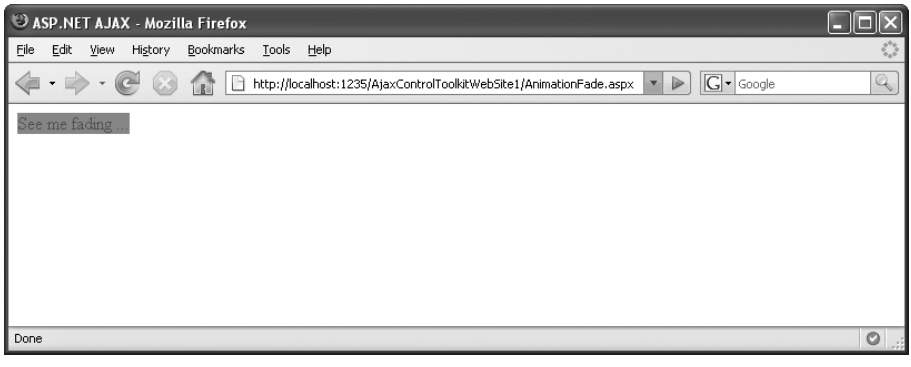


Рис. 11.1. *Плавное исчезновение элемента*

```
        <asp:Label ID="Label1" runat="server" Text="See me fading ..."
                   Style="display: inline-block; background-color: Red;" />
      </div>
      <ajaxToolkit:AnimationExtender ID="AnimationExtender1" runat="server"
        TargetControlID="Label1">
        <Animations>
          <OnLoad>
            <FadeOut Duration="3" Fps="25" />
          </OnLoad>
        </Animations>
      </ajaxToolkit:AnimationExtender>
    </form>
  </body>
  </html>
```

События для запуска анимации

Кроме `<OnLoad>` для запуска анимационных эффектов могут использоваться еще несколько событий:

`<OnClick>`

Анимация запускается щелчком мыши на элементе

`<OnHoverOut>`

Анимация запускается в момент, когда указатель мыши покидает элемент

`<OnHoverOver>`

Анимация запускается в момент, когда указатель мыши попадает в пределы элемента. (Любые анимационные эффекты, запущенные по событию `<OnHoverOut>`, останавливаются.)

`<OnMouseOut>`

Анимация запускается в момент, когда указатель мыши покидает элемент

`<OnMouseOver>`

Анимация запускается в момент, когда указатель мыши попадает в пределы элемента. (Но при этом любые анимационные эффекты, запущенные по событию `<OnHoverOut>`, *не* останавливаются.)

Типы анимационных эффектов

`<Fade>`

Плавное появление или исчезновение элемента

`<FadeIn>`

Плавное появление элемента

`<FadeOut>`

Плавное исчезновение элемента

`<Pulse>`

Пульсация элемента (чередующиеся эффекты плавного появления и исчезновения)

`<Color>`

Плавное изменение цвета элемента между двумя значениями

`<Move>`

Перемещение элемента

`<Resize>`

Изменение размера элемента

`<Scale>`

Изменение масштаба элемента

Помимо этих визуальных эффектов платформа анимации в состоянии выполнять «анимацию» числовых значений. Например, можно заставить некоторое значение изменяться в диапазоне от 1 до 100. Такое значение можно затем использовать в качестве координаты `x` элемента, значения степени прозрачности (альфа-канала), значения цвета или ширины (здесь можно привести еще массу вариантов). Ниже представлены некоторые из таких анимационных эффектов:

`<Discrete>`

Анимация значения на основе списка заданных значений

`<Interpolated>`

Анимация значения через равные промежутки на заданном интервале

`<Length>`

Анимация значения через равные промежутки на заданном интервале, но с добавлением к каждому значению строки, определяющей единицы измерения (например, `"px"` или `"em"`)

Специальные случаи анимационных эффектов, так называемые *действия* (*action*). Действия, определяемые платформой анимации, выполняются не постепенно, как описанные выше, а мгновенно. Ниже приводятся некоторые типичные действия:

`<EnableAction>`

Активирует или деактивирует элемент

`<HideAction>`

Скрывает элемент (используя для этого свойство CSS `display:none`)

`<StyleAction>`

Устанавливает свойство стиля CSS элемента

`<OpacityAction>`

Устанавливает уровень прозрачности элемента

Воспроизведение сложных анимационных эффектов

Реализация платформы анимации допускает задание только одного анимационного эффекта внутри каждого элемента-события (`<OnLoad>`, `<OnClick>` и прочих). Однако существует возможность объединять несколько простых эффектов в один, повышая их сложность, а затем использовать их внутри элемента `<OnLoad>` или внутри других элементов-событий. В зависимости от того, как запускаются отдельные эффекты внутри группы, существует несколько вариантов объединения:

`<Case>`

Запускается один из эффектов в наборе, в зависимости от условия (аналогично оператору `switch()` в C# и `Select Case` в Visual Basic)

`<Condition>`

Запускается один из эффектов, в зависимости от условия (аналогично оператору `?` в C#)

`<Parallel>`

Запускаются все эффекты одновременно

`<Sequence>`

Все эффекты запускаются последовательно, один за другим

В примере 11.2 используются сразу несколько анимационных эффектов. Два эффекта воспроизводятся одновременно (элемент `<Parallel>`). Первый эффект представлен элементом `<FadeOut>`, о котором вы уже знаете, второй – элемент `<Condition>`. В этом элементе используется условие `Math.random() < 0.5`, которое статистически должно выполняться каждый второй раз. В зависимости от значения, возвращаемого функцией `Math.random()` (случайное число в диапазоне от 0 до 1), изменяется значение свойства либо `style.top`, либо `style.left` элемента в диапазоне от 0px до 250px. На рис. 11.2 показан возможный результат. (После запуска примера попробуйте обновить содержимое страницы несколь-

ко раз, чтобы иметь возможность наблюдать два возможных анимационных эффекта.)

Пример 11.2. Одновременный запуск нескольких анимационных эффектов

```
AnimationGroup.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      <asp:Label ID="Label1" runat="server" Text="See me fading ..."
        Style="display: inline-block; background-color: Red; position:
relative; left: 8px; top: 8px;" />
    </div>
    <ajaxToolkit:AnimationExtender ID="AnimationExtender1" runat="server"
      TargetControlID="Label1">
      <Animations>
        <OnLoad>
          <Parallel>
            <FadeOut Duration="3" Fps="25" />
            <Condition ConditionScript="(Math.random( ) &lt; 0.5)">
              <Length StartValue="0" EndValue="250" Unit="px"
                Property="style" PropertyKey="left" />
              <Length StartValue="0" EndValue="250" Unit="px"
                Property="style" PropertyKey="top" />
            </Condition>
          </Parallel>
        </OnLoad>
```

Рис. 11.2. *Воспроизведение двух анимационных эффектов одновременно*

```
      </Animations>
    </ajaxToolkit:AnimationExtender>
  </form>
</body>
</html>
```

Будьте внимательны, соблюдайте синтаксис языка разметки XML. В предыдущем примере условие `Math.random() < 0.5` должно быть экранировано, поскольку открывающая угловая скобка имеет специальное значение в языке XML. Таким образом, правильная запись условия имеет вид `Math.random() < 0.5` (то есть синтаксис в примере 11.2 показан правильно).

Программирование анимационных эффектов

Любой анимационный эффект можно воспроизвести программно. При разработке сценариев в распоряжении программиста нет какого-либо прикладного интерфейса, который позволил бы по желанию добавлять или убирать отдельные анимационные эффекты, но имеется возможность связывать данные в формате XML (на стороне сервера) или в формате JSON (на стороне клиента) с элементом управления `AnimationExtender`.

Рассмотрим сначала возможность программного воспроизведения эффектов на стороне сервера. У элемента `AnimationExtender` имеется свойство `Animations`. Этому свойству можно присвоить строку с описанием анимационного эффекта в формате XML. Просто уберите узел `<Animations>` и начинайте запуск эффекта с первого события, как показано в примере 11.3. Этот пример воспроизводит тот же эффект, что и предыдущий пример, но добавляет информацию с описанием эффекта в программном коде на C#.

Пример 11.3. Воспроизведение анимации с использованием программного кода на стороне сервера

```
AnimationServer.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

  protected void Page_Load(object sender, EventArgs e)
  {
    string animations = "<OnLoad><Parallel><FadeOut Duration='3'
   Fps='25' /><Condition ConditionScript='(Math.random( ) &lt; 0.5)'><Length
   StartValue='0' EndValue='250' Unit='px' Property='style'
   PropertyKey='left' /> <Length StartValue='0' EndValue='250' Unit='px'
   Property='style' PropertyKey='top' /></Condition></Parallel></OnLoad>";
    AnimationExtender1.Animations = animations;
  }
```

```
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      <asp:Label ID="Label1" runat="server" Text="See me fading ..."
        Style="display: inline-block; background-color: Red; position:
             relative; left: 8px; top: 8px;" />
    </div>
    <ajaxToolkit:AnimationExtender ID="AnimationExtender1" runat="server"
      TargetControlID="Label1" />
  </form>
</body>
</html>
```

То же самое можно сделать и в сценарии JavaScript. Единственная разница заключается в формате представления описания анимационного эффекта. Вместо строки в формате XML необходимо использовать строку в формате JSON. Вся информация об анимационном эффекте содержится в виде объекта, представленного строкой JSON. Имя анимационного эффекта (которое идентично имени соответствующего элемента XML) определяется с помощью свойства `AnimationName`. Элементы внутри элемента XML оформляются в виде массива с помощью свойства `AnimationChildren`. Ниже приводится только строка в формате JSON, отформатированная для лучшего восприятия:

```
{
  'AnimationName':'Parallel',
  'AnimationChildren':
  [
    {
      'AnimationName':'FadeOut',
      'Duration':'3',
      'Fps':'25'
    },
    {
      'AnimationName':'Condition',
      'ConditionScript':'(Math.random( ) < 0.5)',
      'AnimationChildren':
      [
        {
          'AnimationName':'Length',
          'StartValue':'0',
          'EndValue':'250',
          'Unit':'px',
          'Property':'style',
```

```
            'PropertyKey':'left'
          },
          {
            'AnimationName':'Length',
            'StartValue':'0',
            'EndValue':'250',
            'Unit':'px',
            'Property':'style',
            'PropertyKey':'top'
          }
        ]
      }
    ]
  }
```

Кроме того, вам может потребоваться средство доступа к элементам управления в странице, поскольку сам по себе элемент `AnimationExtender` не имеет визуального представления. Платформа ASP.NET AJAX предоставляет вспомогательную функцию `$find()`, которая позволяет получить доступ к подобным элементам управления. Получив с ее помощью ссылку на требуемый элемент управления, можно связать строку в формате JSON с одним из анимационных событий. Например, метод `set_OnClick()` позволяет назначить запуск анимационного эффекта по щелчку на анимированном элементе.

В примере 11.4 приводится полный программный код. Результат работы этого примера тот же самый, что и в примерах 11.2 и 11.2, но на этот раз анимационный эффект описывается внутри клиентского сценария.

Пример 11.4. Воспроизведение анимации с использованием программного кода на стороне клиента

```
AnimationClient.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
  <script type="text/javascript">
    function pageLoad( ) {
      var animations =
"{'AnimationName':'Parallel','AnimationChildren':
[{'AnimationName':'FadeOut','Duration':'3','Fps':'25'},
{'AnimationName':'Condition','ConditionScript':
'(Math.random( ) < 0.5)','AnimationChildren':
[{'AnimationName':'Length','StartValue':'0',
'EndValue':'250','Unit':'px','Property'
:'style','PropertyKey':'left'},
{'AnimationName':'Length','StartValue':'0',
```

```
'EndValue':'250','Unit':'px','Property':
'style','PropertyKey':'top'}]}]}";
      var extender = $find("AnimationExtender1");
      extender.set_OnLoad(animations);
    }
</script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      <asp:Label ID="Label1" runat="server" Text="See me fading ..."
        Style="display: inline-block; background-color: Red;
                        position: relative; left: 8px; top: 8px;" />
    </div>
    <ajaxToolkit:AnimationExtender ID="AnimationExtender1" runat="server"
      TargetControlID="Label1" />
  </form>
</body>
</html>
```

Платформа создания анимационных эффектов обладает еще целым рядом возможностей. В дополнение к тем что были продемонстрированы в этой главе, она допускает расширение, обеспечивая большую гибкость и принимая на себя существенную долю труда программиста по написанию программного кода JavaScript. Те, кого это заинтересовало, найдут массу справочной информации и разбор программ на сайте с примерами использования компонентов из пакета ASP.NET AJAX Control Toolkit (который называется *SampleWebsite*).

Перетащить-и-отпустить

Реализация техники перетащить-и-отпустить (drag-and-drop) считается одной из самых сложных задач для разработчиков на JavaScript. Поддержкой техники перетащить-и-отпустить обладают все броузеры, но слишком велики различия в тонкостях ее реализации для разных типов броузеров. К счастью, в состав пакета ASP.NET AJAX Control Toolkit входит расширитель `DragPanel`, который дополняет компонент ASP.NET `Panel` возможностью «перетаскивания». Чтобы сделать элементы доступными для перетаскивания, достаточно просто поместить их в элемент `<asp:Panel>` и добавить в страницу компонент `DragPanelExtender`. Затем требуется задать следующие свойства элемента-расширителя:

`TargetControlID`

Значение атрибута ID элемента `Panel`

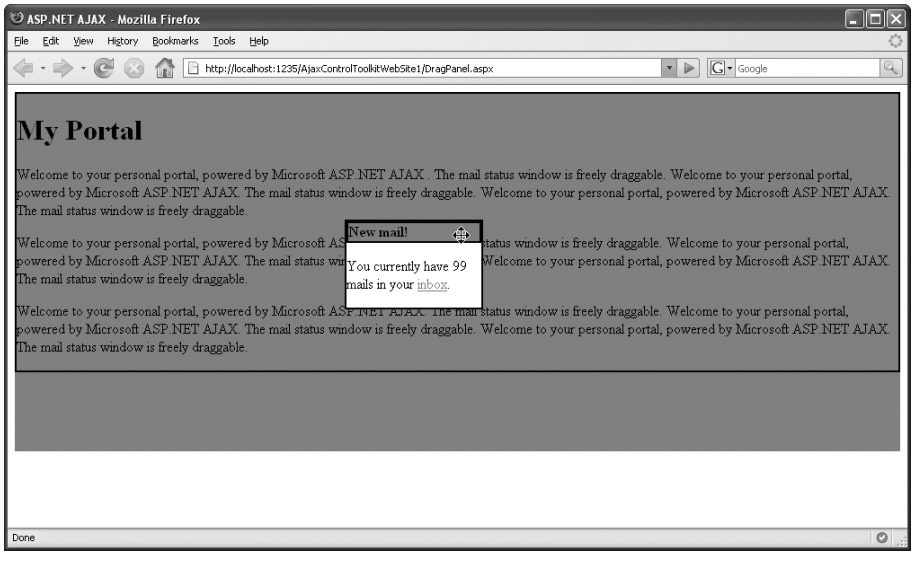


Рис. 11.3. *Панель с возможностью перетаскивания*

`DragHandleID`

Значение атрибута ID элемента, выступающего в роли «рукоятки», за которую можно будет «ухватиться» мышью (в идеале должен размещаться внутри перетаскиваемой панели)

В примере 11.5 имеется две панели: одна содержит некоторый текст, а другая содержит информацию о получении случайного числа новых писем в почтовом ящике. Эта вторая панель может перетаскиваться мышью благодаря использованию элемента управления `DragPanelExtender`. На рис. 11.3 показан конечный результат.

Пример 11.5. Создание «перетаскиваемой» панели

```
DragPanel.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  protected void Page_Load(object sender, EventArgs e)
  {
    inbox.Text = new Random().Next(0, 100).ToString( );
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
```

```
    <style type="text/css">
      .box { border: solid 2px black; }
      .mailbox { border: solid 2px black; width: 150px;
                                          background-color:  white;}
      .mailboxHeader {border: solid 2px black; background-color: blue;
                                            font-weight: bold; cursor: move;}
    </style>
  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server">
      </asp:ScriptManager>
      <div style="height: 400px; background-color: purple">
        <asp:Panel ID="ContentPanel" CssClass="box" runat="server">
          <h1>My Portal</h1>
          <p>
            Welcome to your personal portal, powered by Microsoft ASP.NET AJAX.
           The mail status window is freely draggable. Welcome to your personal
            portal, powered by Microsoft ASP.NET AJAX.
           The mail status window is freely draggable. Welcome to your personal
            portal, powered by Microsoft ASP.NET AJAX.
            The mail status window is freely draggable.
          </p>
          <p>
            Welcome to your personal portal, powered by Microsoft ASP.NET AJAX.
           The mail status window is freely draggable. Welcome to your personal
            portal, powered by Microsoft ASP.NET AJAX.
           The mail status window is freely draggable. Welcome to your personal
            portal, powered by Microsoft ASP.NET AJAX.
            The mail status window is freely draggable.
          </p>
          <p>
            Welcome to your personal portal, powered by Microsoft ASP.NET AJAX.
           The mail status window is freely draggable. Welcome to your personal
            portal, powered by Microsoft ASP.NET AJAX.
           The mail status window is freely draggable. Welcome to your personal
            portal, powered by Microsoft ASP.NET AJAX.
            The mail status window is freely draggable.
          </p>
        </asp:Panel>
        <asp:Panel CssClass="mailbox" ID="DragPanel" runat="server">
          <div id="DragPanelHandle" runat="server"
                                  class="mailboxHeader">New mail!</div>
          <p>
            You currently have <asp:Label id="inbox"
                                    runat="server"></asp:Label>
            mails in your <a href="http://www.hotmail.com/">inbox</a>.
          </p>
        </asp:Panel>
        <ajaxToolkit:DragPanelExtender ID="DragPanelExtender1" runat="server"
          TargetControlID="DragPanel" DragHandleID="DragPanelHandle" />
```

```
    </div>
  </form>
</body>
</html>
```

Обратите внимание на фиолетовую область внутри страницы. Компонент `DragPanelExtender` использует текущий блочный элемент, в котором он находится, в качестве области сброса (область, где панель может быть сброшена). В данном случае для создания такой области сброса был использован элемент `<div>` высотой 400 пикселов; фиолетовый фон показывает, где находится эта область.

Подведение итогов

Не всякое веб-приложение становится лучше от добавления анимационных эффектов. Однако если они действительно необходимы, попробуйте воспользоваться платформой `Animation`, входящей в состав пакета ASP.NET AJAX Control Toolkit. А если требуется реализация техники перетащить-и-отпустить (drag-and-drop) – обратите внимание на расширитель `DragPanel`.

Для дополнительного чтения

http://ajax.asp.net/ajaxtoolkit/Animation/Animation.aspx

Документация с описанием платформы создания анимационных эффектов

http://ajax.asp.net/ajaxtoolkit/DragPanel/DragPanel.aspx

Документация с описанием расширителя `DragPanel`

12

Автодополнение ввода пользователя, борьба со спамом и не только

Пакет ASP.NET AJAX Control Toolkit содержит слишком много элементов управления, чтобы их все можно было здесь перечислить. И хотя управление проектом, в конечном счете, находится в руках Microsoft и поддержка осуществляется отдельными членами команды Microsoft AJAX, нет никакой гарантии, что прикладные интерфейсы и функциональность пакета останутся неизменными. Компания не обеспечивает поддержку пользователей, но все-таки принимает (фактически, поощряет) содействие третьих лиц. При этом печатная книга способна отразить лишь состояние дел, имевшее место на момент ее написания. Таким образом, в этой главе будут продемонстрированы лишь некоторые понравившиеся мне элементы управления, которые существовали на момент издания книги.

Проект ASP.NET AJAX Control Toolkit развивается очень динамично. Постоянно вносятся какие-то изменения то в один его компонент, то в другой. Однако команда разработчиков Control Toolkit стремится сохранить прикладной интерфейс пакета стабильным, если внесение изменений не вызвано уважительными причинами, поэтому не надо полагать, что вас ждет чрезмерно много сюрпризов. В этой главе я решил ограничиться вводными примерами для наиболее важных (пожалуй) элементов управления. Необходимые дополнения в листингах примеров будут выкладываться на веб-сайте этой книги (адрес приводится в предисловии).

Создание раздвижных панелей

Во многих исследованиях по удобству работы авторы советуют не вынуждать пользователей производить прокрутку страниц. От прокрутки

можно будет отказаться, добавив лишь немного программного кода JavaScript. Элемент управления ASP.NET AJAX Control Toolkit – `Accordion` координирует одновременное представление нескольких панелей так, что в каждый момент времени видна будет только одна из них. Щелчок на панели приводит к ее открытию (с использованием анимационного эффекта) и одновременному скрытию содержимого всех остальных панелей.

Компонент `Accordion` состоит из двух частей: заголовка, в котором отображается название и который всегда видим, и области содержимого, которая может скрываться или появляться, напоминая эффект сжатия и растягивания мехов аккордеона. Реализацию эффекта выдвижения панели мы начнем с определения двух классов CSS в новом файле *.aspx*, один будет предназначен для заголовка, другой – для области содержимого. Эти классы определяют внешнее представление элемента управления (и, в некоторой степени, его поведение) и позднее будут использоваться при создании панелей и их содержимого.

```
<style type="text/css">
  .accordionHeader { background-color: blue; border: solid; cursor: pointer; }
  .accordionContent { fbackground-color: white; border: solid; }
</style>
```

В определении стиля заголовка используется свойство CSS `cursor: pointer`. Когда указатель мыши будет находиться над панелью, при определении которой был использован класс CSS `accordionHeader`, он будет приобретать форму кисти с вытянутым указательным пальцем, как это происходит с гиперссылками. Тем самым мы информируем пользователя, что он может щелкнуть на панели.

В фактическом определении элемента управления `Accordion` имеются ссылки на оба класса CSS:

```
<ajaxToolkit:Accordion ID="Accordion1" runat="server"
  ContentCssClass="accordionContent" HeaderCssClass="accordionHeader">
```

Внутри компонента `Accordion` элемент `<Panes>` содержит список выдвижных панелей, представленных элементами управления `AccordionPane`. Каждый элемент `AccordionPane` состоит из двух частей:

`<Header>`

Заголовок панели

`<Content>`

Содержимое панели

Благодаря поддержке технологии IntelliSense практически невозможно допустить ошибку при создании страницы с элементом управления `Accordion`. В примере 1.1 приводится полный исходный код примера, а на рис. 12.1 – результат его работы в броузере.

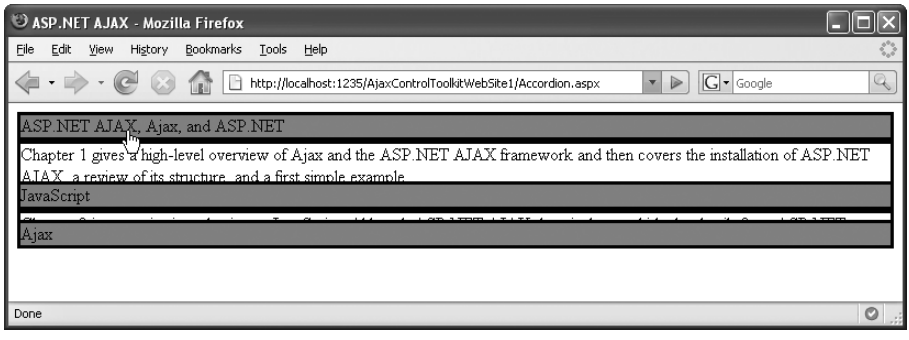


Рис. 12.1. *Создание эффекта выдвижения панелей без программного кода*

Пример 12.1. Использование элемента управления Accordion

```
Accordion.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
  <style type="text/css">
    .accordionHeader {background-color:blue; border:solid; cursor:pointer;}
    .accordionContent {background-color:white; border:solid;}
  </style>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      <ajaxToolkit:Accordion ID="Accordion1" runat="server"
        ContentCssClass="accordionContent" HeaderCssClass="accordionHeader">
        <Panes>
          <ajaxToolkit:AccordionPane ID="AccordionPane1" runat="server">
            <Header>ASP.NET AJAX, Ajax, and ASP.NET</Header>
            <Content>Chapter 1 gives a high-level overview of Ajax and the
ASP.NET AJAX framework and then covers the installation of ASP.NET AJAX,
a review of its structure, and a first simple example.</Content>
          </ajaxToolkit:AccordionPane>
          <ajaxToolkit:AccordionPane ID="AccordionPane2" runat="server">
            <Header>JavaScript</Header>
              <Content>Chapter 2 is a concise introduction to JavaScript.
Although ASP.NET AJAX does its best to hide the details from ASP.NET
programmers, a certain knowledge of JavaScript is required to really master
ASP.NET AJAX.</Content>
          </ajaxToolkit:AccordionPane>
          <ajaxToolkit:AccordionPane ID="AccordionPane3" runat="server">
```

```
            <Header>Ajax</Header>
              <Content>Chapter 3 explains the technologies beyond the hype.
You learn what happens in the background, how Ajax works, and what it really
is all about, in fewer than 20 pages.</Content>
          </ajaxToolkit:AccordionPane>
        </Panes>
      </ajaxToolkit:Accordion>
    </div>
  </form>
</body>
</html>
```

Управление относительным положением элемента

Каскадные таблицы стилей (CSS) позволяют позиционировать HTML-элементы в любом месте страницы с помощью координат x и y. Однако положение элемента управления может изменяться при прокрутке страницы пользователем. Добавив немного программного кода JavaScript, можно обеспечить неизменное положение элемента в окне броузера, независимо от того, в какую сторону пользователь выполняет прокрутку.

Как вы наверное и ожидали, ASP.NET AJAX Control Toolkit предлагает свое решение этой задачи – элемент управления AlwaysVisibleControlExtender. Этот расширитель может быть подключен к любому элементу управления, чтобы обеспечить ему неизменность положения относительно границ окна броузера. Например, свойство HorizontalSide расширителя может принимать значения Center, Left и Right, а свойство VerticalSide – значения Top, Middle или Bottom. Имеется также возможность указать величину смещения относительно левой и верхней границы окна броузера с помощью свойств HorizontalOffset и VerticalOffset. Ниже приводится пример разметки расширителя, который позиционирует элемент HTML в верхнем левом углу окна броузера:

```
<ajaxToolkit:AlwaysVisibleControlExtender ID="AlwaysVisibleControlExtender1"
  runat="server" TargetControlID="banner" HorizontalSide="Left"
  VerticalSide="Top" />
```

В примере 12.2 содержится полный исходный код, а результат его работы приводится на рис. 12.2. На рисунке видно, что баннер остается в верхнем левом углу окна и при прокрутке страницы.

Пример 12.2. Использование AlwaysVisibleControlExtender

```
AlwaysVisible.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

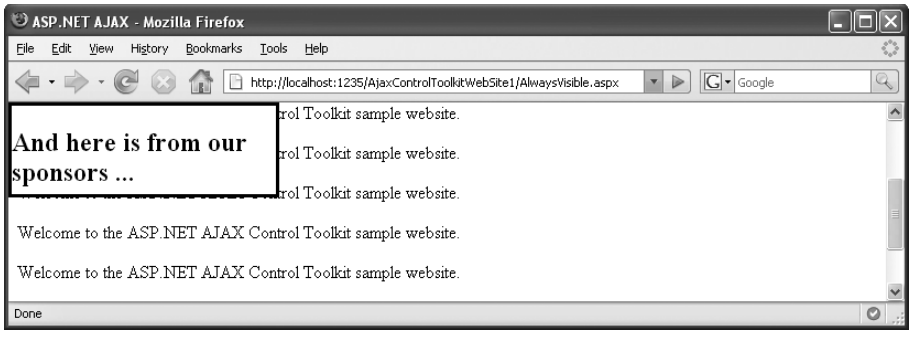


Рис. 12.2. *Баннер остается в верхнем левом углу окна даже после прокрутки страницы*

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div id="banner" style="background-color: white; border: solid; width:
          240px; height: 80px;" runat="server">
      <h2>And here is from our sponsors ...</h2>
    </div>
    <ajaxToolkit:AlwaysVisibleControlExtender ID="
          AlwaysVisibleControlExtender1" runat="server"
      TargetControlID="banner" HorizontalSide="Left" VerticalSide="Top" />
    <p>Welcome to the ASP.NET AJAX Control Toolkit sample website. </p>
    <p>Welcome to the ASP.NET AJAX Control Toolkit sample website. </p>
    <p>Welcome to the ASP.NET AJAX Control Toolkit sample website. </p>
    <p>Welcome to the ASP.NET AJAX Control Toolkit sample website. </p>
    <p>Welcome to the ASP.NET AJAX Control Toolkit sample website. </p>
    <p>Welcome to the ASP.NET AJAX Control Toolkit sample website. </p>
    <p>Welcome to the ASP.NET AJAX Control Toolkit sample website. </p>
    <p>Welcome to the ASP.NET AJAX Control Toolkit sample website. </p>
    <p>Welcome to the ASP.NET AJAX Control Toolkit sample website. </p>
    <p>Welcome to the ASP.NET AJAX Control Toolkit sample website. </p>
  </form>
</body>
</html>
```

Добавление функции автодополнения к элементу управления TextBox

Применение технологий Ajax делает веб-приложения все более и более похожими на обычные настольные приложения. Одна из особенностей, которая имеется в некоторых настольных приложениях, но отсутствует

в веб-приложениях, – это функция автодополнения. Когда в текстовое поле вводится некоторая информация, приложение отыскивает данные, подходящие для дополнения уже введенной информации (в большинстве броузеров, например, выводится список ранее веденных данных в аналогичные поля), и предлагает вариант заполнения поля.

Первым, самым известным веб-приложением, в котором появилась поддержка этой функции, стал сайт Google Suggest (*http://www.google.com/webhp?complete=1&hl=en*). Когда пользователь начинает ввод текста в поле, веб-страница предлагает наиболее популярные варианты строки поиска. Кроме того, страница показывает примерное количество результатов поиска по каждой из строк, как показано на рис. 12.3. Теперь вы знаете, как это сделано: объект `XMLHttpRequest` обращается к веб-службе, которая возвращает возможные варианты строки поиска и предполагаемое число результатов.

Платформа ASP.NET AJAX предоставляет расширитель элемента управления `AutoCompleteExtender`, который используется именно для этих целей – поиск данных в фоновом режиме и представление этих данных в элементе формы. При решении данной задачи для отображения подсказок и обеспечения возможности навигации по ним необходимо использовать каскадные таблицы стилей и программный код на JavaScript. Все это уже реализовано в расширителе из пакета Control Toolkit, а вам нужно лишь воспользоваться этими возможностями и создать веб-службу, которая будет возвращать данные. Однако следует заметить, что некоторые наиболее интересные особенности Google Suggest (включая навигацию по списку с помощью клавиатуры) в расширителе реализованы не полностью.

Рис. 12.3. *Google Suggest*

С точки зрения веб-компонентов единственный элемент, для которого функция автодополнения имеет смысл, – это TextBox. Этот элемент демонстрируется в следующем фрагменте:

```
<asp:TextBox ID="vendor" runat="server"></asp:TextBox>
```

Затем необходимо вставить элемент управления AutoCompleteExtender. Этот элемент имеет несколько свойств, которые используются для настройки эффекта автодополнения. Наиболее важными из них являются следующие:

TargetControlID

Значение атрибута ID элемента управления, который должен быть дополнен функцией автодополнения

ServicePath

Путь к веб-службе, которая генерирует данные для автодополнения

ServiceMethod

Метод веб-службы, который вызывается для получения данных автодополнения

Ниже приводится пример разметки, в которой определяется расширитель:

```
<ajaxToolkit:AutoCompleteExtender runat="server"
  ServicePath="Vendors.asmx" ServiceMethod="GetVendors"
  TargetControlID="vendor" />
```

В примере 12.3 демонстрируется страница ASP.NET с текстовым полем, который поддерживает возможность автодополнения.

Пример 12.3. Добавление функции автодополнения к текстовому полю ввода

```
AutoComplete.aspx

<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
TagPrefix="ajaxToolkit" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <asp:TextBox ID="vendor" runat="server"></asp:TextBox>
    <input type="button" value="Display Information"
           onclick="window.alert('not implemented!');" />
    <ajaxToolkit:AutoCompleteExtender runat="server"
        ServicePath="Vendors.asmx" ServiceMethod="GetVendors"
```

```
        TargetControlID="vendor" />
    </form>
  </body>
  </html>
```

Кромет того, вам необходимо реализовать веб-службу, которая возвращала бы необходимые данные. Веб-служба должна включать метод с определенной сигнатурой, который будет вызываться расширителем. Сигнатура метода приводится ниже:

```
public string[] <MethodName>(string prefixText, int count)
```

Метод принимает два параметра, назначение которых достаточно очевидно:

`prefixText`

Текст, введенный пользователем в текстовое поле, который будет использоваться в качестве префикса для всех совпадений

`count`

Максимальное число возвращаемых результатов

Данные должны возвращаться в виде массива строк, поэтому, к сожалению, вы не можете использовать наборы записей из базы данных или нечто похожее.

В примере веб-службы для реализации функции автодополнения используется база данных *AdventureWorks*. В качестве результата клиенту возвращаются названия компаний всех производителей. Как обычно, вам придется адаптировать строку подключения под свои условия – в программном коде предполагается, что SQL Server 2005 Express Edition доступен с использованием типа аутентификации Windows и строкой подключения `(local)\SQLEXPRESS`.

Сначала программный код веб-службы проверяет строку поиска. Для простоты примера будем считать допустимыми только символы от `a` до `z` (верхнего и нижнего регистров). Эта проверка необходима для предотвращения инъекции SQL-кода, потому что программа будет запускать поисковый запрос, содержащий выражение `LIKE` (подробнее об этом говорится во врезке «Внимание: опасайтесь инъекции SQL-кода»). В качестве альтернативы можно использовать параметризованные запросы.

Исследование данных, отправляемых платформой ASP.NET AJAX

Прослушивание трафика, отправляемого объектом `XMLHttpRequest`, поможет вам понять, какие данные отправляются серверу. А это, как видно из рис. 12.4, очень хорошая способ исследования внутренних механизмов работы ASP.NET AJAX с серверными компонентами.

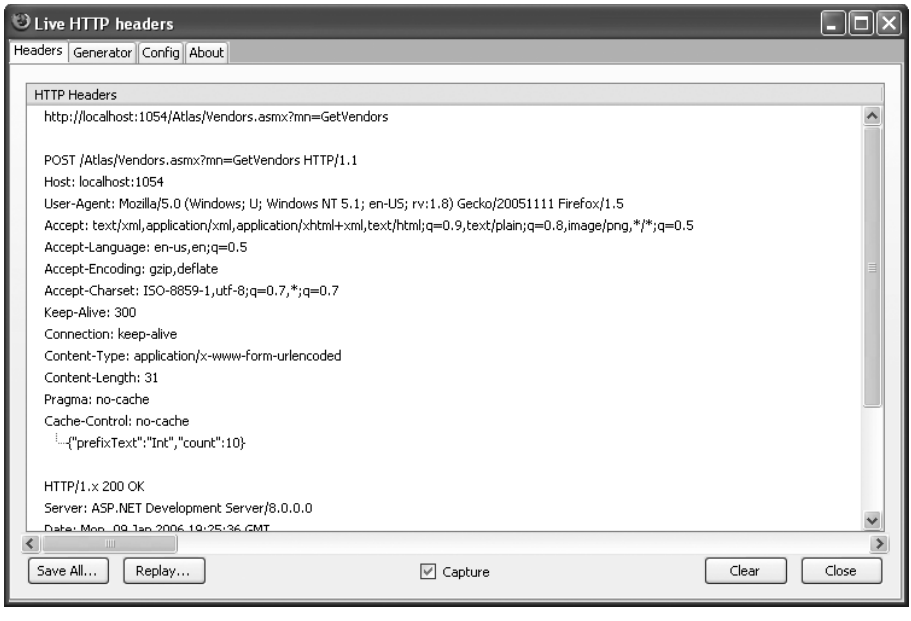


Рис. 12.4. *Инструменты, такие как Live HTTP headers, позволяют сделать тайное явным*

Если для проверки входных данных используется регулярное выражение, желательно учитывать возможность появления символов, используемых в иностранных языках, таких как немецкие символы с умляутами или французские символы с диакритическими знаками. «Опасными» символами с точки зрения безопасности являются апострофы, кавычки, квадратные скобки, символ подчеркивания, двойной дефис, точка с запятой и знак процента. Эти символы (и их закодированные версии) имеют специальное значение в пределах запроса. Поэтому для проверки ввода пользователя лучше использовать методику *белого списка* (когда имеется предопределенный набор допустимых символов), а не *черного списка* (предопределенный набор недопустимых символов).

Веб-служба также будет проверять значение параметра `count`, передаваемого методу: оно должно быть положительным числом и не больше 100 (чтобы предотвратить возможность атак типа «отказ в обслуживании»).

Программный код веб-службы, выполняющий все эти проверки, выглядит примерно следующим образом:

```
using System.Text.RegularExpressions;
...
[WebMethod]
public string[] GetVendors(string prefixText, int count)
{
```

```
Regex regex = new Regex("^[a-zA-Z ]*$");
if (!regex.IsMatch(prefixText) || count < 1 || count > 100)
{
  return null;
}
```

После проверки данных производится сборка SQL-запроса, который затем отправляется в базу данных. Типичный запрос выглядит следующим образом:

```
SELECT TOP 10 Name FROM Purchasing.Vendor WHERE NAME LIKE 'Int%'
```

Здесь предполагается, что параметр `count` имеет значение `10` (что фактически является значением по умолчанию, которое отправляет ASP.NET AJAX) и пользователь ввел строку `Int` в текстовое поле. Ниже приводится полный программный код, выполняющий SQL-запрос к базе данных, включая возврат результатов в виде множества записей:

```
  SqlConnection conn = new SqlConnection(
    "server=(local)\\SQLEXPRESS; Integrated Security=true; Initial
Catalog=AdventureWorks");
  conn.Open( );
  SqlCommand comm = new SqlCommand(
    "SELECT TOP " +
      count +
```

Внимание: опасайтесь инъекции SQL-кода

Возможность инъекции SQL-кода – одна из самых опасных уязвимостей в современных веб-приложениях. Опасность возникает при необходимости использовать динамические данные, получаемые от пользователя, для создания SQL-запроса. Например, взгляните на фрагмент ниже, где веб-служба выполняет конкатенацию строк при создании SQL-команды:

```
SqlCommand comm = new SqlCommand(
  "SELECT TOP " +
    count +
    " Name FROM Purchasing.Vendor WHERE Name LIKE '" +
    PrefixText +
    "%'",
  conn);
```

Теперь представьте себе, что переменная `count` хранит не целое число или переменная `prefixText` не проверяется на наличие «опасных» значений. Такой фрагмент может превратиться в серьезную опасность. Эту уязвимость можно эксплуатировать самыми разными способами, например, допустим, что `prefixText` имеет следующее значение:

```
' OR 2 > 1 --
```

Тогда окончательная SQL-команда будет иметь вид:

```
SELECT TOP 10 Name FROM Purchasing.Vendor
WHERE Name LIKE '' OR 2 > 1 -- %'
```

Она вернет 10 записей из таблицы, но не 10 первых записей, в которых поле Name содержит некоторый конкретный символ. Существуют и другие, более опасные варианты использования этой уязвимости.

Обычно такого рода атаки можно предотвратить с помощью параметризованных запросов, когда для всех значений, получаемых от пользователя, в тексте предложения WHERE запроса предусматриваются символы-заполнители, которые затем замещаются пользовательскими данными. К сожалению, такой прием невозможно использовать в нашем случае, потому что мы добавляем шаблонный символ % в конец строки, полученной от пользователя. Как вариант можно было бы использовать параметризованный запрос, а шаблонный символ % добавлять в конец строки из символов-заполнителей (при этом проверьте, чтобы данные-заполнители не содержали таких служебных символов, как знак процента [%] и символ подчеркивания[_]). Следовательно, программный код сначала должен проверять значение prefixText и прерывать свою работу при обнаружении недопустимых символов.

```
      " Name FROM Purchasing.Vendor WHERE Name LIKE '" +
      prefixText +
      "%'",
    conn);
  SqlDataAdapter adap = new SqlDataAdapter(comm);
  DataSet ds = new DataSet( );
  adap.Fill(ds);
```

Далее данные необходимо преобразовать в массив строк. Этот массив должен содержать не более, чем count элементов (метод Math.Min() обеспечит возврат не более count элементов, даже если в базе данных имеется большее число записей).

Это легко можно сделать с помощью цикла for:

```
  string[] vendors = new string[Math.Min(count, ds.Tables[0].Rows.Count)];
  for (int i = 0; i < Math.Min(count, ds.Tables[0].Rows.Count); i++)
  {
    vendors[i] = ds.Tables[0].Rows[i].ItemArray[0].ToString( );
  }
  return vendors;
}
```

Полный программный код реализации этой веб-службы приводится в примере 12.4. Как говорилось в главах 1 и 5, чтобы платформа

ASP.NET AJAX могла создать объект-посредник на Javascript для доступа к службе, необходимо использовать атрибут [ScriptService]**.**

Пример 12.4. Веб-служба, которая возвращает возможные совпадения

```
Vendors.asmx

<%@ WebService Language="C#" Class="Vendors" %>

using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Data;
using System.Data.SqlClient;
using System.Text.RegularExpressions;

[WebService(Namespace = "http://hauser-wenz.de/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.Web.Script.Services.ScriptService]
public class Vendors : System.Web.Services.WebService
{

  [WebMethod]
  public string[] GetVendors(string prefixText, int count)
  {
    Regex regex = new Regex("^[a-zA-Z ]*$");
    if (!regex.IsMatch(prefixText) || count < 1 || count > 100)
    {
      return null;
    }
    SqlConnection conn = new SqlConnection(
      "server=(local)\\SQLEXPRESS; Integrated Security=true; Initial
Catalog=AdventureWorks");
    conn.Open( );
    SqlCommand comm = new SqlCommand(
      "SELECT TOP " +
        count +
        " Name FROM Purchasing.Vendor WHERE Name LIKE '" +
        prefixText +
        "%'",
      conn);
    SqlDataAdapter adap = new SqlDataAdapter(comm);
    DataSet ds = new DataSet( );
    adap.Fill(ds);

    string[] vendors = new string[Math.Min(count, ds.Tables[0].Rows.Count)];
    for (int i = 0; i < Math.Min(count, ds.Tables[0].Rows.Count); i++)
    {
      vendors[i] = ds.Tables[0].Rows[i].ItemArray[0].ToString( );
    }
    return vendors;
  }
}
```

Теперь посмотрим, как выглядят результаты в броузере. Загрузите страницу и введите несколько символов – не меньше трех (ASP.NET AJAX не будет отправлять запрос веб-службе при меньшем числе символов). Если совпадения будут найдены, они появятся в текстовом поле с некоторой задержкой.

Для веб-службы из примера 12.4 кэширование результатов поиска может принести существенный выигрыш, особенно при многократном использовании одной и той же строки поиска. Чтобы активировать механизм кэширования, достаточно изменить атрибут `WebMethod` метода `GetVendors()`, указав продолжительность хранения значений в кэше:

```
[WebMethod(CacheDuration = 60)]
```

Значение `CacheDuration` измеряется в секундах, поэтому в данном случае веб-служба будет хранить результаты в течение одной минуты.

Если в качестве базы данных вы используете Microsoft SQL Server (как в данном примере), то можно создать `SqlCacheDependency` в объектах `DataSet` (ссылки на дополнительную информацию по этой теме приводятся в разделе «Для дополнительного чтения» в конце этой главы).

Если поиск не дал результатов, попробуйте следующее: в базе данных имеется несколько компаний, названия которых начинаются со слова «International», поэтому после ввода этого слова вы должны получить несколько совпадений. На рис. 12.5 приводятся некоторые типичные результаты.

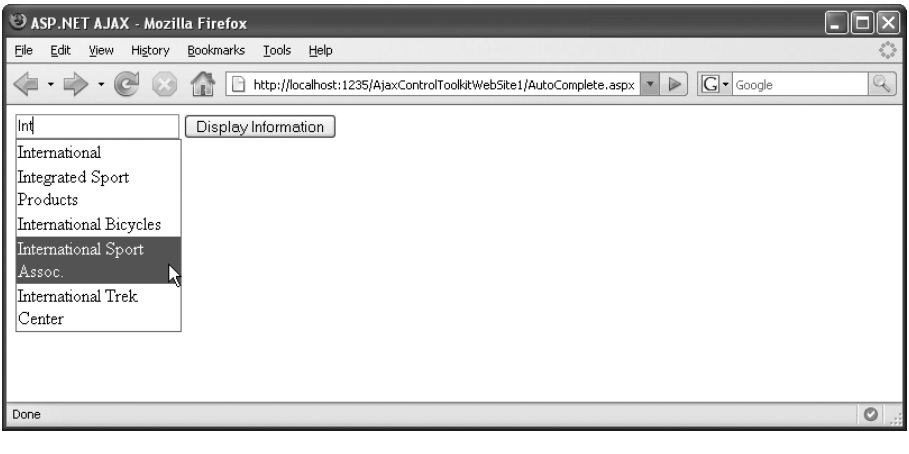


Рис. 12.5. *Платформа ASP.NET AJAX предлагает несколько вариантов названий производителей*

Присоединение календаря к текстовому полю

На любом сайте, который предлагает забронировать номер в гостинице, заказать билеты на самолет, взять машину напрокат, одним словом, везде, где пользователь оформляет бронирование, необходимо предоставить ему возможность ввести дату (дату прибытия и убытия для гостиницы или дату вылета при покупке билета на самолет). Для удобства на большинстве этих веб-сайтов пользователю предоставляется календарь в том или ином виде, в котором он может выбрать требуемую дату.

Реализовать календарь на своем веб-сайте можно с помощью UpdatePanel и элемента управления ASP.NET – Calendar, однако гораздо удобнее использовать компонент CalendarExtender из пакета ASP.NET AJAX Control Toolkit; к тому же он обладает некоторыми дополнительными возможностями, такими как локализация. Как следует из имени компонента CalendarExtender, он является расширителем, поэтому он должен подключаться к некоторому основному элементу управления (обычно к текстовому полю ввода). Взгляните на пример 12.5 и увидите, как немного требуется, чтобы воспроизвести эффект, показанный на рис. 12.6. Когда пользователь щелкает на текстовом поле ввода, появляется календарь. Если пользователь выбирает дату, она автоматически вводится в текстовое поле. Когда поле ввода теряет фокус, календарь исчезает.

Пример 12.5. Использование CalendarExtender

```
Calendar.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      <asp:TextBox ID="TextBox1" runat="server" />
    </div>
    <ajaxToolkit:CalendarExtender ID="CalendarExtender1" runat="server"
      TargetControlID="TextBox1" />
  </form>
</body>
</html>
```

Рис. 12.6. *Календарь появляется, когда пользователь щелкает на текстовом поле ввода*

Динамическая свертка единственной панели

В разделе «Создание раздвижных панелей», ранее в этой главе, демонстрировалась возможность создания серии выдвижных панелей. Теперь мы рассмотрим вариант с единственной панелью. Если панель представлена в компоненте ASP.NET `Panel`, тогда с помощью `CollapsiblePanelExtender` можно реализовать поведение, подобное тому, которое создает расширитель `Accordion` для нескольких панелей. В расширителе `CollapsiblePanelExtender` необходимо определить значения следующих свойств:

`CollapseControlID`

Элемент управления, который будет сворачивать панель

`ExpandControlID`

Элемент управления, который будет разворачивать панель

`TargetControlID`

Элемент управления (панель), который будет сворачиваться и разворачиваться

Как правило, свойствам `CollapseControlID` и `ExpandControlID` присваивается одно и то же значение, а щелчок на этом заголовке вызывает сворачивание или разворачивание панели. В примере 12.6 приводится полный программный код, в котором используется CSS-стиль `cursor: pointer`, который уже встречался вам ранее в примере с компонентом `Accordion`. Снимок с экрана на рис. 12.7 был сделан в момент разворачивания панели. (Разумеется, передать реальный эффект на статическом изображении просто невозможно!)

Пример 12.6. Использование CollapsiblePanelExtender

```
CollapsiblePanel.aspx

<%@ Page Language="C#" %>
```

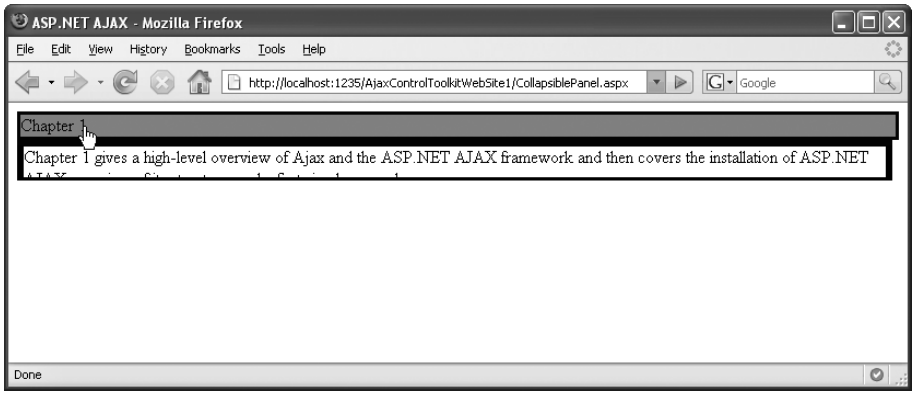


Рис. 12.7. *Панель сворачивается и разворачивается щелчком мыши*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
  <style type="text/css">
    .panelHeader { background-color: blue; border: solid; cursor: pointer; }
    .panelContent { background-color: white; border: solid; }
  </style>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      <asp:Panel ID="Panel1" runat="server" CssClass="panelHeader">Chapter 1
      </asp:Panel>
     <asp:Panel ID="Panel2" runat="server" CssClass="panelContent">Chapter 1
gives a high-level overview of Ajax and the ASP.NET AJAX framework and then
covers the installation of ASP.NET AJAX, a review of its structure, and a
first simple example.</asp:Panel>
    </div>
    <ajaxToolkit:CollapsiblePanelExtender ID="CollapsiblePanelExtender1"
      runat="server" TargetControlID="Panel2"
      CollapseControlID="Panel1" ExpandControlID="Panel1" />
  </form>
</body>
</html>
```

Отображение всплывающего окна на странице

Модальные всплывающие окна были одним из первых эффектов JavaScript, широко принятых на вооружение такими веб-сайтами, как

Google и Amazon, которые стремились повысить степень интерактивности при взаимодействии со своими клиентами. Функция `window.alert()` реализует этот эффект, но внешний вид и содержимое такого всплывающего окна не могут быть изменены. Благодаря каскадным таблицам стилей (CSS) и объектной модели документа (DOM) ныне существует возможность создавать более интересные всплывающие окна без необходимости создавать новые.

Модальное всплывающее окно обычно воспроизводится за счет создания нового элемента DOM (как правило, в виде контейнера `<div>`), затем с помощью JavaScript элемент отображается на экране, а с помощью CSS настраивается его внешний вид. Чтобы всплывающее окно сделать модальным, создаем второй элемент `<div>`, который будет покрывать всю область, отображаемую броузером. Затем с помощью CSS-свойства `z-order` (которое определяет значение виртуальной координаты z элемента и позволяет создавать эффект расположения одного элемента над другим) оба элемента размещаются над текущим содержимым страницы – сначала располагается элемент `<div>`, «охватывающий всю область отображения», а над ним – элемент `<div>` с содержимым всплывающего окна. О размещении компонентов позаботится платформа ASP.NET AJAX, а вам остается реализовать все остальное.

Первоначальное содержимое страницы по-прежнему остается видимым, но поскольку вся видимая область покрыта элементом `<div>`, пользователь не может щелкнуть ни на одной ссылке, расположенной за пределами всплывающего окна, что создает эффект его модальности.

Первый шаг на пути реализации такого окна заключается в том, чтобы создать соответствующие классы CSS. Нам потребуется два таких класса: один для всплывающего окна и второй для элемента `<div>`, охватывающего всю видимую область. Для первого класса мы просто определим параметры рамки, цвет фона и ширину окна (высота определяется механизмом отображения броузера автоматически):

```
.popup {
  border: solid;
  background-color: yellow;
  width: 200px;
}
```

Во втором классе также будет использоваться цвет фона, например серый, но при этом мы еще зададим степень прозрачности, чтобы фон (содержимое страницы) просвечивал сквозь элемент `<div>`. Для создания подобного эффекта в Internet Explorer поддерживается свойство `filter`, во всех остальных броузерах – `opacity`.

```
.content {
  background-color: grey;
  filter: alpha(opacity=50);
  opacity: 0.5;
}
```

Далее нам потребуется серверный элемент управления, который будет отображать модальное всплывающее окно. Для этого подойдет ссылка:

```
<asp:HyperLink ID="HyperLink1"
               runat="server" Text="More ..." NavigateUrl="#" />
```

Теперь создадим само всплывающее окно. Как показано ниже, это будет диалог, предлагающий пользователю зарегистрироваться:

```
<asp:Panel ID="popupPanel" runat="server" CssClass="popup">
  <p>Login to get more information!<br />
    User <input type="text" /> - Password <input type="password" />
    <input type="button" value="Login" id="Button1" runat="server" />
  </p>
</asp:Panel>
```

И наконец, в игру вступает `ModalPopupExtender` из пакета ASP.NET AJAX Control Toolkit. Этот расширитель позволяет создавать модальные всплывающие окна, расположенные поверх текущего содержимого страницы. Здесь придется немного поработать. Вам потребуются следующие свойства элемента `ModalPopupExtender`:

`BackgroundCssClass`

Класс CSS, благодаря которому содержимое страницы будет выглядеть неактивным

`PopupControlID`

Значение атрибута ID элемента, который будет выполнять роль всплывающего окна

`TargetControlID`

Значение атрибута ID элемента, который будет запускать отображение окна

`OKControlID`

Значение атрибута ID элемента, который будет закрывать окно

Полный программный код реализации приводится в примере 12.7. На рис. 12.8 показано всплывающее окно, расположенное поверх страницы. (По умолчанию окно располагается в центре страницы.)

Пример 12.7. Использование ModalPopupExtender

```
ModalPopup.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
  <style type="text/css">
    .content {
```

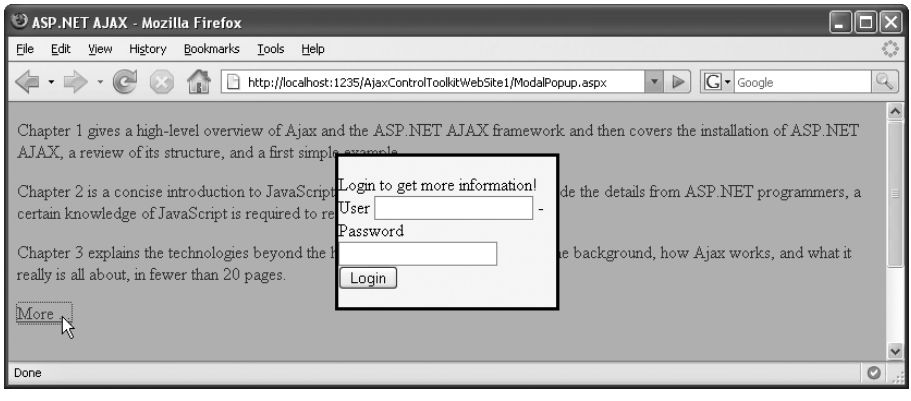


Рис. 12.8. *Всплывающее окно появилось, а содержимое страницы спряталось под серым фоном*

```
        background-color: gray;
        filter: alpha(opacity=50);
        opacity: 0.5;
      }
      .popup {
        border: solid;
        background-color: yellow;
        width: 200px;
      }
    </style>
  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server" />
      <asp:Panel ID="contentPanel" runat="server">
        <p>Chapter 1 gives a high-level overview of Ajax and the ASP.NET AJAX
  framework and then covers the installation of ASP.NET AJAX, a review of its
  structure, and a first simple example.</p>
        <p>Chapter 2 is a concise introduction to JavaScript. Although Atlas
  does its best to hide the details from ASP.NET programmers, a certain
  knowledge of JavaScript is required to really master ASP.NET AJAX.</p>
        <p>Chapter 3 explains the technologies beyond the hype. You learn what
  happens in the background, how Ajax works, and what it really is all about,
  in fewer than 20 pages.</p>
        <p><asp:HyperLink ID="HyperLink1" runat="server" Text="More ..."
  NavigateUrl="#" /></p>
      </asp:Panel>
      <asp:Panel ID="popupPanel" runat="server" CssClass="popup">
        <p>Login to get more information!<br />
          User <input type="text" /> - Password <input type="password" />
          <input type="button" value="Login" id="Button1" runat="server" />
        </p>
      </asp:Panel>
```

```
    <ajaxToolkit:ModalPopupExtender ID="ModalPopupExtender1" runat="server"
      PopupControlID="popupPanel" TargetControlID="HyperLink1"
      OkControlID="Button1" BackgroundCssClass="content" />
  </form>
</body>
</html>
```

Борьба со спамом в блогах и других формах ввода информации

Любой, кто имеет собственный блог или гостевую книгу, знает, сколько неудобств доставляют спамеры, особенно те, кто не участвует в общественной жизни и у кого слишком много свободного времени. Автоматизированные роботы путешествуют по Всемирной паутине и пытаются оставить свои ссылки, рекламные объявления или другую нежелательную информацию везде, где обнаруживают HTML-форму.

Одной из распространенных мер по борьбе со спамом, предпринимаемых во многих блогах, является CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart – полностью автоматизированный открытый тест Тьюринга для различения компьютеров и людей). Если вы попытаетесь отправить URL в Google, создать учетную запись на сайте Yahoo! или купить билет на сайте Ticketmaster, то увидите изображение с несколькими искаженными символами. Идея на удивление проста: только человек способен правильно воспринять текст – алгоритму распознавания это пока не под силу. К сожалению, у такого способа существует один недостаток: большинство реализаций алгоритма CAPTCHA требуют наличие у клиента поддержки графических изображений, что порождает проблему доступности. Кроме того, некоторые реализации CAPTCHA настолько искажают текст, что даже человек, обладающий неплохими визуальными навыками, оказывается не в состоянии распознать его.

Элемент управления `NoBot` из пакета ASP.NET AJAX Control Toolkit пытается обеспечить более слабую, но при этом более доступную защиту форм. Когда `NoBot` добавляется в страницу, он привносит в механизм отправки формы некоторые дополнительные проверки, обеспечивающие повышенные меры безопасности. Например, этот элемент использует механизм управления сеансами ASP.NET, чтобы определить, когда последний раз пользователь выполнял отправку формы. Если пользователь произвел несколько попыток отправки формы за короткий промежуток времени, генерируется сообщение об ошибке.

Это сообщение не отображается броузером автоматически. Вместо этого метод `IsValid()` компонента `NoBot` проверяет форму и возвращает сообщение о состоянии (в виде параметра `out`), если была обнаружена какая-либо проблема. Затем эти данные могут использоваться сервером для подготовки соответствующего сообщения об ошибке. На рис. 12.9 показано такое сообщение, созданное программным кодом из примера 12.8. Что-

бы получить такое сообщение, форма была отправлена дважды в течение двух секунд, что меньше значения, указанного в свойстве ResponseMinimumDelaySeconds компонента NoBot (3 секунды – для данного примера).

Пример 12.8. Использование элемента управления NoBot

```
AntiSpam.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

  protected void ProcessData(object sender, EventArgs e)
  {
    NoBotState state;
    if (!NoBot1.IsValid(out state))
    {
      Label1.Text = "Entry refused ("
        + HttpUtility.HtmlEncode(state.ToString( )) + ")!";
    }
    else
    {
      Label1.Text = "Entry accepted!";
    }
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
```

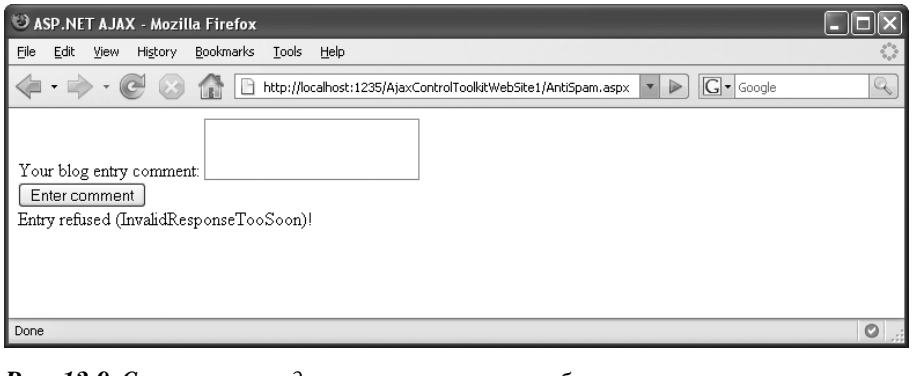


Рис. 12.9. *Сервер отверг данные, поскольку они были отправлены несколько раз за слишком короткий промежуток времени*

```
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      Your blog entry comment:
      <asp:TextBox ID="TextBox1" TextMode="MultiLine" runat="server" />
      <br />
      <asp:Button ID="Button1" runat="server"
        Text="Enter comment" OnClick="ProcessData" />
      <br />
      <asp:Label ID="Label1" runat="server" />
    </div>
    <ajaxToolkit:NoBot ID="NoBot1" runat="server"
      CutoffMaximumInstances="5" CutoffWindowSeconds="30"
      ResponseMinimumDelaySeconds="3" />
  </form>
</body>
</html>
```

Защита, предоставляемая компонентом NoBot, никоим образом не может считаться достаточной для сайтов, испытывающих серьезный натиск со стороны спамеров. Однако она может помочь предотвратить менее изощренные атаки. Она также позволит избежать многих проблем с удобством использования и доступностью, свойственных алгоритмам CAPTCHA.

Создание интерфейса с вкладками

Большинство настольных приложений и многие современные броузеры реализуют интерфейс с вкладками. Для создания вкладок на веб-сайте можно использовать элементы управления TabPanel и TabContainer, входящие в состав пакета ASP.NET AJAX Control Toolkit.

Каждый компонент TabPanel представляет одну вкладку. Для отображения заголовка на вкладке используется свойство HeaderText. Содержимое самой вкладки, включая необязательную информацию о стилях оформления, хранится в дочернем элементе <ContentTemplate> компонента TabPanel:

```
<ajaxToolkit:TabPanel ID="TabPanel1" runat="server" HeaderText="Chapter 1">
  <ContentTemplate>
    Chapter 1 gives a high-level overview of Ajax and the ASP.NET AJAX
framework and then covers the installation of ASP.NET AJAX, a review of its
structure, and a first simple example.
  </ContentTemplate>
</ajaxToolkit:TabPanel>
```

Элемент TabContainer служит контейнером для размещения компонентов TabPanel. Все остальное берет на себя пакет ASP.NET AJAX Control Toolkit. Когда пользователь щелкает на вкладке, ее содержимое автоматически выводится на экран (пример 12.9).

Пример 12.9. Использование элементов управления TabContainer и TabPanel

```
Tab.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      <ajaxToolkit:TabContainer ID="TabContainer1" runat="server">
        <ajaxToolkit:TabPanel ID="TabPanel1" runat="server"
                              HeaderText="Chapter 1">
          <ContentTemplate>
           Chapter 1 gives a high-level overview of Ajax and the ASP.NET AJAX
framework and then covers the installation of ASP.NET AJAX, a review of its
structure, and a first simple example.
          </ContentTemplate>
        </ajaxToolkit:TabPanel>
        <ajaxToolkit:TabPanel ID="TabPanel2" runat="server"
                              HeaderText="Chapter 2">
          <ContentTemplate>
            Chapter 2 is a concise introduction to JavaScript. Although
ASP.NET AJAX does its best to hide the details from ASP.NET programmers, a
certain knowledge of JavaScript is required to really master ASP.NET AJAX.
          </ContentTemplate>
        </ajaxToolkit:TabPanel>
        <ajaxToolkit:TabPanel ID="TabPanel3" runat="server"
                              HeaderText="Chapter 3">
          <ContentTemplate>
            Chapter 3 explains the technologies beyond the hype. You learn
what happens in the background, how Ajax works, and what it really is all
about, in fewer than 20 pages.
          </ContentTemplate>
        </ajaxToolkit:TabPanel>
      </ajaxToolkit:TabContainer>
    </div>
  </form>
</body>
</html>
```

На рис. 12.10 показан интерфейс с вкладками, содержащий три вкладки. Однако здесь есть один подвох, по крайней мере, в настоящее время. Если в броузере будет отключена поддержка JavaScript, результат работы примера 12.9 будет выглядеть так, как показано на рис. 12.11. Причина заключается в том, что изначально содержимое

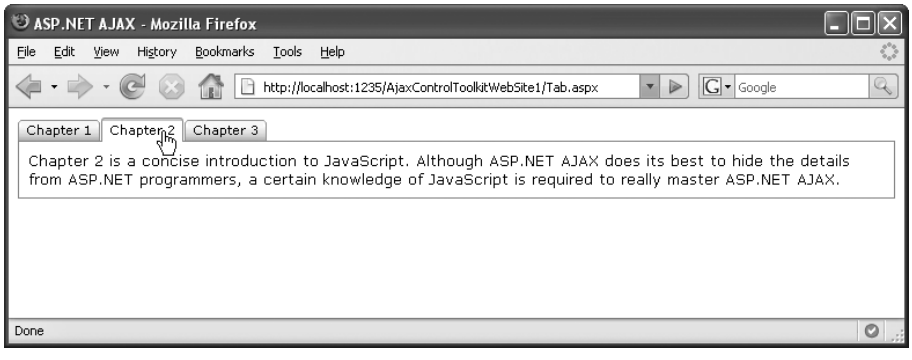


Рис. 12.10. *Щелчок на вкладке приводит к отображению ее содержимого*

Рис. 12.11. *При отсутствии поддержки JavaScript вкладки не отображаются*

вкладок скрыто, а видимым его делает программный код JavaScript. Если отсутствует поддержка JavaScript, то не будет и никаких вкладок. Но кто знает, может быть, в одном из ближайших выпусков ASP.NET AJAX Control Toolkit будет реализован лучший способ, не зависящий от наличия поддержки JavaScript. Не бойтесь экспериментировать с элементами управления!

В главе 17 будет продемонстрирован альтернативный способ реализации вкладок, основанный на использовании пакета ASP.NET AJAX Futures.

Подведение итогов

В этой главе говорилось о различных элементах управления из пакета ASP.NET AJAX Control Toolkit. В состав этого пакета входит еще много других компонентов и с каждой новой версией в нем появляются новые компоненты и новые возможности, поэтому старайтесь чаще посещать домашнюю страницу проекта Control Toolkit!

Для дополнительного чтения

http://ajax.asp.net/ajaxtoolkit/

Тестовая страница, где можно опробовать компоненты пакета Control Toolkit

http://www.google.com/webhp?complete=1&hl=en

Веб-приложение Google Suggest

13

Создание собственных элементов управления и помощь сообществу

Как вы уже видели в главах 10, 11 и 12, пакет ASP.NET AJAX Control Toolkit предлагает множество самых разнообразных элементов управления. В этой главе мы продвинемся еще на один шаг и посмотрим, как можно использовать инфраструктуру пакета для создания своих собственных элементов управления. Кроме того, здесь вы узнаете, как можно оказать содействие сообществу Control Toolkit.

Создание собственных элементов управления ASP.NET AJAX

Спектр элементов управления, предлагаемых пакетом ASP.NET AJAX Control Toolkit, продолжает расширяться от версии к версии. Более того, помимо этого впечатляющего списка он содержит еще одну инфраструктуру, которая предназначена для создания своих собственных элементов управления. Если вы постоянно используете одни и те же эффекты, реализованные на JavaScript, вполне оправданным будет сделать их доступными через ASP.NET AJAX.

В этом разделе будет создан расширитель, который ограничивает возможность ввода в текстовое поле предопределенным набором символов – данная функциональная возможность не поддерживается средствами HTML. В пакете ASP.NET AJAX Control Toolkit имеется шаблон проекта для Visual Studio 2005, облегчающий эту работу. Сначала вам нужно будет установить шаблон, затем изменить его и добавить логику работы нового расширителя.

Как вы уже знаете, пакет ASP.NET AJAX Control Toolkit поставляется в виде одного файла DLL, который содержит полный комплект элементов управления. Для создания собственного компонента вам потре-

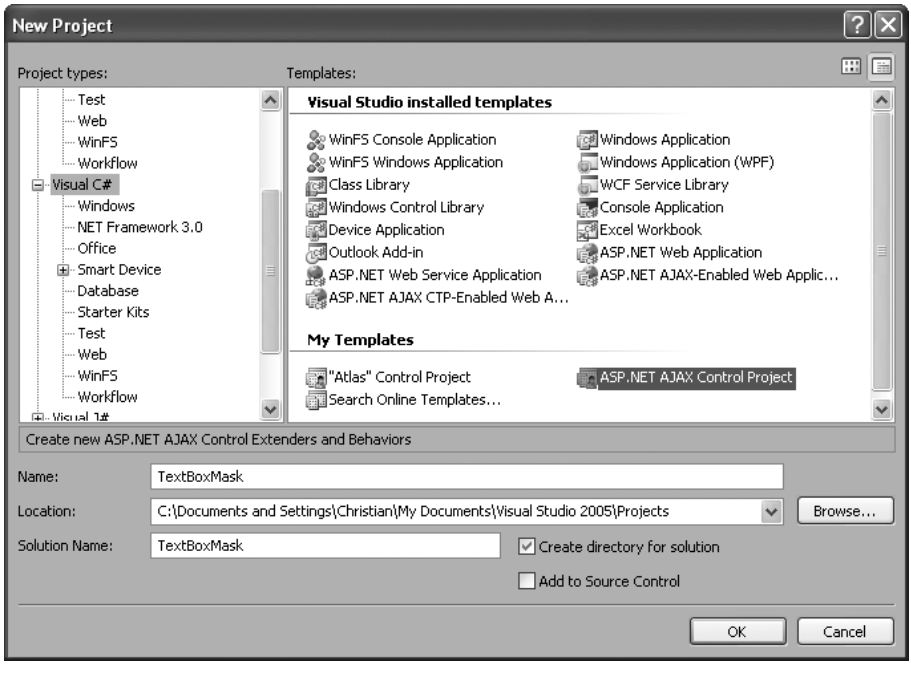


Рис. 13.1. *Шаблон ASP.NET AJAX Control*

буется скомпилировать программный код. К счастью, для упрощения процесса создания расширителей, таких как тот, что мы собираемся написать, в состав пакета входит шаблон для Visual Studio.

В главе 10 вы познакомились с мастером установки VSI, который кроме всего прочего создает шаблоны для разработки веб-сайтов, управляемых компонентами из ASP.NET AJAX Control Toolkit. Это «все прочее» в основном является шаблонами на C# и VB, позволяющими создавать свои собственные элементы управления. Когда вы запустите среду разработки Visual Studio и приступите к созданию нового проекта, обратите внимание на шаблон ASP.NET AJAX Control Project (рис. 13.1).

Если вы используете Visual Web Developer Express Edition, вы сможете установить VSI, но не сможете создать новый проект расширителя. Версия Express Edition позволяет создавать только веб-проекты, но не проекты собственных элементов управления. Однако шаблоны проектов можно использовать в Microsoft Visual Basic 2005 Express Edition и Microsoft Visual C# 2005 Express Edition. Эти продукты, так же как и Visual Web Developer Express Edition, распространяются бесплатно. Если у вас еще нет этих продуктов, посетите веб-сайт Microsoft Visual Express Edition (*http://msdn.microsoft.com/vstudio/express*), загрузите и установите любой из них. После этого вы сможете создавать проекты, которые компилируются в сборки .NET (файлы *.dll*).

Совершенно очевидно, что самый удобный способ заключается в использовании Visual Studio 2005. Если в вашем распоряжении имеется Visual Studio 2005, то вы сможете создать отдельное решение, которое будет содержать как проект вашего собственного расширителя, так и проект веб-сайта, использующего вновь созданный расширитель.

В следующем примере мы будем использовать Visual Studio 2005 и C#. Следует отметить, что пример также будет работать в версиях Express Edition среды Visual Web Developer, Visual C# и Visual Basic. Однако при использовании Visual C# Express Edition или Visual Basic Express Edition в процессе разработки вам придется выполнять некоторые дополнительные действия: при каждом изменении исходных текстов расширителя вам будет нужно перекомпилировать его в Visual C# Express Edition или Visual Basic Express Edition и обновить ссылки.

Предполагается, что создаваемый расширитель должен расширять возможности существующего серверного элемента управления ASP.NET. Поэтому в Visual Studio откройте веб-сайт, где вы сможете одновременно работать со своим расширителем и с существующим элементом управления ASP.NET. Загрузите веб-сайт ASP.NET AJAX в Visual Studio. В меню File (файл) выберите пункт Add (добавить) и затем пункт New Project (новый проект). Выберите шаблон ASP.NET AJAX Control Project, как показано на рис. 13.1. Используйте в примере имя проекта `TextBoxMask`.

Новый шаблон создает проект по умолчанию, используя имя проекта (расширитель `TextBoxMask`). Изначально он состоит из четырех файлов:

TextBoxMaskBehavior.js

Программный код JavaScript, составляющий основу расширителя

TextBoxMaskDesigner.cs

Программный код, используемый конструктором Visual Studio

TextBoxMaskExtender.cs

Программный код C#, который работает с инспектором свойств Visual Studio в режиме проектирования и позволяет с его помощью изменять свойства элемента управления

Основная работа будет вестись внутри самой важной части расширителя – в файле *TextBoxMaskBehavior.js*, где будет сосредоточена вся JavaScript-логика работы на стороне клиента.

Но сначала необходимо разобраться с двумя другими файлами примера. Откройте файл *TextBoxMaskDesigner.cs* и обратите внимание: содержимое этого файла – не что иное, как заготовка класса. Она выглядит, как показано в примере 13.1.

Пример 13.1. Класс TextBoxMaskDesigner

```
TextBoxMaskDesigner.cs

using System.Web.UI.WebControls;
```

```
using System.Web.UI;

namespace TextBoxMask
{
  class TextBoxMaskDesigner : AjaxControlToolkit.Design.
ExtenderControlBaseDesigner<TextBoxMaskExtender>
  {

  }
}
```

Файл *TextBoxMaskExtender.cs* содержит информацию о расширителе для конструктора. Как видно из примера 13.2, в программном коде имеется ссылка на файл *TextBoxMaskBehavior.js*. По умолчанию для элементов, используемых совместно с этим расширителем, выбран тип данных `Control`. Однако мы предполагаем расширять текстовое поле, поэтому измените тип `Control` на `TextBox`.

```
[TargetControlType(typeof(TextBox))]
```

Далее в файле вы можете увидеть единственное свойство, присутствующее в шаблоне: `MyProperty`. Удалите этот член класса и создайте строковое свойство `ValidChars` с соответствующими методами доступа. Позднее в этом свойстве будут храниться символы, допустимые для ввода в текстовое поле.

В качестве методов доступа к значению свойства будут использоваться функции `GetPropertyValue()` и `SetPropertyValue()`. Кроме того, установите атрибут `DefaultProperty`, чтобы сделать `ValidChars` свойством расширителя по умолчанию. Полный программный код представлен в примере 13.2.

Пример 13.2. Класс TextBoxMaskExtender

```
TextBoxMaskProperties.cs

using System;
using System.Web.UI.WebControls;
using System.Web.UI;
using System.ComponentModel;
using System.ComponentModel.Design;
using AjaxControlToolkit;

[assembly: System.Web.UI.WebResource("TextBoxMask.TextBoxMaskBehavior.js",
"text/javascript")]

namespace TextBoxMask
{
  [Designer(typeof(TextBoxMaskDesigner))]
  [ClientScriptResource("TextBoxMask.TextBoxMaskBehavior",
"TextBoxMask.TextBoxMaskBehavior.js")]
  [TargetControlType(typeof(TextBox))]
  [DefaultProperty("ValidChars")]
  public class TextBoxMaskExtender : ExtenderControlBase
  {
```

```
        // TODO: Добавьте сюда свои методы доступа к свойствам.
        //
        [ExtenderControlProperty]
        [DefaultValue("")]
        public string ValidChars
        {
          get
          {
            return GetPropertyValue("ValidChars", "");
          }
          set
          {
            SetPropertyValue("ValidChars", value);
          }
        }
      }
    }
```

Единственное свойство, доступное по умолчанию и не требующее регистрации, – это TargetControlID, ссылающееся на элемент управления, с которым будет связан расширитель.

Наконец, нам необходимо создать программный код JavaScript, который будет расширять функциональные возможности текстовых полей ввода, с которым связан элемент управления. Этот код будет находиться в файле *TextBoxMaskBehavior.js*.

Файл шаблона *.js* содержит несколько полезных комментариев, описывающих действия, которые следует выполнить в соответствующих местах программы. В первую очередь необходимо определить переменные для каждого свойства расширителя. В соответствии с синтаксическими соглашениями имена переменных должны начинаться с символа подчеркивания (_), за которым следует символ в нижнем регистре:

```
this._validChars = "";
```

После этого необходимо написать методы доступа ко всем свойствам, которые должны быть доступны из программного кода JavaScript. Это довольно простая задача, которую практически полностью можно выполнить с помощью операций копирования и вставки. Следует учитывать, что язык JavaScript чувствителен к регистру символов, поэтому вы должны быть последовательны при выборе регистра как для имен переменных JavaScript, так и для свойств C#.

```
get_ValidChars : function( ) {
  return this._validChars;
},

set_ValidChars : function(value) {
  this._validChars = value;
}
```

Последний этап заключается в создании программного кода инициализации расширителя. Это то место, где код JavaScript присоединяет-

ся к рассматриваемому элементу управления. Мы создаем пример расширителя, который проверяет символы, вводимые пользователем в текстовом поле. Работать он будет следующим образом: расширитель определяет свойство, которое позволяет разработчику указать строку, содержащую допустимые для ввода символы. Например, если свойству будет присвоена строка "0123456789", текстовое поле сможет принимать только цифровые символы.

В нашем примере мы хотим, чтобы функция проверки вызывалась по нажатию клавиши. Если пользователь нажимает клавишу, не соответствующую ни одному допустимому символу, событие должно быть отменено, чтобы этот символ не появился в текстовом поле.

Обработчик события должен быть помещен в метод `initialize()` класса `TextBoxMaskBehavior` (шаблоны класса и метода уже были созданы).

Возможно, вам понадобится добавить свой код в метод `dispose()`. Этот метод вызывается в процессе завершения работы. Как правило, в методе `dispose()` производится освобождение ресурсов, использовавшихся элементом управления, включая отсоединение всех обработчиков событий.

Мы предоставим платформе ASP.NET AJAX выполнение проверки возможностей броузера по определению и обработке нажатий клавиш пользователем. Большую часть этой работы возьмет на себя метод `$addHandler()`. Сначала нужно вставить его в метод `initialize()`. Порядок подключения метода к событию выглядит следующим образом:

```
this._keydownHandler = Function.createDelegate(this, this._onkeydown);
$addHandler(this.get_element( ),"keydown", this._keydownHandler);
```

Используя метод `dispose()`, вы можете этот обработчик удалить, на этот раз с помощью метода `removeHandler()`, как показано ниже:

```
$removeHandler(this.get_element( ), "keydown", this._keydownHandler);
this._keydownHandler = null;
```

Наконец, необходимо написать программный код JavaScript расширителя. Сначала, проанализировав тип броузера, этот код должен определить, какая клавиша была нажата. Затем код должен отыскать введенный символ в списке допустимых символов. Если искомый символ отсутствует в этом списке, метод завершается инструкцией `return false`, что приводит к отмене события и предотвращает появление символа в текстовом поле. Если искомый символ обнаружен в списке, метод завершается инструкцией `return true` и событие передается дальше.

Кроме того, метод возвращает значение `true` для символов с кодами 8, 9, 16, 35, 36, 37, 38, 39, 40, 45 и 46 – это коды клавиш забоя (Backspace), табуляции, Shift, Home, End, четырех клавиш со стрелками, Insert и Delete. Другой специальный случай – клавиши на дополнительной цифровой клавиатуре (с кодами от 96

до 105) – JavaScript-метод `String.fromCharCode()` не преобразует их в связанные с ними цифровые символы. Поэтому коды клавиш между 96 и 105 будут преобразованы в коды соответствующих цифровых клавиш на основной клавиатуре.

В Internet Explorer возврат значения `false` из обработчика события не может использоваться для предотвращения появления введенного символа в текстовом поле. Для этого в обработчике события нажатия клавиши необходимо вызвать метод `preventDefault()`.

```
this._onkeydown : function(e) {
  var key = e.rawEvent.keyCode;
  if (key >= 96 && key <= 105) {
    key -= 48;
  }
  if (key == 8 || key == 9 || key == 16
     || (key >= 35 && key <= 40) || key == 45 || key == 46
     || _validChars.indexOf(String.fromCharCode(key)) != -1) {
    return true;
  } else {
    e.preventDefault( );
    return false;
  };
}
```

И это все, что имеет отношение к JavaScript. Полный программный код расширителя приводится в примере 13.3. Все комментарии, имевшиеся в шаблоне элемента управления, остались нетронутыми; к ним можно обратиться как к справочной информации.

Пример 13.3. Реализация расширителя на JavaScript

```
TextBoxMaskBehavior.js

// README
//
// Добавление свойства выполняется в два этапа:
//
// 1. Создать переменную-член, в которой будет храниться значение свойства.
// 2. Добавить методы доступа к свойству get_ и set_.
//
// Не забывайте, что имена переменных и методов
// чувствительны к регистру символов!
//

Type.registerNamespace('TextBoxMask');

TextBoxMask.TextBoxMaskBehavior = function(element) {

  TextBoxMask.TextBoxMaskBehavior.initializeBase(this, [element]);
  // TODO : (Этап 1) Добавьте сюда переменные ваших свойств
  //
  this._validChars = null;

}
```

```
TextBoxMask.TextBoxMaskBehavior.prototype = {

  initialize : function( ) {
    TextBoxMask.TextBoxMaskBehavior.callBaseMethod(this, 'initialize');

    // TODO: добавьте сюда код инициализации
    this._keydownHandler = Function.createDelegate(this, this._onkeydown);
    $addHandler(this.get_element( ), "keydown", this._keydownHandler);
  },

  dispose : function( ) {
    // TODO: добавьте сюда код, освобождающий ресурсы
    $removeHandler(this.get_element( ), "keydown", this._keydownHandler);
    this._keydownHandler = null;

    TextBoxMask.TextBoxMaskBehavior.callBaseMethod(this, 'dispose');
  },

  _onkeydown : function(e) {
    var key = e.rawEvent.keyCode;
    if (key >= 96 && key <= 105) {
      key -= 48;
    }
    if (key == 8 || key == 9 || key == 16
       || (key >= 35 && key <= 40) || key == 45 || key == 46
       || this._validChars.indexOf(String.fromCharCode(key)) != -1) {
      return true;
    } else {
      e.preventDefault( );
      return false;
    }
  },

  // TODO: (Этап 2) Добавьте здесь методы доступа к свойствам
  //
  get_ValidChars : function( ) {
    return this._validChars;
  },

  set_ValidChars : function(value) {
    this._validChars = value;
  }

}

TextBoxMask.TextBoxMaskBehavior.registerClass(
  'TextBoxMask.TextBoxMaskBehavior',
  AjaxControlToolkit.BehaviorBase);
```

Теперь давайте соберем проект. В результате будет создан файл *TextBoxMask.dll*, куда будет включен файл *.js* в виде внедренного ресурса, доступного для элемента управления `ScriptManager` на этапе исполнения. Обычно расширитель `TextBoxMask` появляется в палитре компонентов автоматически. Но добавлять его в проект веб-сайта придется

вручную. Для этого в окне Solution Explorer (обозреватель решений) щелкните правой кнопкой мыши на веб-сайте ASP.NET AJAX и выберите пункт контекстного меню Add Reference (добавить ссылку). Во вкладке Projects (проекты) загрузите сборку *TextBoxMask.dll*. После этого сборка автоматически будет скопирована в каталог *Bin*.

Если вы используете Visual Web Developer Express Edition, то не сможете так просто добавить ссылку на проект своего элемента управления. Вместо этого вам придется добавить ссылку на сборку *TextBoxMask.dll*. В окне Solution Explorer (обозреватель решений) щелкните правой кнопкой мыши на имени веб-сайта и выберите пункт контекстного меню Add Reference (добавить ссылку). В диалоге Add Reference щелкните на кнопке Browse (обзор) и перейдите в каталог сборки проекта вашего элемента управления. Обычно путь к каталогу вывода проекта выглядит, как показано ниже:

C:\Documents and Settings\<name>\My Documents\Visual Studio 2005\Projects\TextBoxMask\TextBoxMask\bin\Release

Выберите файл *TextBoxMask.dll* и щелкните на кнопке OK. (Если Visual Web Developer попросит подтвердить, следует ли перезаписать существующий файл *.dll*, щелкните на кнопке No (нет).)

Ссылка на файл *.dll* добавлена в веб-проект. Всякий раз, когда вы будете перекомпилировать свой элемент управления в Visual C# или Visual Basic, вам необходимо будет обновлять ссылку в Visual Web Developer. Для этого в окне Solution Explorer (обозреватель решений) откройте папку *Bin*, правой кнопкой мыши щелкните на *TextBoxMask.dll* и выберите пункт контекстного меню Update Reference (обновить ссылку). Если у вас открыта страница, где используется этот элемент управления, вам может потребоваться закрыть и затем опять открыть ее.

Если вы используете Visual Studio 2005, повторная сборка модуля расширителя на C# вызовет автоматическое обновление ссылки в проекте веб-сайта.

В проекте веб-сайта создайте новую страницу ASP.NET. Зарегистрируйте префикс тега расширителя в самом начале страницы, добавив следующую разметку:

```
<%@ Register Assembly="TextBoxMask" Namespace="TextBoxMask" TagPrefix="cc1"%>
```

Затем добавьте в страницу элемент управления `TextBoxMask` – не забудьте добавить элемент управления `ScriptManager`. Добавьте текстовое поле, а затем свяжите расширитель с текстовым полем установкой свойства расширителя – `TargetControlID`. Программный код, приведенный в примере 13.4, создает текстовое поле, которое допускает ввод исключительно цифровых символов, перечисленных в свойстве `ValidChars`. Реализовать такое поведение только силами JavaScript было бы несколько сложнее, поэтому расширитель `TextBoxMask` действительно по-

может вам сэкономить время и силы. И как дополнительное преимущество – получившийся расширитель может использоваться многократно.

Пример 13.4. Использование собственного расширителя

```
TextBoxMask.aspx

<%@ Page Language="C#" %>
<%@ Register Assembly="TextBoxMask" Namespace="TextBoxMask" TagPrefix="cc1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <cc1:TextBoxMaskExtender ID="TextBoxMaskExtender1" runat="server"
      TargetControlID="TextBox1" ValidChars="1234567890" />
    <div>
     <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
  </form>
</body>
</html>
```

На рис. 13.2 показано, как выглядит страница в окне броузера[1]. И хотя вы не можете видеть, что происходит при нажатии нецифровых клавиш (ничего не происходит), снимок с экрана поможет вам понять,

Рис. 13.2. *В текстовое поле теперь можно ввести только цифры*

[1] Автор привел иллюстрацию аналогичного примера из книги по Atlas, прежней версии ASP.NET AJAX. Рассмотренный здесь пример 13.4 покажет строку заголовка «ASP.NET AJAX», а в адресной строке в броузере ожидается обращение к порту 1234 отладочного сервера: *http://localhost:1234/AJAX... – Примеч. науч. ред.*

как может использоваться этот расширитель для ограничения ввода на странице.

Дополнительная возможность, которую можно было бы добавить в этот расширитель (который реализует методику ограничения на базе белого списка), – это подключить механизм реализации черного списка, когда допустимыми считаются все символы, за исключением тех, что были явно указаны. Так же можно было бы реализовать расширитель, позволяющий определять маску символов. Тогда расширитель мог бы проверять пользовательский ввод на соответствие заданной маске.

Содействие проекту Control Toolkit

До сих пор мы говорили о создании компонентов с использованием инфраструктуры ASP.NET AJAX Control Toolkit. Теперь у вас могло бы появиться желание передать новый расширитель для включения в состав проекта Control Toolkit. После этого команда разработчиков рассмотрела бы его и если бы сочла его достаточно полезным, могла бы со временем принять его в состав пакета для использования всем сообществом. Однако существует более простой способ оказать содействие сообществу. Например, вы могли бы писать исправления к пакету: фрагменты кода, исправляющие ошибки или добавляющие новые функциональные возможности. Группа разработчиков пакета предоставляет удобные программные средства, которые позволяют загружать самые свежие версии и отправлять свои исправления. В данном разделе мы рассмотрим именно этот процесс, то есть постараемся удовлетворить существующий (на момент написания книги) запрос на добавление новой функциональной возможности (мы будем фокусировать наше внимание на самом процессе, а не на функциональной возможности или программном коде).

В первую очередь необходимо получить утилиту для работы с исправлениями, которая доступна по адресу: *http://ajax.asp.net/ajaxtoolkit/patchtool/*. Обязательно установите .NET Framework 2.0 и Visual J# .Net Redistributable Package 2.0 (на странице имеются ссылки на оба пакета). На рис. 13.3 показана страница утилиты для работы с исправлениями.

Для запуска процесса установки щелкните на кнопке Install (установить). На экране появится диалог, требующий подтвердить ваше намерение выполнить установку (рис. 13.4). После подтверждения приложение и необходимые инструменты будут загружены и установлены (рис. 13.5).

Мастер установки добавит утилиту в меню Start (Пуск) и запустит приложение. В начальном диалоге вам будет предложен выбор из двух вариантов (рис. 13.6):

Create a Patch (создать исправление)

Поможет загрузить пакет ASP.NET AJAX Control Toolkit, чтобы приступить к работе над исправлениями

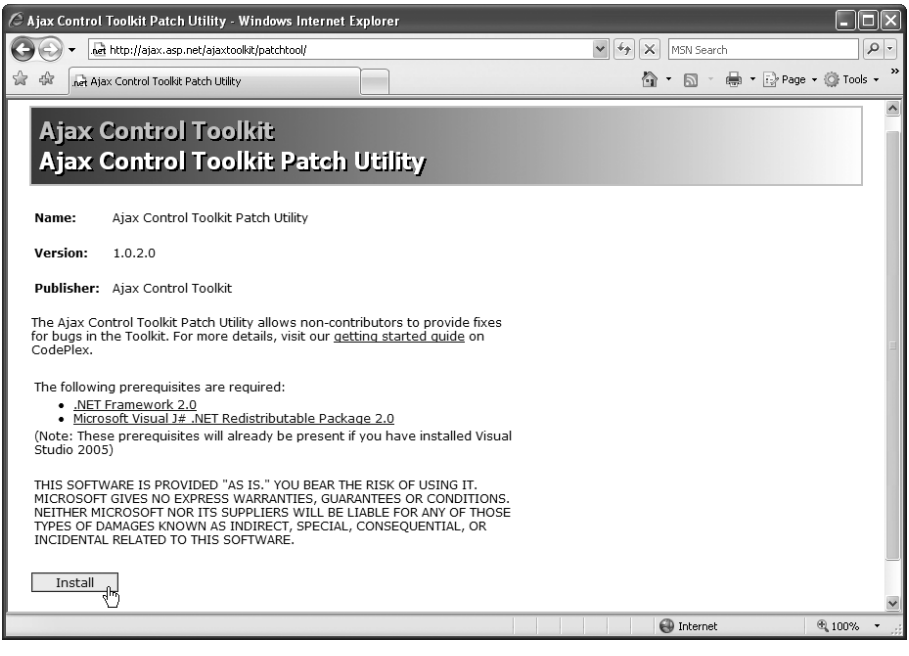


Рис. 13.3. *Страница AJAX Control Toolkit Patch Utility*

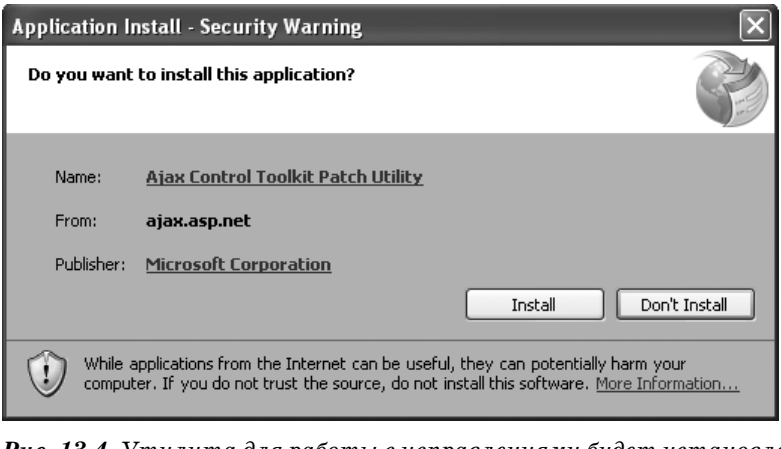


Рис. 13.4. *Утилита для работы с исправлениями будет установлена*

Prepare Patch for Submission (подготовка исправлений к передаче)

Поможет упаковать и подготовить исправления к выгрузке на страницу проекта пакета на сайт CodePlex

Выберите вариант Create a Patch (создать исправление). Спустя короткий промежуток времени будет получен список всех доступных версий ASP.NET AJAX Control Toolkit (рис. 13.7). Вы можете загрузить

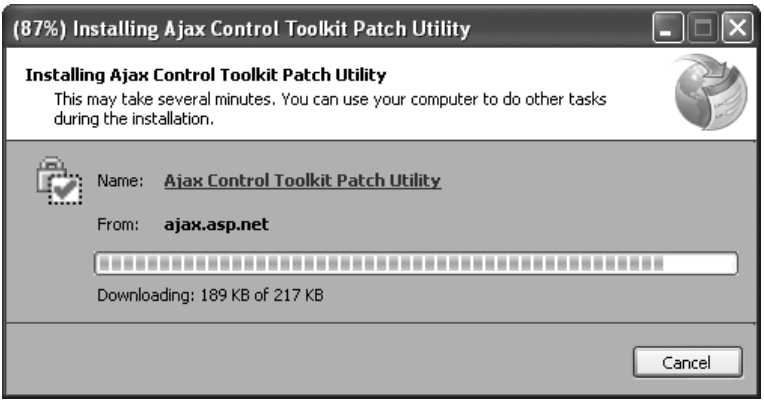


Рис. 13.5. *Выполняется загрузка утилиты*

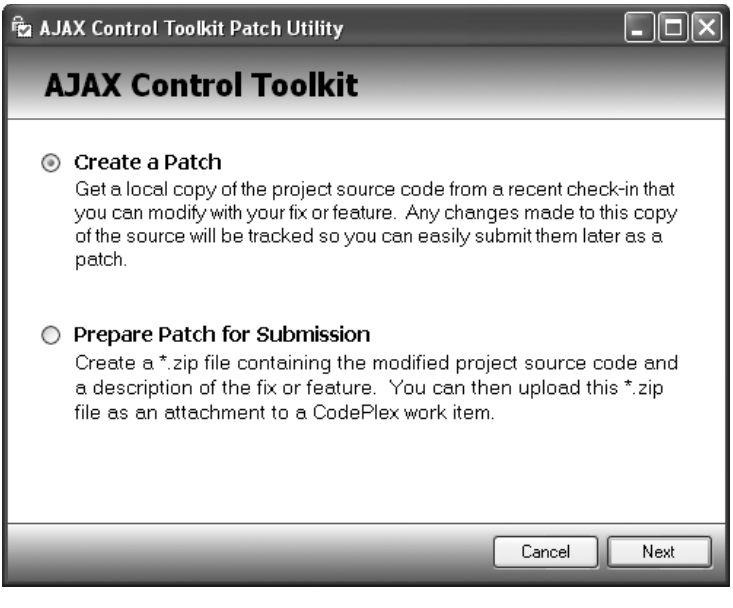


Рис. 13.6. *Начальный диалог утилиты для работы с исправлениями*

и использовать самую последнюю официальную версию пакета, но наилучшим выбором будет использовать самый свежий программный код. В противном случае ваши изменения и дополнения в некотором файле могут конфликтовать с остальным содержимым того же файла. Первый пункт предложенного списка всегда соответствует наиболее свежей версии.

После того как будут приняты лицензионные соглашения (рис. 13.8), будет выполнена загрузка выбранной версии. После этого утилита для работы с исправлениями попросит указать каталог, куда можно будет

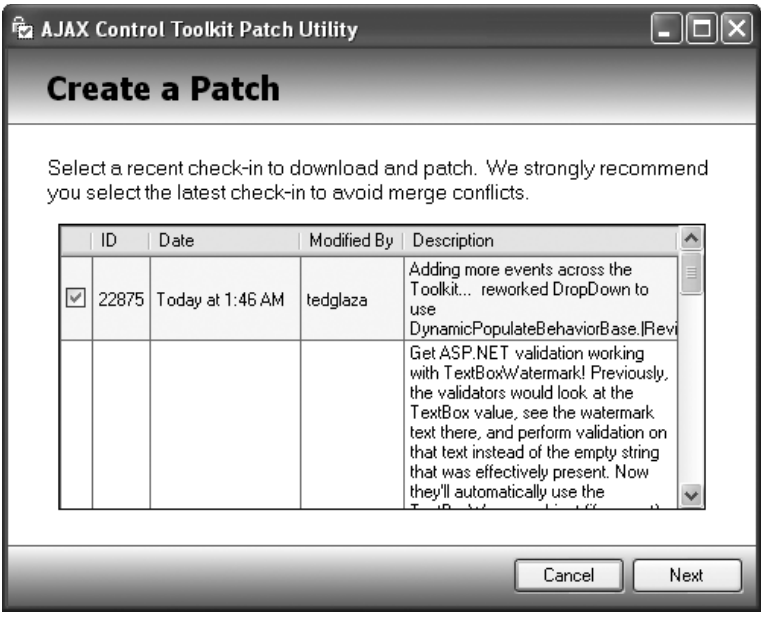


Рис. 13.7. *Выбор версии пакета – рекомендуется выбирать самую свежую версию*

распаковать загруженный код. После щелчка на ссылке «Choose folders to include» (выбор каталогов для подключения) вам будет представлен список ветвей проекта, как показано на рис. 13.9. Выбирайте только Development Branch (ветвь для разработчиков) – другие ветви проекта Release Branch (официально выпущенная версия) и Orcas[1] Branch будут недоступны для внесения исправлений. Даже не все члены проекта имеют право вносить изменения в код из этих двух ветвей проекта.

После распаковки проекта утилита автоматически откроет проект в Visual Studio (здесь предполагается, что остался включенным флажок «Open the project in Visual Studio upon completion» (открыть проект в Visual Studio по завершении)). После этого Visual Studio может один или два раза вывести предупреждение системы безопасности (как показано на рис. 13.10) – выбирайте вариант «Load project normally» (загрузить проект обычным образом), чтобы получить полный доступ к проекту. Следующий и, пожалуй, самый сложный шаг – это создание самого исправления. Откройте исходные тексты пакета, добавьте новую функциональную возможность или исправьте ошибку, сохраните изменения, пересоберите проект и проверьте работу кода.

---

[1] Orcas – кодовое название очередного релиза Visual Studio. Пакет ориентирован, в частности, на разработчиков приложений для Windows Vista, Microsoft Office 2007. – *Примеч. науч. ред.*

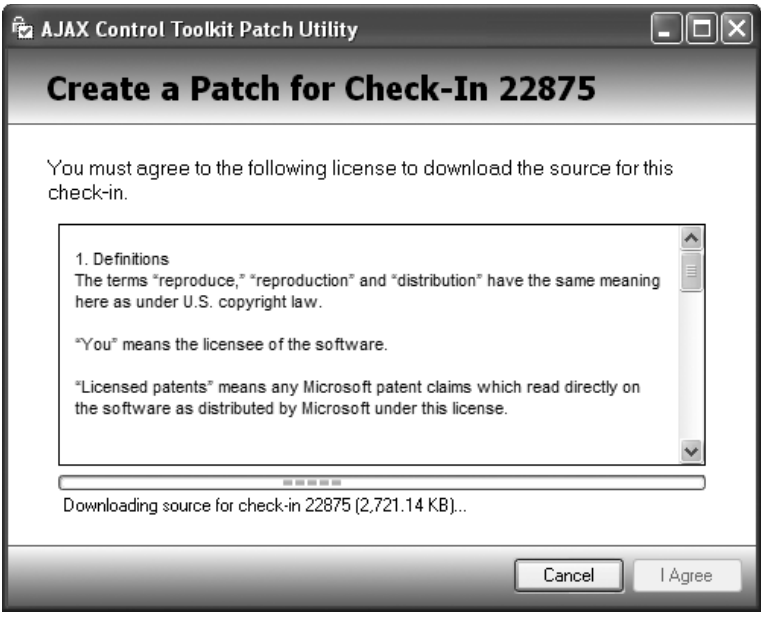


Рис. 13.8. *Прочитайте лицензионное соглашение, и после этого исходные тексты будут загружены*

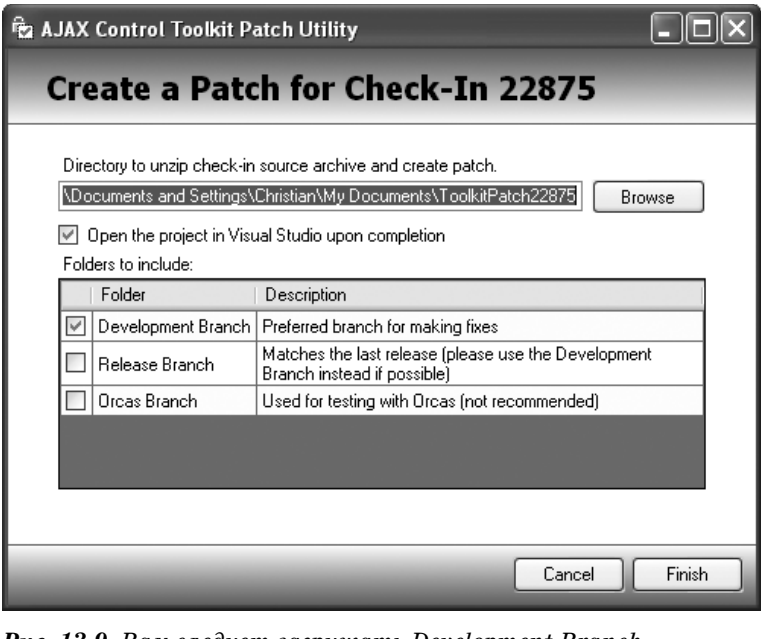


Рис. 13.9. *Вам следует загружать Development Branch (ветвь для разработчиков)*

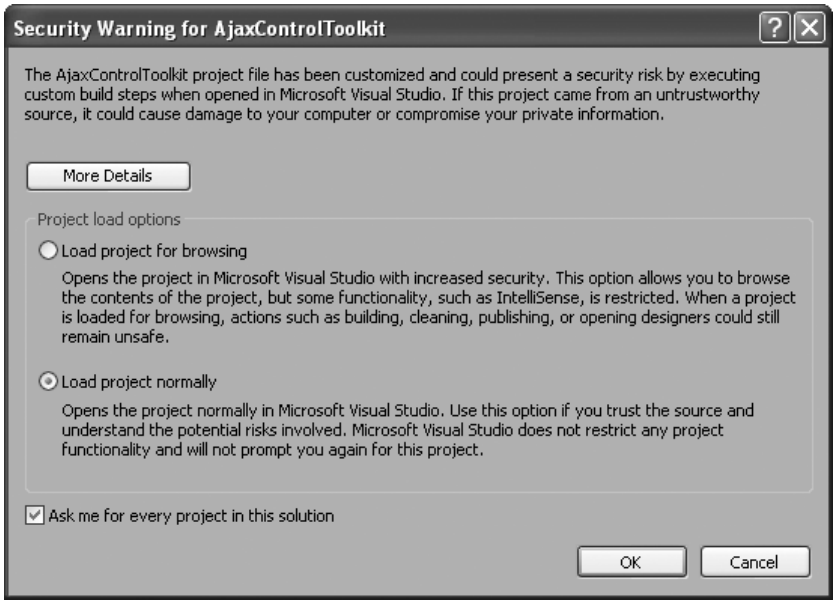


Рис. 13.10. *При открытии проекта в Visual Studio это предупреждение можно проигнорировать*

Когда работа будет закончена, снова запустите утилиту для работы с исправлениями. На этот раз в начальном диалоге выберите второй вариант – Prepare Patch for Submission (подготовка исправлений к передаче). Утилита попросит вас подтвердить, что вы действительно протестировали и прокомментировали свой код (рис. 13.11). Если вы добавили новую функциональность, вам также необходимо будет представить результаты ее тестирования. Посетите тестовый веб-сайт, чтобы просмотреть имеющиеся тесты и добавить свои.

На следующем шаге необходимо указать имя каталога, куда ранее был загружен пакет ASP.NET AJAX Control Toolkit. По умолчанию утилита выбирает путь к последней загрузке (рис. 13.12).

После этого вам будет представлен список измененных вами файлов (рис. 13.13). Снимите флажки напротив тех файлов, которые вообще не нужны, – например, файл *Web.config*. Затем по щелчку на ссылке Diff (различия) напротив каждого файла будет выполнено сравнение измененного файла с его оригинальной версией и на экран будет выведен подробный листинг всех произведенных изменений (рис. 13.14).

Теперь необходимо составить достаточно осмысленное описание ваших исправлений (рис. 13.15) и предоставить свою контактную информацию на тот случай, если разработчики пожелают связаться с вами. Вы должны понимать, что эта информация будет общедоступна. В заключение щелкните на кнопке Finish (закончить), чтобы сохранить исправления в ZIP-файле (рис. 13.16).

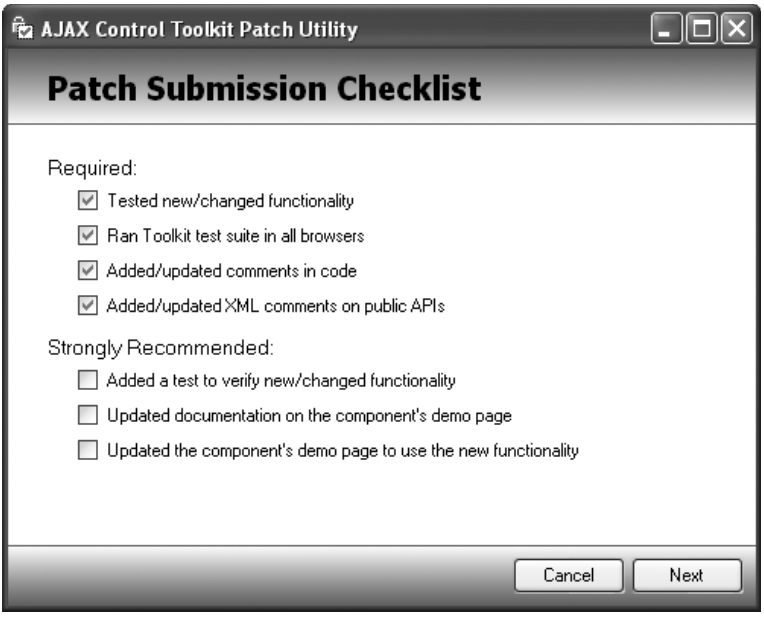


Рис. 13.11. *Проверка перед передачей исправлений*

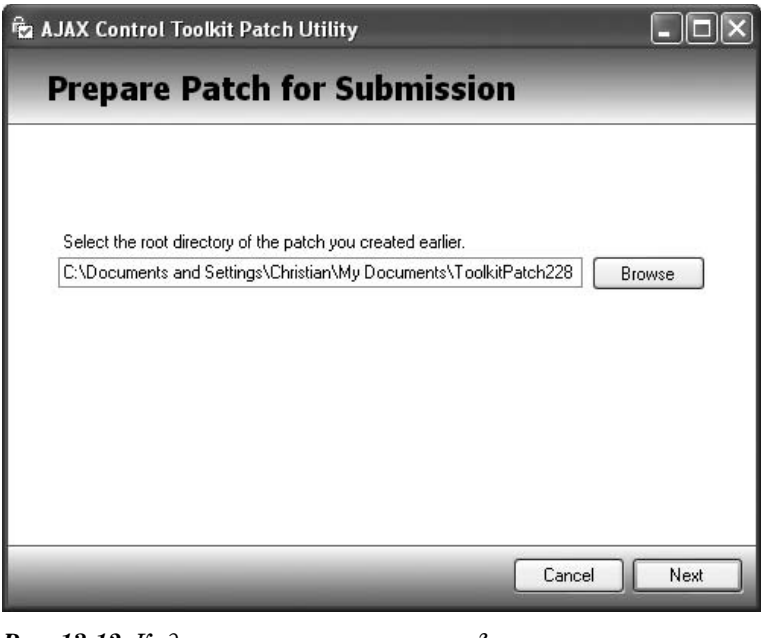


Рис. 13.12. *Куда сохранить исправления?*

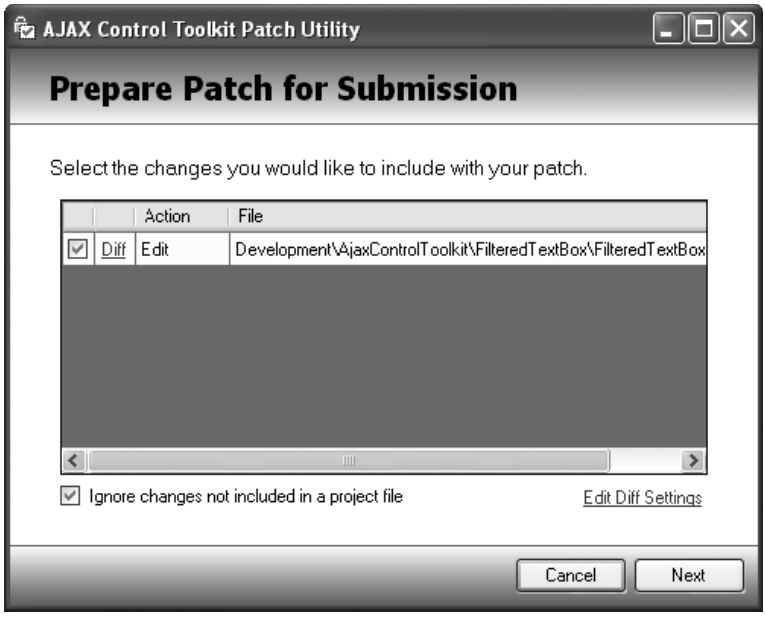


Рис. 13.13. *Список измененных файлов*

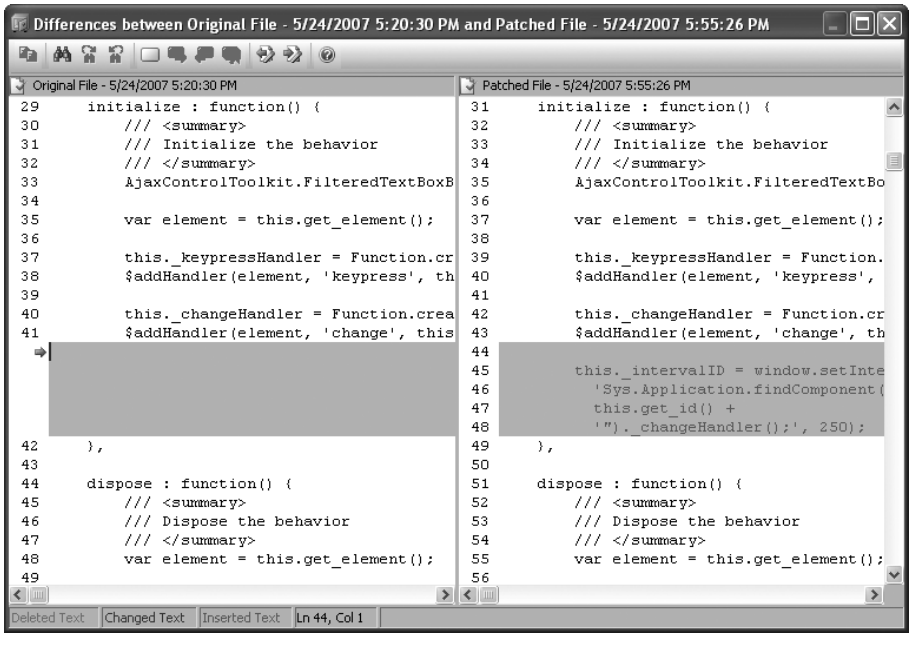


Рис. 13.14. *Различия между старой и новой версиями файла*

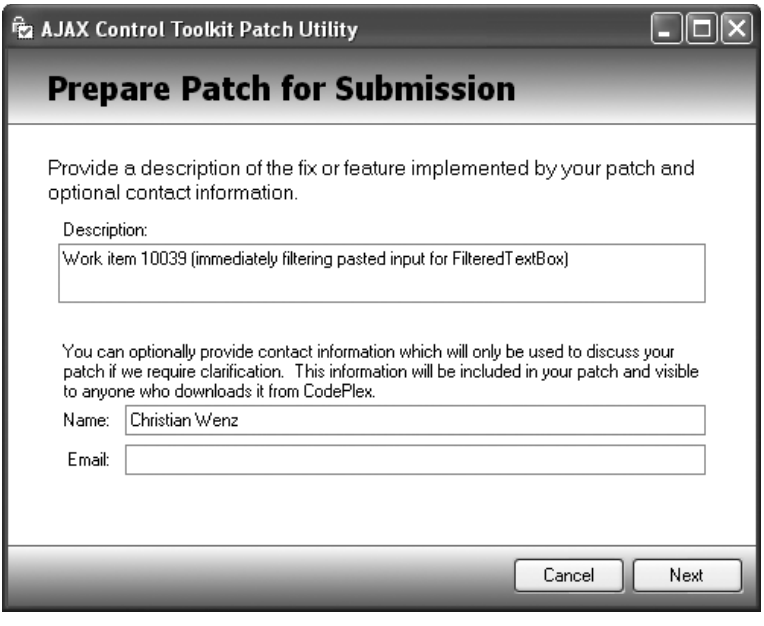


Рис. 13.15. *Информация об исправлении*

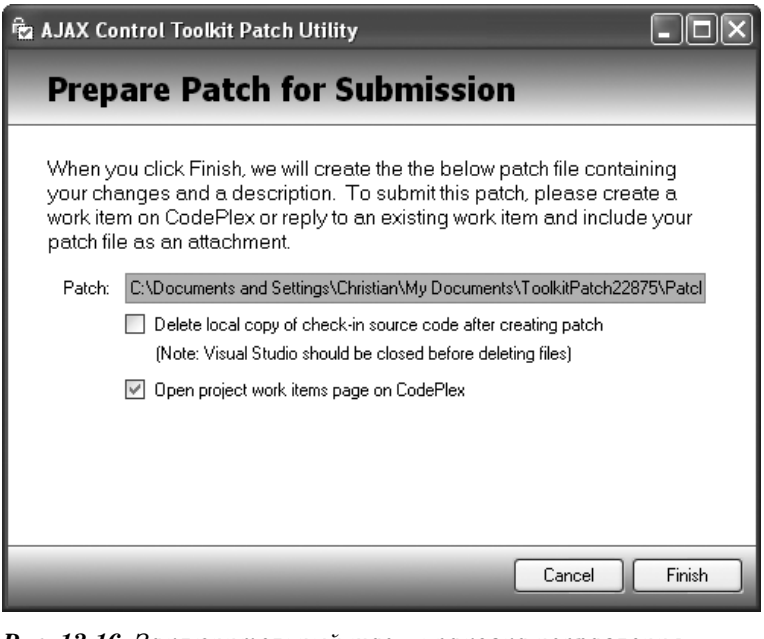


Рис. 13.16. *Заключительный шаг – упаковка исправления*

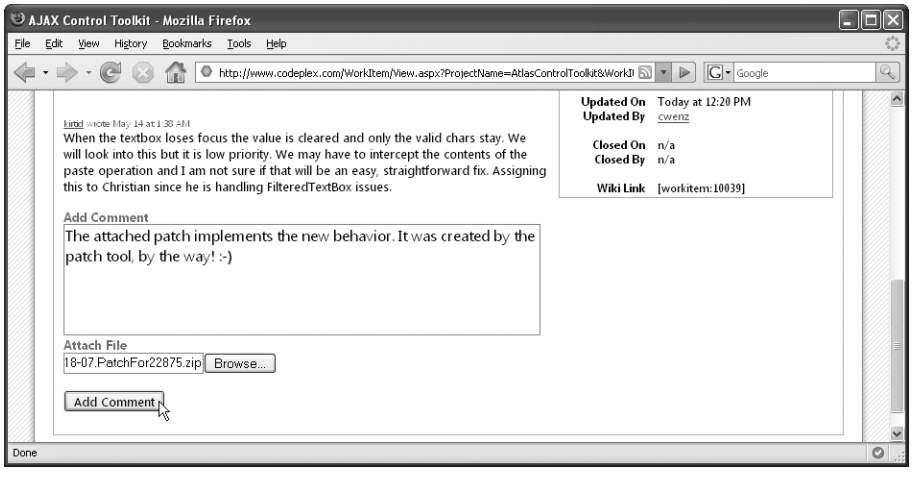


Рис. 13.17. *Загрузка исправлений в рабочий раздел (пункт) CodePlex*

И самый последний шаг: посетите страницу ASP.NET AJAX Control Toolkit на сайте CodePlex (*http://www.codeplex.com/AtlasControlToolkit*) и перейдите к пункту, над которым вы работали (запись в системе слежения за ошибками). Если пункта еще не существует, создайте его.

Затем загрузите свои исправления и напишите короткий комментарий, описывающий их (рис. 13.17). Если у разработчиков из проекта ASP.NET AJAX Control Toolkit появятся вопросы, они свяжутся с вами, или ваши исправления просто будут приняты и внесены.

Существует масса способов стимулирования лиц, оказывающих содействие проекту, например выставление баллов за внесенные исправления, а наиболее активным разработчикам предлагают присоединиться к проекту в качестве официальных сотрудников. Надеюсь вскоре увидеть вас в проекте!

Подведение итогов

В этой главе вы узнали, как установить и использовать пакет ASP.NET AJAX Control Toolkit. Вы также узнали, как с помощью пакета создать свой собственный элемент управления. Измененная и дополненная версия примера, приведенного в этой главе, в настоящее время вошла в состав пакета. Отыщите компонент `FilteredTextBox` и испытайте его в деле!

Для дополнительного чтения

http://ajax.asp.net/default.aspx?tabid=47&subtabid=477

Сайт Microsoft, посвященный пакету ASP.NET AJAX Control Toolkit, содержащий примечания к вышедшим версиям и демонстрационные примеры.

http://www.codeplex.com/Wiki/View.aspx?ProjectName=AtlasControlToolkit

Сайт сообщества, расположенный на сайте CodePlex, новый сайт компании Microsoft для размещения проектов с совместно используемыми исходными текстами.

http://www.microsoft.com/resources/sharedsource/licensingbasics/sharedsourcelicenses.mspx

Текст лицензии Microsoft Permissive License находится на сайте проекта ASP.NET AJAX Control Toolkit, а этот сайт предоставляет общий обзор и описание других лицензий Microsoft на совместное использование исходных текстов.

IV

ASP.NET AJAX Futures

Глава 14. Клиентские элементы управления

Глава 15. Привязка и проверка данных

Глава 16. Использование аспектов клиентского поведения и компонентов

Глава 17. Использование данных, размещенных на стороне сервера

Глава 18. Связь с внешними веб-службами

Глава 19. Использование анимационных эффектов

Глава 20. Кнопки Назад/Вперед и закладки

Глава 21. Веб-части

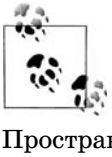
14

Клиентские элементы управления

В этой главе рассказывается о клиентских элементах управления, входящих в состав пакета ASP.NET AJAX Futures. Эти элементы управления имитируют поведение веб-элементов управления ASP.NET и обеспечивают непротиворечивость разработки веб-приложений как на стороне сервера, так и на стороне клиента. Кроме того, они поддерживают такую интересную возможность, как привязка данных, которая будет рассматриваться в главе 15.

Введение в клиентские элементы управления ASP.NET AJAX

В базовой версии ASP.NET AJAX 1.0 отсутствуют исключительно клиентские элементы управления. Однако их можно найти в пакете Futures, где они размещены в пространстве имен `Sys.Preview.UI`. Пространство имен `Sys.Preview.UI` — это эквивалент на стороне клиента хорошо известного пространства имен `Web.UI` в ASP.NET.



В предварительной версии ASP.NET AJAX на стороне клиента пространство имен называлось `Web.UI` и `Sys.UI`.

Пространство имен `Sys.Preview.UI` содержит большое число HTML- и веб-элементов управления ASP.NET AJAX. Элементы управления ASP.NET AJAX обладают функциональностью, похожей на функциональность серверных элементов управления ASP.NET, но не идентичной ей полностью. Они реализуют непротиворечивую модель, не зависящую от типа броузера, которая дает возможность программному коду JavaScript получать и изменять значения свойств клиентских элементов управления. При попытке использования элементов управления без обращения к платформе ASP.NET AJAX вам потребовались бы недюжинные

знания JavaScript. Кроме того, вам пришлось бы проявить невероятную ловкость при программировании обходных путей, учитывающих несовместимость броузеров.

В табл. 14.1 перечислены элементы управления, входящие в состав пакета Futures. В таблице перечислены элементы HTML, с которыми работают элементы управления ASP.NET AJAX, а также эквивалентные им объекты и методы JavaScript DOM, которые вам пришлось бы использовать в противном случае.

Таблица 14.1. Элементы управления ASP.NET AJAX

Элемент управления ASP.NET AJAX	Описание	HTML-элемент	Эквивалент JavaScript
Sys.Preview.UI.Window	Реализует всплывающие окна JavaScript	нет	window.alert(), window.confirm(), window.prompt()
Sys.Preview.UI.Label	Реализует внутристрочный встроенный элемент или метку	, <label>	Label
Sys.Preview.UI.Image	Реализует элемент-изображение	<img>	Image
Sys.Preview.UI.HyperLink	Реализует ссылку	<a href="...">>	Link
Sys.Preview.UI.Button	Реализует кнопку	<input type="button">, <input type="submit">, <input type="reset">, <button>	button
Sys.Preview.UI.CheckBox	Реализует флажок	<input type="checkbox">	Checkbox
Sys.Preview.UI.Selector	Реализует список или раскрывающийся список	<select>	Select
Sys.Preview.UI.TextBox	Реализует текстовое поле	<input type="text">, <inputtype="password">, <textarea>	text, password, textarea

Использование элементов управления ASP.NET AJAX

Элементы управления из адресного пространства Sys.Preview.UI в платформе ASP.NET AJAX делятся на две большие группы. В первую

группу входят некоторые из элементов управления, которые представляют собой абстракцию, упрощающую использование JavaScript при решении различных задач. Они не являются элементами управления в обычном смысле и не предназначены для отображения элементов пользовательского интерфейса на странице. Элементы управления из второй группы обеспечивают доступ из JavaScript к HTML-элементам на текущей странице. В этом разделе будут рассмотрены обе группы элементов.

Доступ к методам JavaScript

Одним из элементов управления, который представляет собой абстракцию функциональности JavaScript, является реализация диалогов в виде класса `Sys.Preview.UI.Window`. В языке JavaScript поддерживаются три типа модальных панелей диалогов:

`window.alert()`

Диалог с кнопкой OK

`window.confirm()`

Диалог с кнопками OK/Cancel (OK/Отмена) или Yes/No (Да/Нет)

`window.prompt()`

Диалог с полем ввода и кнопкой OK

Внутри класса `Sys.Preview.UI.Window` функциональность обращения к методам `window.alert()` и `window.confirm()` инкапсулирована в методе `messageBox()`. По умолчанию этот метод выводит диалог с помощью `window.alert()`, что соответствует стилю диалога `Sys.Preview.UI.MessageBoxStyle.OK`. Альтернативный вариант заключается в использовании стиля `Sys.Preview.UI.MessageBoxStyle.OKCancel`, которому соответствует вызов метода `window.confirm()`.

А что же с методом `window.prompt()`? С целью соблюсти единство стиля с Visual Basic данный диалог реализован в виде метода `inputBox()`, а не `messageBox()`.

В следующем примере реализованы все три варианта вывода модальных диалогов на стороне клиента. Для вызова диалогов средствами ASP.NET AJAX на странице использованы три кнопки:

```
<input type="button" value="MessageBoxOK" onclick="MessageBoxOKClick();" />
<input type="button" value="MessageBoxOKCancel"
       onclick="MessageBoxOKCancelClick();" />
<input type="button" value="InputBox" onclick="InputBoxClick();" />
```

Каждая из трех функций – `MessageBoxOKClick()`, `MessageBoxOKCancelClick()` и `InputBoxClick()` – вызывает метод объекта `Sys.Preview.UI.window` платформы ASP.NET AJAX, как показано в следующем фрагменте:

```
<script language="JavaScript" type="text/javascript">
function MessageBoxOKClick() {
```

```
    Sys.Preview.UI.Window.messageBox("Using Sys.Preview.UI.Window");
  }
  function MessageBoxOKCancelClick() {
    Sys.Preview.UI.Window.messageBox("Using Sys.Preview.UI.Window",
      Sys.Preview.UI.MessageBoxStyle.OKCancel);
  }
  function InputBoxClick() {
    Sys.UI.Window.inputBox("Using Sys.Preview.UI.Window", "<enter text here>");
  }
  </script>
```

Для использования функциональности ASP.NET AJAX к странице необходимо подключить библиотеку ASP.NET AJAX. Все, что необходимо, возьмет на себя элемент `ScriptManager`:

```
<asp:ScriptManager runat="server">
```

Он загрузит базовую библиотеку ASP.NET AJAX. Однако, чтобы использовать компоненты адресного пространства `Sys.Preview.UI`, необходимо загрузить еще и библиотеку Futures. В отличие от базовой эту библиотеку придется загружать вручную, используя следующий синтаксис:

```
    <Scripts>
      <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                           Name="PreviewScript.js" />
    </Scripts>
  </asp:ScriptManager>
```

В примере 14.1 приводится полный программный код первого примера использования ASP.NET AJAX в этой главе. Не забудьте, чтобы иметь возможность использовать пакет Futures и сборку `Sys.Preview.UI`, вы должны создавать веб-сайт, настроенный на использование функциональности Futures. О том, как это делается, рассказывается в главе 1.

Пример 14.1. Модальные панели JavaScript средствами ASP.NET AJAX

```
ControlMessageBox.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
  function MessageBoxOKClick() {
    Sys.Preview.UI.Window.messageBox("Using Sys.Preview.UI.Window");
  }

  function MessageBoxOKCancelClick() {
    Sys.Preview.UI.Window.messageBox("Using Sys.Preview.UI.Window",
      Sys.Preview.UI.MessageBoxStyle.OKCancel);
  }
```

```
    function InputBoxClick() {
      Sys.Preview.UI.Window.inputBox("Using Sys.Preview.UI.Window",
        "<enter text here>");
    }
    </script>
  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager runat="server">
        <Scripts>
          <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                               Name="PreviewScript.js" />
        </Scripts>
      </asp:ScriptManager>
      <div>
        <input type="button" value="MessageBoxOK"
               onclick="MessageBoxOKClick();" />
        <input type="button" value="MessageBoxOKCancel"
               onclick="MessageBoxOKCancelClick();" />
        <input type="button" value="InputBox" onclick="InputBoxClick();" />
      </div>
    </form>
  </body>
  </html>
```

На рис. 14.1 показан результат щелчка на кнопке `InputBox`[1].

Рассмотренные возможности достаточно удобны, но они не представляют собой ничего особенного – это самая обычная функциональность JavaScript, обернутая в элементы управления ASP.NET AJAX. Однако существуют и другие элементы управления, которые обладают более широкими возможностями.

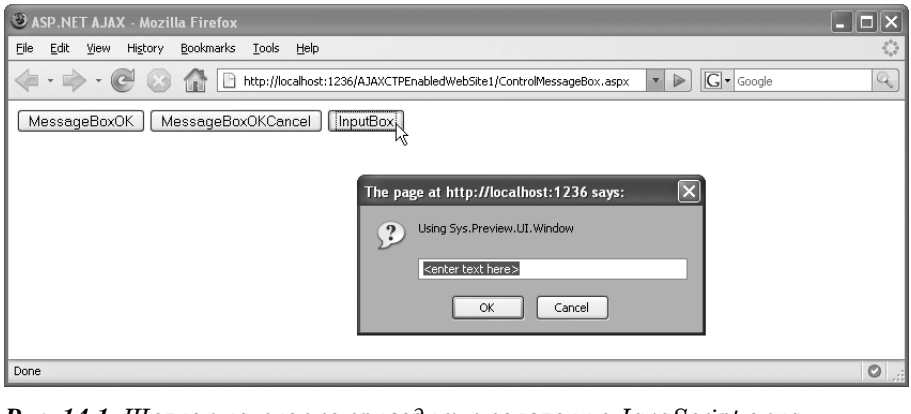


Рис. 14.1. *Щелчок на кнопке приводит к появлению JavaScript-окна*

[1] Отметим, что этот и последующие примеры демонстрируются автором книги на отладочном сервере с другим портом (1236). – *Примеч. науч. ред.*

Доступ к элементам HTML

Элементы управления ASP.NET AJAX также позволяют вставлять HTML-элементы в страницу и обращаться к ним на стороне клиента с использованием объектно-ориентированного синтаксиса. Это означает, что даже при обращении к элементам HTML вы можете использовать на стороне клиента уровень абстракции для доступа к их содержимому.

Первое время синтаксис, используемый в ASP.NET AJAX для доступа к элементам HTML, будет казаться вам странным. Рассмотрим страницу, которая содержит элемент `<span>`, как показано в следующем фрагменте:

```
<span id="Label1">This is a label</span>
```

Получить доступ к этому элементу из JavaScript можно следующим образом:

```
var label = document.getElementById("Label1")
```

ASP.NET AJAX предоставляет для этого метода удобный псевдоним – `$get()`:

```
var label = $get("Label1")
```

Затем можно задать значения для свойств этого элемента, включая информацию о стилях. При использовании JavaScript вам пришлось бы писать отдельный код для разных броузеров. А для этого, как уже отмечалось ранее, требуется основательное знание JavaScript и DOM – простого знания синтаксиса будет недостаточно.

В ASP.NET AJAX используется иной подход. Вы *должны* знать свойства и методы класса, соответствующего клиентскому элементу управления ASP.NET AJAX, и не более того (просмотрите еще раз содержимое табл. 14.1). В нашем случае с элементом `<span>` используется класс `Sys.Preview.UI.Label`. Программный код должен создать экземпляр этого класса и передать ему значение атрибута ID HTML-элемента. Значение ID должно быть задано определенным образом: с помощью упоминавшегося выше метода `$get()` с фактическим значением ID в скобках:

```
var label = new Sys.Preview.UI.Label($get("Label1"));
```

В действительности вы приводите объект к типу ссылки на него.

Затем вызовом метода `initialize()` регистрируются делегаты и обработчики событий. Впрочем, этот шаг для данного конкретного примера необязателен, но обычно рекомендован для большей части сценариев.

```
label.initialize();
```

Если вы не используете обработчики событий (как в следующих нескольких примерах), то вызов метода `initialize()` можно опустить.

Метки

Элемент управления ASP.NET AJAX – `Label` поддерживает два дополнительных метода, описываемых ниже. Оба они демонстрируются в примере 14.2.

`get_text()`

Возвращает текущий текст из элемента

`set_text()`

Записывает (изменяет) текст в элемент

JavaScript и DOM броузеры не имеют эквивалента свойству `InnerText` в ASP.NET. Свойство, с которым работают методы `get_text()` и `set_text()`, – `innerHTML`, поэтому вам придется вылавливать служебные символы и экранировать их, чтобы избежать побочных эффектов.

В примере 14.2 показано, как можно манипулировать элементом управления `label`. В примере демонстрируются три действия:

1. Создание объекта `Sys.Preview.UI.Label` на стороне клиента.
2. Чтение прежнего текста с помощью метода `get_text()`.
3. Запись нового текста с помощью метода `set_text()`.

Пример 14.2. Использование элемента управления ASP.NET AJAX – Label

```
ControlLabel.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
  function pageLoad() {
    var label = new Sys.Preview.UI.Label($get("Label1"));
    var d = new Date();
    var time = d.getHours() + ":" + d.getMinutes() + ":" + d.getSeconds();
    label.set_text(label.get_text() + time);
  }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server">
      <Scripts>
        <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                             Name="PreviewScript.js" />
```

```
        </Scripts>
      </asp:ScriptManager>
      <div>
        <span id="Label1">time goes here: </span>
      </div>
    </form>
  </body>
  </html>
```

После загрузки страницы определяется текущее время и его значение помещается в элемент `<span>`. На рис. 14.2 показан результат работы примера. Чтобы увидеть новое значение времени, достаточно обновить страницу в броузере.

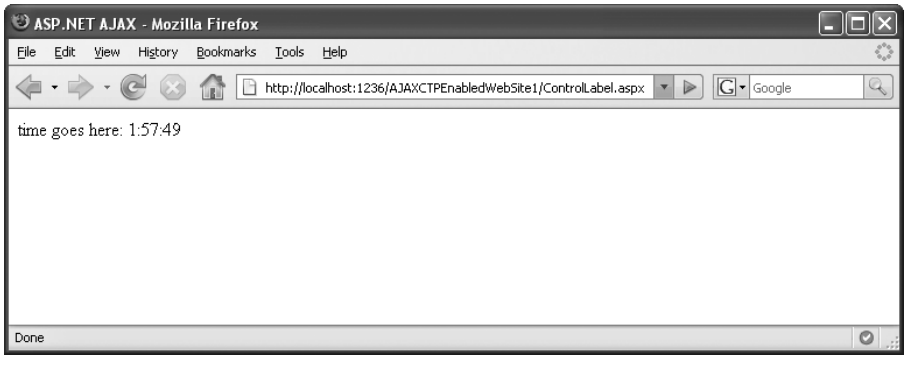


Рис. 14.2. *Значение текущего времени выводится в метке*

Изображения

Изображения на странице представляет HTML-тег `<img>`. Класс `Sys.Preview.UI.Image` реализует ASP.NET AJAX-версию изображения (представленного в дереве DOM объектом `Image`). Кроме обычных методов, перечисленных ранее, класс ASP.NET AJAX `Image` поддерживает следующие дополнительные методы:

`get_alternateText()`

Возвращает значение атрибута `alt`

`set_alternateText()`

Изменяет значение атрибута `alt`

`get_height()`

Возвращает высоту изображения

`set_height()`

Изменяет высоту изображения

`get_width()`

Возвращает ширину изображения

`set_width()`

Изменяет ширину изображения

`get_imageURL()`

Возвращает относительный или абсолютный адрес URL изображения (атрибут `src`)

`set_imageURL()`

Изменяет относительный или абсолютный адрес URL изображения (атрибут `src`)

Здесь также стандартные свойства DOM инкапсулированы в состав класса. Вам не нужно обладать глубокими познаниями JavaScript, достаточно лишь привыкнуть к использованию методов, предлагаемых платформой ASP.NET AJAX. Пример 14.3 показывает, как можно манипулировать пустым элементом `<img>` на странице. Изначально он выглядит следующим образом:

```
<img id="Image1" />
```

По умолчанию функция проверки корректности XHTML в Visual Studio будет предупреждать об отсутствующих атрибутах, но мы будем устанавливать обязательные атрибуты `src` и `alt` с помощью JavaScript.

Пример 14.3. Использование элемента управления ASP.NET AJAX – Image

```
ControlImage.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
    function pageLoad() {
      var image = new Sys.Preview.UI.Image($get("Image1"));
      image.set_imageURL("ajaxlogo.png");
      image.set_alternateText("ASP.NET AJAX logo");
    }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server">
      <Scripts>
        <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                             Name="PreviewScript.js" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <img id="Image1" />
```

```
    </div>
  </form>
</body>
</html>
```

Результат работы примера показан на рис. 14.3. Для этого примера вам необходимо загрузить изображение *ajaxlogo.png* (по адресу: *http://ajax.asp.net/images/ajax-poster-photo-logo.png*) и поместить его в корневой каталог веб-сервера. Файл с программным кодом примера вы найдете на сайте книги (*http://www.oreilly.com/catalog/9780596514242*).

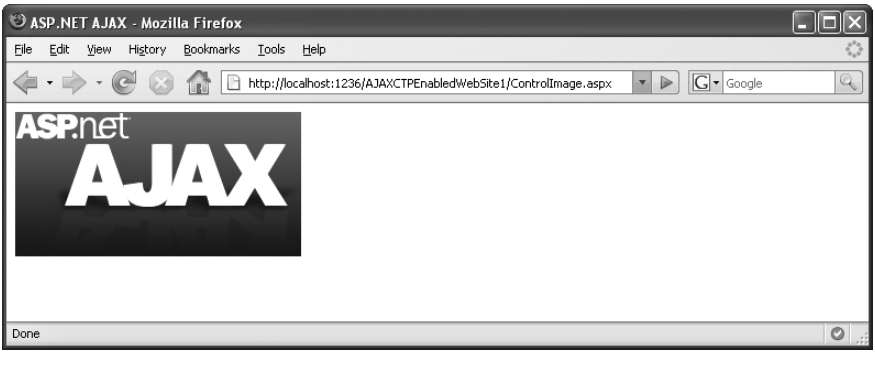


Рис. 14.3. *Элемент управления Image; альтернативный текст выводится в окне Properties (Свойства)*

Гиперссылки

Ссылки на другие страницы и документы в HTML представляет элемент `<a>`, и он же используется в закладках. В ASP.NET AJAX гиперссылки представлены классом `Sys.Preview.UI.HyperLink`. Этот класс реализует методы `get_navigateURL()` и `set_navigateURL()`, которые позволяют изменять адрес ссылки (только адрес URL, но не HTML-атрибут *target* для адресации фрейма или окна). Он также предоставляет событие `click`, для которого вы можете реализовать свой обработчик. (Обработка событий будет описываться ниже, в этой же главе, в разделе «Обработка событий в элементах управления».)

В примере 14.4 создается пустая ссылка (`<a></a>`), а адрес, на который она ссылается, будет изменяться динамически. В примере ссылка указывает на то же самое изображение логотипа ASP.NET AJAX, которое использовалось в предыдущем примере.

Текст ссылки непосредственно через элемент управления `Link` изменить нельзя. Впрочем, ссылка не обязательно должна быть текстовой ссылкой, она может содержать изображение или другой элемент. Поэтому текст ссылки можно представить как другой элемент. Если вам требуется изменять текст ссылки, вам нужно поместить внутрь ссылки другой элемент (и задать для него ID).

Пример 14.4. Использование элемента управления ASP.NET AJAX – Link

```
ControlHyperLink.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
    function pageLoad() {
      var link = new Sys.Preview.UI.HyperLink($get("Link1"));
      link.set_navigateURL("http://ajax.asp.net/");
      var image = new Sys.Preview.UI.Image($get("Image1"));
      image.set_imageURL("ajaxlogo.png");
      image.set_alternateText("ASP.NET AJAX logo");
    }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server">
      <Scripts>
        <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                             Name="PreviewScript.js" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <a id="Link1"><img id="Image1" /></a>
    </div>
  </form>
</body>
</html>
```

Результат работы примера приводится на рис. 14.4.

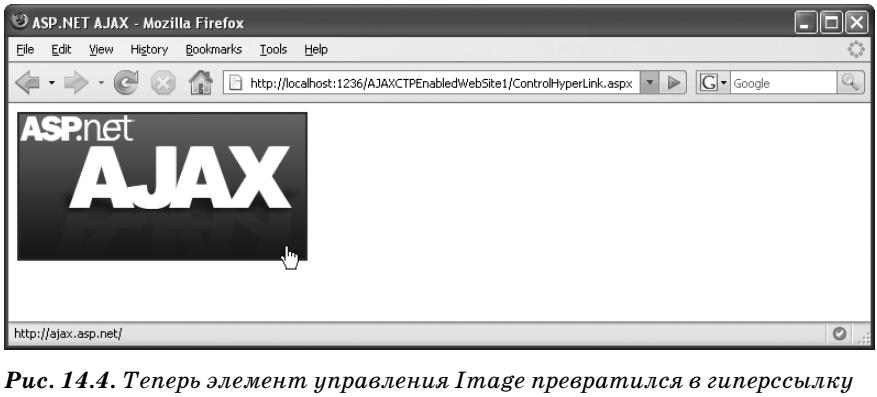


Рис. 14.4. *Теперь элемент управления Image превратился в гиперссылку*

Кнопки

Язык разметки HTML поддерживает кнопки различных типов. Например: кнопка `<input type="submit">` служит для отправки формы, `<input type="reset">` – для очистки формы (для перевода формы в первоначальное состояние) и, наконец, кнопки `<input type="button">` и `<button>`, не имеющие предопределенного поведения и действие которых можно определить с помощью JavaScript. В ASP.NET AJAX кнопки реализованы в виде класса `Sys.Preview.UI.Button`. Этот класс поддерживает следующие методы:

`get_argument()`

Возвращает аргументы команды, которая была послана в результате щелчка на кнопке

`set_argument()`

Устанавливает аргументы кнопки

`get_command()`

Возвращает команду, которая была послана в результате щелчка на кнопке

`set_command()`

Устанавливает команду кнопки

Всякий раз, когда вы устанавливаете значение свойства `argument` или `command`, активируется встроенный механизм обработки событий (который будет описан ниже в этой же главе). Различные способы назначения кнопкам функциональности можно найти в главе 16.

Флажки

Флажки в HTML определяются с помощью элемента `<input type="checkbox">`. Флажок может иметь всего два состояния: установлен или сброшен. Эти состояния можно изменять из JavaScript, а значит, и ASP.NET AJAX также обеспечивает подобную возможность. Метод `set_checked()` позволяет изменить состояние флажка (получением логического значения), а метод `get_checked()` возвращает текущее состояние. Соответствующий класс в ASP.NET AJAX называется `Sys.Preview.UI.CheckBox`.

В примере 14.5 с помощью разметки HTML создается флажок, а с помощью ASP.NET AJAX/JavaScript производится изменение его состояния в значение `true`.

Пример 14.5. Использование элемента управления ASP.NET AJAX – CheckBox

```
ControlCheckBox.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

```
    <title>ASP.NET AJAX</title>

    <script language="JavaScript" type="text/javascript">
      function pageLoad() {
        var checkbox = new Sys.Preview.UI.CheckBox($get("CheckBox1"));
        checkbox.set_checked(true);
      }
    </script>

  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager runat="server">
        <Scripts>
          <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                               Name="PreviewScript.js" />
        </Scripts>
      </asp:ScriptManager>
      <div>
        <input type="checkbox" id="CheckBox1" />
        <label for="CheckBox1">click me!</label>
      </div>
    </form>
  </body>
  </html>
```

Результат работы примера показан на рис. 14.5.

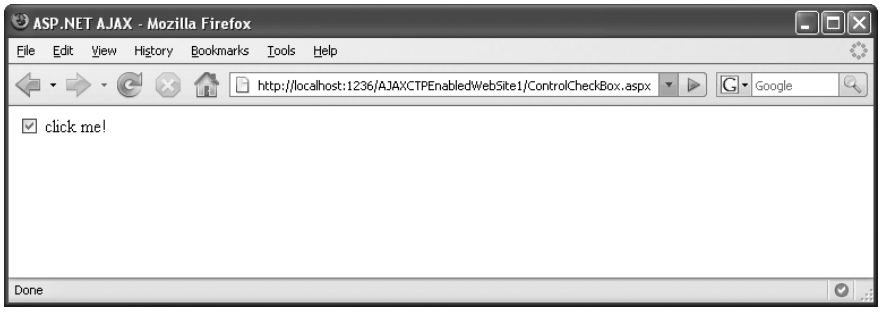


Рис. 14.5. *ASP.NET AJAX установил флажок*

Списки выбора

В языке разметки HTML списки выбора (`<select>...</select>`) могут иметь две формы представления: первая – раскрывающийся список, который требует от пользователя произвести некоторое действие, чтобы отобразить все элементы списка, и вторая – когда некоторые элементы списка уже отображаются. Списки обоих типов в ASP.NET AJAX представлены одним классом `Sys.Preview.UI.Selector`. В отличие от JavaScript классы ASP.NET AJAX *не* предоставляют возможность изменять значения отдельных элементов списка.

Если содержимое списка существует в форме объекта .NET DataTable, имеется возможность воспользоваться механизмом привязки данных (в противном случае вам придется представить список элементов в виде разметки). Этот способ описывается в главе 17.

Тем не менее мы можем продемонстрировать метод get_selectedValue(), который возвращает значение атрибута value выбранного в настоящий момент элемента списка.

Когда форма передается серверу HTTP-методом GET или POST, можно не устанавливать атрибут value, поскольку в этом случае[1] в качестве значения серверу передается заголовок элемента (текст между тегами <option> и </option>). Однако в JavaScript элемент списка без свойства value считается пустым. Поэтому вы *всегда* должны устанавливать свойство value для всех элементов списка.

Поскольку обработка событий пока нами не рассматривается, событие change в примере 14.6 не обрабатывается. Вместо этого состояние списка анализируется каждую секунду. Реализовано это с помощью функции JavaScript setInterval(). Такая методика циклического опроса используется исключительно ради этого примера. В главе 16 представлен лучший способ обеспечить синхронизацию двух элементов с помощью механизма привязки данных.

```
function pageLoad() {
  window.setInterval(
    function() {
      // Обратиться к списку и вывести выбранное значение
    },
    1000);
}
```

В примере 14.6 показано, как используется ASP.NET AJAX для проверки и отображения текущего выбранного значения в элементе (Sys.Preview.UI.Label).

Пример 14.6. Использование элемента управления ASP.NET AJAX – Select

```
ControlSelector.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

[1] Имеется в виду ситуация, когда именно отсутствует атрибут *value* элементов *<option>*. То есть рассматривается вариант *<option>*, но не *<option value="">*. – *Примеч. науч. ред.*

```
    <title>ASP.NET AJAX</title>

    <script language="JavaScript" type="text/javascript">
    var label;
    var select;

    function pageLoad() {
      label = new Sys.Preview.UI.Label($get("Label1"));
      select = new Sys.Preview.UI.Selector($get("Select1"));

      // Выполнять опрос каждую секунду, чтобы определить,
      // какое значение было выбрано.
      window.setInterval(
        function() {
          label.set_text(select.get_selectedValue());
        },
        1000);
    }
    </script>

  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager runat="server">
        <Scripts>
          <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                               Name="PreviewScript.js" />
        </Scripts>
      </asp:ScriptManager>
      <div>
        <select id="Select1" size="3">
          <option value="1">one</option>
          <option value="2">two</option>
          <option value="3">three</option>
        </select><br />
        Selected value: <label id="Label1"></label>
      </div>
    </form>
  </body>
  </html>
```

Результат работы примера показан на рис. 14.6.

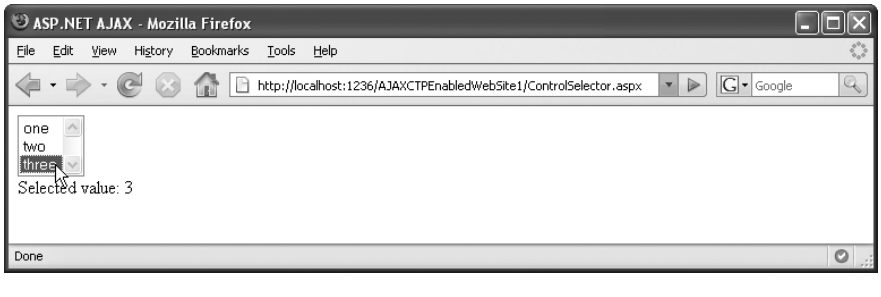


Рис. 14.6. *Выбранное значение переписывается в элемент управления Label*

Метод `get_selectedValue()` представляет определенное удобство, но лишь при работе со списками, допускающими одновременный выбор только одного элемента. Если список допускает одновременный выбор нескольких элементов (`<select multiple="multiple">`), этот метод вернет значение свойства `value` только первого выбранного элемента списка, а не всех выбранных элементов. Чтобы отобрать все выбранные элементы, вам придется обойти все элементы списка в цикле, как показано в следующем фрагменте:

```
var op = document.forms[0].elements["Select1"].options;
for (var i=0; i < op.length; i++) {
  if (op[i].selected) {
    // элемент выбран
  } else {
    // элемент не выбран
  }
}
```

Текстовые поля

Однострочное текстовое поле в языке разметки HTML представлено тегом `<input type="text">`. Управлять этим элементом можно с помощью класса ASP.NET AJAX `Sys.Preview.UI.TextBox`. Платформа ASP.NET AJAX поддерживает обработку событий от клавиатуры и, разумеется, доступ на чтение и запись к текстовому содержимому элемента. Последние две операции выполняются с помощью методов `get_text()` и `set_text()`.

Пример 14.7 читает данные, введенные в текстовое поле, с использованием той же методики циклического опроса, что и в предыдущем примере (`setInterval()`), и периодически копирует их из текстового поля в элемент управления ASP.NET AJAX `Label`.

Пример 14.7. Использование элемента управления ASP.NET AJAX – TextBox

```
ControlTextBox.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
  function pageLoad() {
    window.setInterval(
      function() {
        var label = new Sys.Preview.UI.Label($get("Label1"));
        var textbox = new Sys.Preview.UI.TextBox($get("TextBox1"));
```

```
            label.set_text(textbox.get_text());
          },
          1000);
      }
      </script>

    </head>
    <body>
      <form id="form1" runat="server">
        <asp:ScriptManager runat="server">
          <Scripts>
            <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                                 Name="PreviewScript.js" />
          </Scripts>
        </asp:ScriptManager>
        <div>
          <input type="text" id="TextBox1" /><br />
          Entered value: <label id="Label1"></label>
        </div>
      </form>
    </body>
    </html>
```

Результат работы примера показан на рис. 14.7.

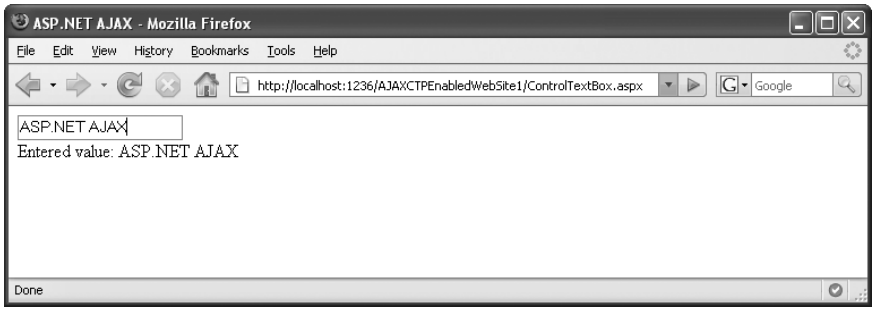


Рис. 14.7. *Текст из текстового поля копируется в метку*

Однострочное текстовое поле (`<input type="text">`), многострочное текстовое поле (`<textarea>`) и поле ввода пароля (`<input type="password">`) имеют одну общую черту: с точки зрения JavaScript все они совершенно одинаково управляются. Свойство `value`, доступное для чтения и для записи, обеспечивает доступ к текстовому содержимому поля. Поэтому вы можете использовать `Sys.Preview.UI.TextBox` для работы со всеми тремя видами полей.

Базовые методы

Как уже обсуждалось выше, в разделе «Введение в клиентские элементы управления ASP.NET AJAX», платформа ASP.NET AJAX поддер-

живает общие методы для всех элементов управления, входящих в пространство имен Sys.Preview.UI. Большинство из них предназначены для организации доступа к свойствам, имеющимся во всех элементах управления. Два примера таких методов – get_accessKey() и set_accessKey(), с помощью которых выполняется контроль над свойством DOM accesskey.

Несколько более эффектный результат имеют методы, управляющие классом CSS элемента. Это существенно упрощает изменение схемы размещения элементов в процессе работы страницы. Ниже приводится список поддерживаемых методов:

addCssClass()

Добавляет в элемент класс CSS

removeCssClass()

Удаляет класс CSS из элемента

toggleCssClass()

Добавляет в элемент класс CSS, если его там еще нет, в противном случае удаляет класс

В примере 14.8 демонстрируется использование метода toggleCssClass(). В нем также определяется текущий класс CSS. Для этого вызывается метод get_element() объекта метки, который возвращает ссылку на фактический элемент DOM. Список классов, используемых в настоящий момент, содержится в свойстве className элемента DOM.

На странице определены и могут дополнять друг друга следующие три класса CSS (каждый класс добавляет свой элемент оформления):

```
<style type="text/css">
.style1 { font-family: Monospace; }
.style2 { border-style: solid; }
.style3 { color: #00f; }
</style>
```

Программный код JavaScript в примере случайным образом выбирает один из этих классов и затем вызывает метод toggleCssClass(). В элементе управления Label периодически отображаются имена используемых классов.

Пример 14.8. Использование базовых методов элементов ASP.NET AJAX для работы с классами CSS

```
ControlCSS.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
```

```
    <style type="text/css">
    .style1 { font-family: Monospace; }
    .style2 { border-style: solid; }
    .style3 { color: #00f; }
    </style>

    <script language="JavaScript" type="text/javascript">
    function pageLoad() {
      window.setInterval(
        function() {
          var label = new Sys.Preview.UI.Label($get("Label1"));
          var rnd = Math.ceil(3 * Math.random());
          label.toggleCssClass("style" + rnd);
          label.set_text(label.get_element().className);
        },
        1000);
    }
    </script>

  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager runat="server">
        <Scripts>
          <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                               Name="PreviewScript.js" />
        </Scripts>
      </asp:ScriptManager>
      <div>
        CSS class(es):
        <label id="Label1">
        </label>
      </div>
    </form>
  </body>
  </html>
```

Результат работы примера показан на рис. 14.8.

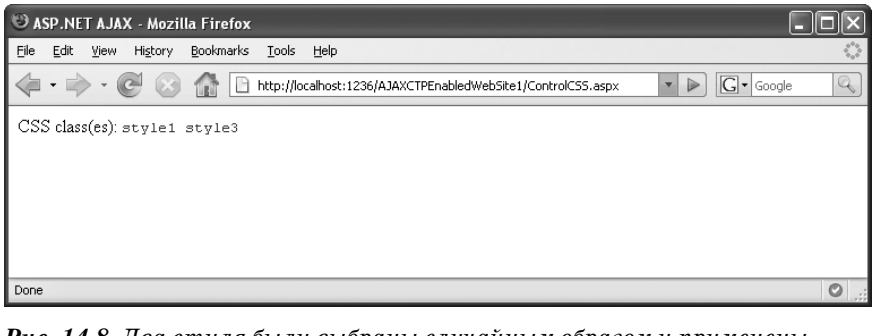


Рис. 14.8. *Два стиля были выбраны случайным образом и применены*

Обработка событий в элементах управления

Для клиентских элементов управления платформа ASP.NET AJAX реализует свой механизм обработки событий. Этот механизм работает немножко не так, как можно было бы ожидать, но все-таки интуитивно оправданно.

Первый и самый важный шаг на пути к использованию этого механизма заключается в вызове метода `initialize()` того элемента, события которого предполагается обрабатывать. Он активирует все механизмы, которые основаны на перехвате событий. Процесс подготовки к обработке событий состоит из двух этапов:

1. Создание функции-обработчика, которая будет вызываться при появлении события. Как и в платформе .NET Framework, функция-обработчик принимает два аргумента: в первом передается объект, породивший событие, а во втором – объект, который в зависимости от типа события может содержать дополнительную информацию о событии.
2. Связывание функции-обработчика с элементом: `<элемент>.add_<имя события>(<имя метода>)`. Этот синтаксис напоминает реализацию делегатов в .NET Framework.

События для кнопок

Вспомните пример с тремя модальными окнами, который приводился в начале этой главы. В этом примере программный код JavaScript, который вызывал появление диалогов, был добавлен непосредственно в объявления кнопок. То же самое можно сделать и при использовании библиотеки ASP.NET AJAX. Но при таком подходе вы не выигрываете от использования ASP.NET AJAX по сравнению с «чистым» JavaScript, за исключением уверенности в том, что библиотека ASP.NET AJAX будет полностью загружена до того, как любой программный код JavaScript будет подключен к элементу. Однако основная идея ASP.NET AJAX состоит в том, чтобы сблизить разработку серверной и клиентской частей, привнести новые объектно-ориентированные возможности и обеспечить независимость от типа броузера, используемого клиентом. Таким образом, по-прежнему имеет смысл использовать ASP.NET AJAX для решения задач, которые легко могут быть решены с помощью JavaScript.

Пример 14.9 представляет собой пересмотренную версию примера 14.1 «с тремя окнами», в которую была добавлена возможность обработки событий средствами ASP.NET AJAX. В этом примере ссылки на HTML-кнопки организованы с помощью класса `Sys.Preview.UI.Button` и ассоциированы с событием `click` (что совершенно очевидно).

Пример 14.9. Использование событий элементов управления ASP.NET AJAX – Button

```
ControlEventButton.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
    function pageLoad() {
      var button1 = new Sys.Preview.UI.Button($get("MessageBoxOK"));
      var button2 = new Sys.Preview.UI.Button($get("MessageBoxOKCancel"));
      var button3 = new Sys.Preview.UI.Button($get("InputBox"));

      button1.initialize();
      button2.initialize();
      button3.initialize();

      button1.add_click(MessageBoxOKClick);
      button2.add_click(MessageBoxOKCancelClick);
      button3.add_click(InputBoxClick);
    }

    function MessageBoxOKClick() {
      Sys.Preview.UI.Window.messageBox("Using Sys.Preview.UI.Window");
    }

    function MessageBoxOKCancelClick() {
      Sys.Preview.UI.Window.messageBox("Using Sys.Preview.UI.Window",
        Sys.Preview.UI.MessageBoxStyle.OKCancel);
    }

    function InputBoxClick() {
      Sys.Preview.UI.Window.inputBox("Using Sys.Preview.UI.Window",
        "<enter text here>");
    }

  </script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server">
      <Scripts>
        <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                             Name="PreviewScript.js" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="button" value="MessageBoxOK" id="MessageBoxOK" />
      <input type="button" value="MessageBoxOKCancel"
             id="MessageBoxOKCancel" />
```

```
      <input type="button" value="InputBox" id="InputBox" />
    </div>
  </form>
</body>
</html>
```

Подключать к элементам обработчики событий можно двумя разными способами. Первый заключается в том, чтобы установить обработчик с помощью функции `$addHandler()`:

```
$addHandler(button1.get_element(), 'click', MessageBoxOK);
```

Функцию `$addHandler()` вы уже видели в действии в предыдущих главах, и мы коротко обсуждали ее в главе 4.

Если у вас имеется несколько обработчиков для одного элемента, – под рукой функция `$addHandlers()`:

```
$addHandlers(
  <current element>.get_element(),
  {
    <event>: <handler function>,
    <another event>: <another handler function>
  }
);
```

События для списков

В клиентских элементах управления ASP.NET AJAX реализовано событие `propertyChanged`, которому нет аналога в JavaScript. Оно используется во всех элементах управления и уведомляет программный код о том, что что-то изменилось: была нажата клавиша на клавиатуре, был выбран элемент списка и тому подобное. В этом случае вместе с событием передается имя изменившегося свойства.

Кроме того, для каждого элемента поддерживается возможность работать с отдельными событиями изменения, благодаря чему вы точно можете определить, какое значение изменилось. Например, когда в списке выбирается некий элемент, возникает событие `selectionChanged` (в JavaScript это событие называется `changed`). Для иллюстрации этого события был переписан один из предыдущих примеров (пример 14.7). На этот раз мы не будем периодически проверять список выбора на наличие изменений; вместо этого будет перехватываться соответствующее событие. Не забывайте о необходимости вызвать метод `initialize()`, в противном случае события невозможно будет перехватить. В примере 14.10 приводится программный код, который обрабатывает событие `selectionChanged` элемента управления `Selector`.

Пример 14.10. Использование событий списков выбора в ASP.NET AJAX

```
ControlEventSelector.aspx

<%@ Page Language="C#" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
  var select;
  var label;

  function pageLoad() {
    select = new Sys.Preview.UI.Selector($get("Select1"));
    label = new Sys.Preview.UI.Label($get("Label1"));

    select.initialize();
    select.add_selectionChanged (listHasChanged);
  }
  function listHasChanged(sender, args) {
    label.set_text(select.get_selectedValue());
  }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server">
      <Scripts>
        <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                             Name="PreviewScript.js" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <select id="Select1" size="3">
        <option value="1">one</option>
        <option value="2">two</option>
        <option value="3">three</option>
      </select><br />
      Selected value: <label id="Label1"></label>
    </div>
  </form>
</body>
</html>
```

Быстродействие этого примера намного выше, чемв предыдущей версии, так как приложение реагирует на выбор нового элемента в списке немедленно, а не в конце каждого временного интервала длительностью 1000 миллисекунд. Результат работы этого примера показан на рис. 14.9.

Рис. 14.9. *Щелчок в списке приводит к немедленному отображению выбранного элемента*

Подведение итогов

В данной главе было показано, что предлагается платформой ASP.NET AJAX в пространстве имен `Sys.Preview.UI` на стороне клиента – в частности, специфичный для ASP.NET AJAX способ работы с HTML-элементами из программного кода JavaScript. Кроме того, здесь описывается возможность обработки событий средствами ASP.NET AJAX. В следующей главе будет показано, как выполняется привязка данных к клиентским элементам, чтобы избежать необходимости выполнять присваивание значений вручную. Механизм привязки данных также позволяет выполнять синхронизацию элементов – то есть связывать элементы между собой так, что изменения в одном элементе отражаются в другом и наоборот.

Для дополнительного чтения

http://quickstarts.asp.net/Futures/ajax/doc/intro.aspx

Предварительная версия документации Microsoft к пакету ASP.NET AJAX Futures

15

Привязка и проверка данных

Привязка данных (*data binding*) – это средства, с помощью которых элемент управления (то есть элемент HTML-страницы) связан с данными. Обычно это делается с целью отображения данных перед пользователем. Например, с помощью механизма привязки данных можно связать содержимое текстового поля с меткой или преобразовать введенные пользователем данные во что-то еще (например, в HTML) и произвести дальнейшую обработку. Очень часто привязка данных связана с выборкой информации из базы данных. В этой главе описываются основы механизма привязки данных в ASP.NET AJAX, а в главе 17 рассказывается о том, как с помощью ASP.NET AJAX получить доступ к данным на стороне сервера.

В примерах из главы 14 не использовался декларативный код для привязки значений к элементам управления на странице, хотя декларативная модель – это одно из преимуществ платформы ASP.NET AJAX. Кроме того, мы сочли необходимым использовать один или два грубых (или, выражаясь дипломатическим языком, – «не совсем элегантных») приема, таких как использование функции `setInterval()`, для синхронизации двух HTML-элементов. В этой главе вы изучите xml-сценарии – декларативную разметку, которая поддерживается пакетом ASP.NET AJAX Futures.

Привязка данных

Механизм привязки данных связывает данные с HTML-элементами, которые обеспечивают визуальное представление этих данных. В ASP.NET привязка данных используется с такими элементами управления, как `GridView`, `FormView` и `DetailsView`. При этом, разумеется, имеется возможность привязывать данные и к другим объектам, таким как маркированные списки.

Платформа ASP.NET AJAX предлагает два подхода к привязке данных. Первый – программный, а второй основан на использовании уникальной разновидности разметки XML, которая интерпретируется платформой ASP.NET AJAX на лету.

Программный способ привязки данных

Программная привязка данных кажется более сложной, чем есть на самом деле. В сущности, требуется создать экземпляр класса, а затем установить значения некоторых его свойств. Класс, который используется платформой ASP.NET AJAX для привязки данных на стороне клиента, называется `Sys.Preview.Binding`.

После создания привязки путем создания экземпляра класса необходимо задать следующую информацию:

Контекст данных

Имя элемента, содержащего данные для привязки

Путь к данным

Имя свойства, которое будет служить источником данных для привязки (binding source)

Свойство

Имя свойства, которое будет служить пунктом назначения привязки (binding target)

Преобразователь

Необязательный программный код, который будет выполнять те или иные преобразования исходных данных перед записью их в пункт назначения

Направление привязки

Значение, которое определяет, являются ли данные входящими, исходящими или могут перемещаться в обоих направлениях

Некоторые из терминов, такие как *Путь к данным* или *Свойство*, используются в контексте, новом для пользователя ASP.NET. Такая терминология была выбрана для обеспечения совместимости со словарем, используемым в Windows Presentation Foundation (WPF – графическая подсистема Windows) в операционной системе Windows Vista. Но этот подход достаточно очевиден: определяется объект привязки, к которому может быть присоединен целевой элемент (именно поэтому для источника вам необходимо знать элемент-источник и путь, но для приемника достаточно знать только целевое свойство-имя объекта-приемника).

Для необязательного изменения данных в процессе передачи используется преобразователь (*transformer*). Есть возможность использовать встроенные преобразователи, предлагаемые платформой ASP.NET AJAX, или определить свой собственный преобразователь. В состав пакета Futures входят следующие преобразователи:

`Sys.Preview.BindingBase.Transformers.Invert`

Преобразует значение `true` в `false`, а `false` в `true`

`Sys.Preview.BindingBase.Transformers.ToString`

Преобразует значение в строку аналогично методу `String.Format()`; допускается использование символов-заполнителей

`Sys.Preview.BindingBase.Transformers.Adds`

Добавляет к исходному значению некоторое другое значение

`Sys.Preview.BindingBase.Transformers.Multiply`

Умножает исходное значение на некоторое другое значение

`Sys.Preview.BindingBase.Transformers.Compare`

Сравнивает исходное значение с заданным и возвращает `true` в случае их равенства и `false` – в противном случае

`Sys.Preview.BindingBase.Transformers.CompareInverted`

Сравнивает исходное значение с заданным и возвращает `false` в случае их равенства и `true` – в противном случае

Некоторые из этих преобразователей принимают аргумент, значение которого может быть установлено с помощью метода `set_transformerArgument()` (например, строка формата для преобразователя `ToString`).

Программная привязка данных с использованием встроенных преобразователей

Вернемся к программному коду. Мы снова, с целью сэкономить силы и время, воспользуемся существующим примером из главы 14. Вернемся к примеру 14.7, где мы создавали текстовое поле и метку. Изменение текста в поле ввода приводило к изменению текста в элементе управления `Label`. Теперь мы попробуем связать эти два компонента с помощью механизма привязки данных. В первую очередь нам необходимы два HTML-элемента, такие как в следующем фрагменте:

```
<input type="text" id="TextBox1" /><br />
<label id="Label1"></label>
```

Затем нам необходим программный код, который будет создавать их экземпляры в JavaScript:

```
function pageLoad() {
  var textbox = new Sys.Preview.UI.TextBox($get("TextBox1"));
  var label = new Sys.Preview.UI.Label($get("Label1"));
```

Теперь можно выполнить привязку. Сначала создается экземпляр класса `Sys.Preview.Binding`:

```
var binding = new Sys.Preview.Binding();
```

Далее необходимо присоединить источник данных (контекст). В этом примере источником служит элемент управления `TextBox`:

```
binding.set_dataContext(textbox);
```

Поскольку наша цель – разместить текст внутри текстового поля ввода, путь к данным (свойство *Путь к данным*) – это `text`:

```
binding.set_dataPath("text");
```

Данные будут записываться в свойство `text` элемента управления `Label`:

```
binding.set_property("text");
```

Теперь перейдем к преобразователю. В данном примере будет использоваться преобразователь `ToString`:

```
binding.add_transform(Sys.Preview.BindingBase.Transformers.ToString);
```

По умолчанию входные данные передаются преобразователю в виде аргумента. Вы можете вставить в аргумент дополнительный текст, например метки, начальные и конечные предложения или даже информацию о форматировании:

```
binding.set_transformerArgument("Text entered: {0}");
```

На этом привязка данных почти закончена. Нам осталось лишь определить элемент-приемник с помощью метода `set_target()`:

```
binding.set_target(label);
```

А теперь самая хитрая (и заключительная) стадия. Необходимо инициализировать оба элемента управления и механизм привязки:

```
  textbox.initialize();
  label.initialize();
  binding.initialize()
}
```

Почему эта часть самая хитрая? Все дело в своевременности. Метод `initialize()` должен вызываться в самом конце, уже *после* того, как все необходимые объекты будут созданы и связаны. Если вызвать метод `initialize()` на ранних стадиях, привязка окажется неинициализированной и ничего происходить не будет. В предыдущей главе метод `initialize()` использовался для инициализации механизма обработки событий, где таких ограничений просто не существует. Метод может быть вызван в любом месте. Однако в данном случае очень важно, чтобы вызов метода `initialize()` производился в последнюю очередь. Порядок, в каком будут вызываться другие методы настройки привязки, не имеет такого значения. В предыдущей версии ASP.NET AJAX/Atlas не требовалось инициализировать механизм привязки, поэтому вы должны помнить об этой особенности при обновлении устаревшего программного кода.

Наконец надо не забыть загрузить файл *PreviewScript.js*, где содержится реализация механизма привязки:

```
<asp:ScriptManager runat="server">
  <Scripts>
    <asp:ScriptReference Name="PreviewScript.js"
```

```
                                Assembly="Microsoft.Web.Preview" />
    </Scripts>
  </asp:ScriptManager>
```

Полный программный код приводится в примере 15.1. При вводе текста в поле ничего происходить не будет. Но когда вы покинете поле ввода либо с помощью клавиши табуляции, либо щелчком мыши за пределами поля, произойдет событие `propertyChanged`, отработает механизм привязки и введенный текст появится в метке, как показано на рис. 15.1.

Пример 15.1. Использование механизма привязки данных ASP.NET AJAX с преобразователем

```
ControlBindingTextBox.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
  function pageLoad() {
    var textbox = new Sys.Preview.UI.TextBox($get("TextBox1"));
    var label = new Sys.Preview.UI.Label($get("Label1"));

    var binding = new Sys.Preview.Binding();
    binding.set_dataContext(textbox);
    binding.set_dataPath("text");
    binding.set_property("text");
    binding.add_transform(Sys.Preview.BindingBase.Transformers.ToString);
    binding.set_transformerArgument("Text entered: {0}");
    binding.set_target(label);

    textbox.initialize();
    label.initialize();
```

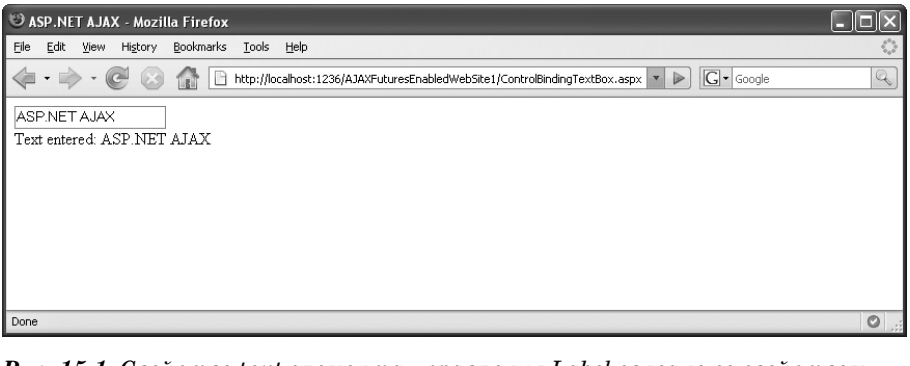


Рис. 15.1. *Свойство text элемента управления Label связано со свойством text элемента управления TextBox*

```
    binding.initialize();
  }
</script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="TextBox1" /><br />
      <label id="Label1"></label>
    </div>
  </form>
</body>
</html>
```

Направление привязки

По умолчанию привязка является «входящей», то есть данные копируются из источника в приемник. Представьте, что вы заменили элемент `Label` другим текстовым полем и настроили привязку, как и прежде. В таком случае данные будут копироваться из первого текстового поля во второе, но не наоборот. Такое поведение можно изменить, вызвав метод `set_direction()` объекта `Binding`. Допустимыми значениями аргумента этого метода являются следующие:

`Sys.Preview.BindingDirection.In`

Данные копируются из источника в приемник (значение по умолчанию)

`Sys.Preview.BindingDirection.Out`

Данные копируются из приемника в источник. Это особенно удобно при создании обратного эффекта в некоторых преобразователях

`Sys.Preview.BindingDirection.InOut`

При изменении значения свойства источника или приемника данные копируются в другой элемент управления

Следующий фрагмент создаст двунаправленную привязку:

```
binding.set_direction(Sys.Preview.BindingDirection.InOut);
```

Направление привязки имеет важное значение при использовании преобразователей `Add` или `Multiply`. Если используется направление привязки `Sys.Preview.BindingDirection.InOut`, ASP.NET AJAX будет выполнять преобразование в обратном направлении, интерпретируя преобразователь `Add` как операцию вычитания, а `Multiply` – деления.

Создание собственных преобразователей

Если встроенные преобразователи ASP.NET AJAX не удовлетворяют вашим потребностям, вы без труда сможете создавать свои собственные. Например, при копировании текста в элемент `Label` разметка HTML в текстовом поле из примера 15.1 не экранируется. Таким образом, если пользователь введет в текстовое поле код HTML, то разметка (например, `<b>Text</b>`) в элементе `Label` будет интерпретироваться как HTML (в данном случае текст будет выведен жирным шрифтом). Если в поле будет введен программный код JavaScript, то вместо того, чтобы вывести этот код на экран, броузер исполнит его.

Чтобы избежать этого, можно было бы написать свой собственный преобразователь, который будет преобразовывать управляющие символы HTML, такие как угловые скобки и кавычки, в соответствующие им сущности языка разметки HTML.

Заглянув в исходные тексты ASP.NET AJAX на JavaScript (точнее – в файл *Atlas.js*), вы сможете увидеть, как реализуются преобразователи. Функция преобразователя ожидает получить два аргумента. Первый – это объект-источник события, который обычно не используется. Второй аргумент содержит данные для преобразования:

```
function myTransformer(sender, args) {
  var value = args.get_value();
  ...
```

Результаты преобразования должны быть записаны обратно в свойство `value` аргумента с помощью метода `set_value()`:

```
  ...
  args.set_value(value);
}
```

Ниже приводится возможная реализация преобразователя, который экранирует разметку HTML с использованием регулярных выражений JavaScript. Модификатор `g` в конце выражения означает, что все символы амперсанда, угловые скобки и кавычки будут замещены их эквивалентами.

```
function customHtmlEncode(sender, args) {
  var value = args.get_value();
  var newValue = value.replace(/&/g, "&amp;")
                      .replace(/</g, "&lt;")
                      .replace(/>/g, "&gt;")
                      .replace(/"/g, "&quot;")
                      .replace(/'/g, "&apos;");
  args.set_value(newValue);
}
```

В заключение необходимо добавить эту функцию к привязке данных в качестве преобразователя. Делается это точно так же, как и в случае со встроенными преобразователями. В примере 15.2 приводится пол-

ный код страницы, в которой используется нестандартный преобразователь. Результаты работы страницы приводятся на рис. 15.2.

Пример 15.2. Использование собственного преобразователя

```
ControlBindingCustom.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
    function pageLoad() {
      var textbox = new Sys.Preview.UI.TextBox($get("TextBox1"));
      var label = new Sys.Preview.UI.Label($get("Label1"));

      var binding = new Sys.Preview.Binding();
      binding.set_dataContext(textbox);
      binding.set_dataPath("text");
      binding.set_property("text");
      binding.add_transform(customHtmlEncode);
      binding.set_target(label);

      textbox.initialize();
      label.initialize();
      binding.initialize();
    }

    function customHtmlEncode(sender, args) {
      var value = args.get_value();
      var newValue = value.replace(/&/g, "&amp;")
                          .replace(/</g, "&lt;")
                          .replace(/>/g, "&gt;")
                          .replace(/"/g, "&quot;")
                          .replace(/'/g, "&apos;");
      args.set_value(newValue);
    }
```

Рис. 15.2. *Разметка HTML теперь экранируется при копировании в метку*

```
  </script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="TextBox1" /><br />
      <label id="Label1"></label>
    </div>
  </form>
</body>
</html>
```

Использование разметки для привязки данных: xml–script

Программный подход к привязке данных прекрасно работает, но декларативный способ также имеет свои преимущества. Например, при декларативном подходе проблемы с вызовом метода `initialize()` (о которой говорилось в предыдущем разделе) просто не существует.

В предварительные версии ASP.NET AJAX компания Microsoft ввела *xml-script* – формат разметки специально для добавления функциональных возможностей в страницы ASP.NET AJAX. Группа разработчиков ASP.NET AJAX предполагала, что использование встроенной разметки XML прекрасно подойдет для представления дополнительной информации, которая сможет быть интерпретирована JavaScript во время исполнения на стороне клиента. Кроме того, такой подход дает разработчикам стандартную разметку, которая легко читается и даже может поддерживаться инструментальными средствами. Недостаток – отсутствие поддержки разметки xml-script механизмом IntelliSense в Visual Studio. (Дополнительную информацию об этом решении вы найдете в блоге *http://www.nikhilk.net/AtlasXMLScript.aspx*.) В какой-то момент компания Microsoft решила не продолжать дальнейшую работу над xml-script. Эта особенность по-прежнему присутствует в пакете Futures (и потому будет описана в этом разделе), но надежды на то, что она попадет в ядро ASP.NET AJAX Extensions, минимальны.

Основная схема xml-script выглядит примерно так:

```
<script type="text/xml-script">
  <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
    <components />
  </page>
</script>
```

В основе этой особенности лежит элемент разметки `<script>`, но платформа ASP.NET AJAX вводит для него специальный тип `text/xml-script`. Этот элемент используется для декларативного определения функциональности ASP.NET AJAX, в данном случае – для организации привязки данных. Внутри элемента `<script>` имеется элемент `<page>`, который используется для предоставления информации об элементах в странице и взаимосвязях между ними. Раздел `<components>` служит для декларативного создания экземпляров элементов управления ASP.NET AJAX, аналогичных тем, о которых говорилось выше при обсуждении программного подхода, для элементов в странице. Имена тегов, обеспечивающих поддержку тегов HTML, очень похожи на имена классов в пространстве имен `Sys.Preview.UI` за исключением того, что в именах тегов первое слово записывается символами нижнего регистра, а все последующие слова начинаются с символа верхнего регистра (например, `checkBox`). Ниже приводится список элементов, которые могут использоваться в разделе `<components>` для ссылки на элементы HTML:

`<control>`

Универсальный элемент, может использоваться для обозначения любого элемента управления

`<label>` или `<span>`

Текстовая метка

`<image>`

Изображение

`<hyperLink>`

Ссылка

`<button>`

Кнопка

`<checkBox>`

Флажок

`<selector>`

Список выбора

`<textBox>`

Текстовое поле

Для идентификации элемента в странице, которому соответствует тот или иной тег, необходимо устанавливать значение свойства `id` равным значению атрибута ID соответствующего элемента:

```
<label id="Label1" />
```

Привязка данных

Привязка данных выполняется с помощью элемента `<binding>`. Этот элемент должен объявляться как дочерний для связываемого элемен-

та управления, получающего данные. Внутри элемента <binding> можно устанавливать значения свойств, перечисленных в табл. 15.1. Все они должны быть знакомы вам по примерам, приводившимся ранее в этой главе.

Таблица 15.1. Свойства элемента <binding>

Свойство	Описание
dataContext	Элемент с данными для привязки
dataPath	Свойство, которое служит источником данных
property	Свойство, которое служит приемником данных
transformerArgument	Аргумент для преобразователя
transform	Используемый преобразователь
direction	Направление привязки

Назначение каждого из этих свойств вполне очевидно. Одно из удобств, которое вы наверняка оцените по достоинству, заключается в том, что вам не нужно приводить полное имя пространства имен для обозначения преобразователей и направления привязки. Например, вместо Sys.Preview.Binding.Transformers.ToString можно просто указать ToString или вместо Sys.Preview.BindingDirection.InOut использовать обозначение InOut.

С помощью разметки xml-script можно реализовать привязку данных без единой строчки программного кода. Один немаловажный факт: каждая ссылка на HTML-элемент в xml-script эквивалентна вызову метода initialize() этого элемента. Говоря другими словами, на любой элемент управления, который требуется инициализировать, должна быть ссылка в xml-script. Отсюда следует, что в разметке вы должны указать еще и ссылку на текстовое поле, хотя привязка относится к элементу <label>. Пример 15.3 демонстрирует, как это делается.

Пример 15.3. Использование механизма привязки данных ASP.NET AJAX через xml-script

```
ControlBindingDeclarative.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server">
      <Scripts>
```

```
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="TextBox1" /><br />
      <label id="Label1"></label>
    </div>
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <textBox id="TextBox1" />
        <label id="Label1">
           <bindings>
              <binding dataContext="TextBox1"
                   dataPath="text"
                   property="text"
                   transform="ToString"
                   transformerArgument="Text entered: {0}" />
          </bindings>
        </label>
      </components>
    </page>
  </script>
</body>
</html>
```

Если вы будете использовать собственный преобразователь, вам потребуется написать программный код, но только для самой функции преобразования. После этого вы просто указываете имя функции преобразования в атрибуте `transform`, и эта функция автоматически вызывается при связывании данных.

Обработка событий

В главе 14 рассматривалась обработка событий в клиентских элементах управления ASP.NET AJAX. С помощью xml-script можно настроить обработку событий полностью декларативным способом.

Как и в случае с привязкой данных, все необходимое должно объявляться в разделе `<components>` блока xml-script. Каждый элемент события (например, `click`) поддерживает следующие три дочерних элемента:

`<setPropertyAction>`

Определяет свойства элемента

`<invokeMethodAction>`

Вызов метода

`<button click="someFunction">`

Декларативно добавляет обработчик события

Начнем с элемента `<setPropertyAction>`. Мы воспользуемся немного видоизмененной версией примера 14.8, в которой динамически изменялись классы CSS. На этот раз изменение класса CSS будет производиться за счет установки свойства `class` требуемого элемента.

Тег `<setPropertyAction>` поддерживает следующие атрибуты:

`target`

Элемент, к которому осуществляется доступ

`property`

Имя изменяемого свойства

`propertyKey`

В этом свойстве используется «точечная» нотация (`.`) при использовании подсвойств, таких как `style.borderStyle`, при этом атрибут `property` должен иметь значение `"element"`.

`value`

Новое значение

Например, нам нужно реализовать действие по щелчку на кнопке. Для этого нам нужно перехватить событие `<click>`. Следующий фрагмент изменяет стиль рамки метки по нажатию кнопки `Button1`.

```
<label id="Label1" />
<button id="Button1">
  <click>
  <setProperty target="Label1"
               property="element"
               propertyKey="style.borderStyle"
               value="dotted" />
  </click>
</button>
```

В результате мы пришли к разметке, которая показана в примере 15.4, где определены две кнопки, каждая из которых имеет свое собственное определение `setPropertyAction`. Результат работы этой страницы показан на рис. 15.3.

Пример 15.4. Изменение значений свойств через xml-script

```
ControlDeclarativeProperty.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
```

```
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <label id="Label1">This text will be reformatted</label>
    </div>
    <input type="button" id="Button1" value="Solid" />
    <input type="button" id="Button2" value="Dotted" />
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <label id="Label1" />
        <button id="Button1">
          <click>
            <setPropertyAction target="Label1"
                               property="element"
                               propertyKey="style.borderStyle"
                               value="solid" />
          </click>
        </button>
        <button id="Button2">
          <click>
            <setPropertyAction target="Label1"
                               property="element"
                               propertyKey="style.borderStyle"
                               value="dotted" />
          </click>
        </button>
      </components>
    </page>
  </script>
</body>
</html>
```

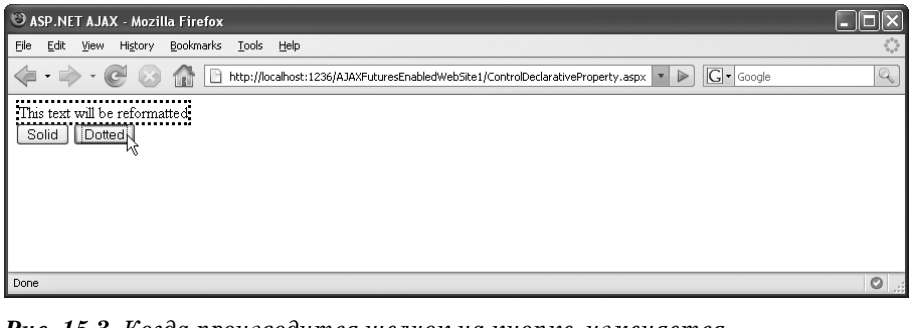


Рис. 15.3. *Когда производится щелчок на кнопке, изменяется класс CSS текста*

Вызов методов

Возможность изменять значение свойства – это удобно, но обязательно должна присутствовать и возможность вызывать методы при появлении событий. Как и следовало ожидать, такая возможность также обеспечивается с помощью xml-script. Для этого необходимо определить два элемента:

- Элемент `<invokeMethodAction>`
- Элемент `<parameters>`

Элемент `<invokeMethodAction>` поддерживает следующие атрибуты:

`method`

Определяет имя вызываемого метода

`target`

Определяет имя объекта, которому принадлежит вызываемый метод

В данном случае вы не ограничены встроенными функциональными возможностями. Например, можно использовать определение `<invokeMethodAction>` для обращения к методу веб-службы. Попробуем создать простую веб-службу, которая будет возвращать два значения: `style1` или `style2`. Программный код веб-службы приводится в примере 15.5.

Пример 15.5. Веб-служба, случайным образом возвращающая имя класса CSS

```
RandomCssClass.asmx

<%@ WebService Language="C#" Class="RandomCssClass" %>

using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.Web.Script.Services.ScriptService]
public class RandomCssClass : System.Web.Services.WebService {

  [WebMethod]
  public string getRandomCssClass() {
    Random r = new Random();
    return "style" + r.Next(1, 3);
  }
}
```

Далее мы будем использовать эту веб-службу для форматирования элемента в соответствии с классом CSS, получаемым от веб-службы. Ниже приводится определение двух классов CSS:

```
<style type="text/css">
  .style1 { font-family: Monospace; border-style: dotted; color: #0f0; }
  .style2 { font-family: Sans-Serif; border-style: solid; color: #0ff; }
</style>
```

На первом шаге в xml-script необходимо определить веб-службу, чтобы платформа ASP.NET AJAX могла инициализировать ее. Для этого мы используем элемент `<serviceMethodRequest>`, который имеет следующие атрибуты:

`id`

Идентификатор, который используется для ссылки на заданный метод веб-службы из других частей xml-script

`url`

Адрес URL веб-службы

`methodName`

Имя метода веб-службы

`useGet`

Определяет, следует ли использовать метод HTTP GET для обращения к веб-службе. Устанавливайте этот атрибут в значение `false`, поскольку по умолчанию использование метода HTTP GET запрещено.

```
<serviceMethodRequest id="randomCssMethod"
                      url="RandomCssClass.asmx"
                      methodName="getRandomCssClass"
                      useGet="false">
```

Несмотря на то, что кнопки порождают событие, такое как `<click>`, обработку результатов вызова метода веб-службы можно реализовать в элементе `<completed>`, который отрабатывает после того, как будут получены данные от веб-службы. В процессе обработки этого события нам нужно изменить класс CSS метки в соответствии с полученными результатами. Эта работа будет выполняться элементом `<setPropertyAction>`:

```
<completed>
  <setPropertyAction target="Label1"
                     property="element"
                     propertyKey="className">
```

Последний шаг крайне важен. Для записи данных, полученных от веб-службы, в свойство `className` метки мы снова воспользуемся механизмом привязки. В предыдущем примере мы записывали в свойство заранее определенные значения. А в следующем примере мы будем записывать значения, полученные динамически, – результат вызова метода. Для этого нам и необходим механизм привязки данных.

Определить значения атрибутов `dataContext` (`"randomCssMethod"`, идентификатор элемента `<serviceMethodRequest>`) и `property` (`"value"`) не составляет труда, но значение атрибута `dataPath` (`result`) еще нужно поискать (это зарезервированное слово). В результате мы получаем следующую разметку:

```
<bindings>
  <binding dataContext="randomCssMethod"
           dataPath="result"
```

```
                    property="value" />
        </bindings>
      </setPropertyAction>
    </completed>
  </serviceMethodRequest>
```

Наконец, необходимо реализовать вызов метода по щелчку на кнопке. По умолчанию для передачи дополнительных данных при обращении к веб-службам поддерживается свойство `userContext` (глава 5). Все параметры определяются внутри элемента `<parameters>` в виде атрибутов, поэтому передать один или более параметров будет очень просто. Эти атрибуты определяются как пары `parametername=parametervalue`, вследствие чего отпадает необходимость использовать подэлементы.

Разметка, показанная в следующем фрагменте, вызывает метод веб-службы и передает ему пустой пользовательский контекст (это значение по умолчанию, поэтому атрибут `userContext` можно было бы опустить).

```
<button id="Button1">
  <click>
    <invokeMethodAction target="randomCssMethod" method="invoke">
      <parameters userContext="" />
    </invokeMethodAction>
  </click>
</button>
```

Нам пришлось приложить немало усилий, а кроме того, такой подход увеличивает вероятность появления ошибок. Например, обнаружение синтаксических ошибок в разметке xml-script окажется нелегким делом, – сценарий просто не будет работать, но кроме этого вы не получите никаких подсказок. Поэтому вам следует рассмотреть возможность использования средств проверки разметки XML для проверки кода xml-script. В примере 15.6 приводится полный код страницы, а на рис. 15.4 – результат ее отображения в броузере.

Пример 15.6. Вызов методов веб-служб и привязка данных с использованием xml-script

ControlDeclarativeMethod.aspx

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
  <style type="text/css">
    .style1 { font-family: Monospace; border-style: dotted; color: #0f0; }
    .style2 { font-family: Sans-Serif; border-style: solid; color: #0ff; }
  </style>
</head>
<body>
  <form id="form1" runat="server">
```

```
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Services>
        <asp:ServiceReference Path="RandomCssClass.asmx" />
      </Services>
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <label id="Label1">This text will be reformatted</label>
    </div>
    <input type="button" id="Button1" value="Random style" />
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <label id="Label1" />
        <button id="Button1">
        <click>
          <invokeMethodAction target="randomCssMethod" method="invoke">
            <parameters userContext="" />
          </invokeMethodAction>
        </click>
      </button>
      <serviceMethodRequest id="randomCssMethod"
                            url="RandomCssClass.asmx"
                            methodName="getRandomCssClass"
                            useGet="false">
        <completed>
          <setPropertyAction target="Label1"
                             property="element"
                             propertyKey="className">
            <bindings>
              <binding dataContext="randomCssMethod"
                       dataPath="result"
                       property="value" />
            </bindings>
```

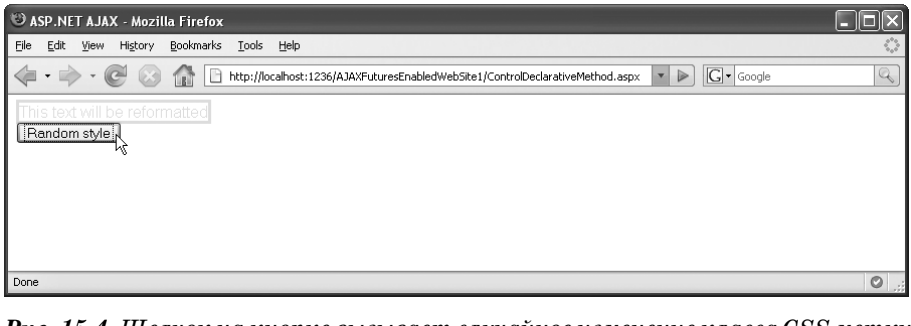


Рис. 15.4. *Щелчок на кнопке вызывает случайное изменение класса CSS метки*

```
            </setPropertyAction>
          </completed>
        </serviceMethodRequest>
      </components>
    </page>
  </script>
  </body>
  </html>
```

Несмотря на то, что ссылка на веб-службу уже создана в xml-script, вам по-прежнему необходимо создавать объект-посредник на JavaScript с помощью элемента управления `ScriptManager`, для чего используются элементы `<Services>` и `<asp:ServiceReference>`.

Проверка данных

В дополнение к элементам управления, реализующим механизм привязки данных, в пакете ASP.NET AJAX Futures содержатся свои собственные клиентские элементы управления, предназначенные для выполнения проверки данных, введенных пользователем, – особенность, которую многие разработчики на ASP.NET считают весьма полезной. Платформа ASP.NET AJAX поддерживает следующие валидаторы:

`requiredFieldValidator`

Проверяет, было ли что-то введено пользователем в элемент управления

`regexValidator`

Проверяет данные в элементе управления на соответствие регулярному выражению

`typeValidator`

Проверяет, соответствуют ли данные в элементе управления требуемому типу данных

`rangeValidator`

Проверяет данные в элементе управления на соответствие заданному диапазону значений

`customValidator`

Проверяет данные в элементе управления с использованием нестандартной функции

Если вы работаете с элементом управления `UpdatePanel`, то для проверки данных, введенных в серверные элементы управления внутри панели, можно использовать серверные валидаторы ASP.NET. Однако вам необходимо обновить версии этих компонентов, чтобы они были совместимы с ASP.NET AJAX. Дополнительные сведения вы найдете в блоге Мэтта Гиббса (Matt Gibbs) по адресу: *http://blogs.msdn.com/mattgi/archive/2007/05/12/validators-update-available.aspx*.

Для реализации проверки данных необходимы:

- Проверяемый элемент управления
- Способ отображения сообщений об ошибках в случае, когда проверка завершается неудачей
- Программный код или разметка, которые будут выполнять проверку

В следующих разделах будет показано, как можно использовать каждый из валидаторов ASP.NET AJAX, а также рассмотрено создание своего собственного валидатора.

Проверка обязательных полей

Наиболее часто используется класс `requiredFieldValidator`, который проверяет, чтобы элемент управления содержал какие-либо данные. Ниже приводится разметка, которая создает поле ввода и область вывода сообщения об ошибке, создаваемого валидатором:

```
<input type="text" id="TextBox1" />
<span id="Error1" style="color: red;">*</span>
```

Как видите, метка, куда выводятся сообщения об ошибках, по умолчанию не скрыта. Платформа ASP.NET AJAX сама позаботится о том, чтобы ее скрыть.

Добавим в xml-script разметку с описанием элементов управления, принимающих участие в процессе проверки корректности данных, ограничившись проверкой элементов ввода и оставив за пределами рассмотрения другие элементы управления, выдающие сообщения об ошибках. В подэлементе `<validators>`, отвечающем за ввод, указываем используемый валидатор. В свойство `errorMessage` помещаем текст, отображаемый в случае ошибки. Валидаторы ASP.NET AJAX несколько отличаются от своих сородичей в ASP.NET. В ASP.NET AJAX значение свойства `errorMessage` выводится в виде всплывающей подсказки, которая появляется при наведении указателя мыши на неверный текст (то есть на элемент-валидатор ASP.NET AJAX).

Хотя мы и говорим о контроле текста, тем не менее это не эквивалент свойства `Text`, которое имеется у элементов-валидаторов в ASP.NET. Текст в метке с сообщением об ошибке – это текст, который задан заранее. Следующий пример демонстрирует xml-script, определяющий валидатор проверки обязательных для заполнения полей, связанный с элементом управления `TextBox`:

```
<textBox id="TextBox1">
  <validators>
    <requiredFieldValidator errorMessage="** TextBox1 value missing" />
  </validators>
</textBox>
```

На следующем шаге требуется задействовать элемент `<validationErrorLabel>`. Этот элемент имеет следующие атрибуты:

id

Идентификатор элемента управления, отображающего сообщения об ошибках

associatedControl

Идентификатор проверяемого элемента

Полный код страницы приводится в примере 15.7.

Пример 15.7. Использование валидатора для проверки полей, обязательных к заполнению

ControlValidationRequiredField.aspx

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="TextBox1" />
      <span id="Error1" style="color: red;">*</span>
      <br />
      <input type="submit" />
    </div>
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <textBox id="TextBox1">
          <validators>
            <requiredFieldValidator
              errorMessage="** TextBox1 value missing" />
          </validators>
        </textBox>
        <validationErrorLabel id="Error1" associatedControl="TextBox1" />
      </components>
    </page>
  </script>
</body>
</html>
```

Рис. 15.5. *Признак наличия ошибки и всплывающая подсказка, содержащая более информативное описание*

Загрузите страницу, введите какие-нибудь данные в текстовое поле и затем покиньте его (что приведет к появлению события `change` – не надо щелкать на кнопке Submit Query). Теперь войдите в поле еще раз, удалите все данные и покиньте поле. Событие `change` произойдет еще раз и запустит проверку, предусмотренную элементом-валидатором. Как показано на рис. 15.5, ошибка индицируется появлением всплывающей подсказки и метки с более подробным поясняющим текстом.

В данном случае валидатор генерирует ошибку, если пользователь не ввел никаких данных, но тем не менее может отправить форму на сервер. Валидатор имеет лишь информационный характер. Позднее в этой же главе будет продемонстрировано, как предотвратить отправку формы в случае, если проверка потерпела неудачу.

Проверка на соответствие регулярному выражению

Использование регулярных выражений для проверки данных функционально напоминает работу с элементом управления ASP.NET `RegularExpressionValidation`, хотя имя XML-элемента и его атрибуты отличаются. Свойство `regex` (или атрибут, в зависимости от того, какой подход вы используете – программный или декларативный) содержит регулярное выражение, на соответствие которому выполняется проверка данных:

```
<regexValidator regex="/\d*/" errorMessage="** digits only" />
```

Код страницы, который приводится в примере 15.8, содержит два валидатора: один проверяет, содержится ли *хоть что-нибудь* в текстовом поле, а второй допускает наличие в поле ввода только цифровых символов. Тот же самый эффект можно было бы получить с помощью валидатора, проверяющего тип данных, однако здесь я хочу продемонстрировать использование элемента управления `regexValidator`.

Пример 15.8. Использование валидатора для проверки полей на соответствие регулярному выражению

```
ControlValidationRegex.aspx

<%@ Page Language="C#" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="TextBox1" />
      <span id="Error1" style="color: red;">*</span>
      <br />
      <input type="submit" />
    </div>
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <textBox id="TextBox1">
          <validators>
            <requiredFieldValidator
              errorMessage="** TextBox1 value missing" />
            <regexValidator regex="/\d*/" errorMessage="** digits only" />
          </validators>
        </textBox>
        <validationErrorLabel id="Error1" associatedControl="TextBox1" />
      </components>
    </page>
  </script>
</body>
</html>
```

Проверка типа данных

Проверка типа данных выполняется с помощью элемента `<typeValidator>`. В настоящее время поддерживается только тип данных `Number`, однако в будущих версиях будут добавлены и другие типы данных. Проверяемый тип данных определяется свойством `type` элемента `<typeValidator>`:

```
<typeValidator type="Number" errorMessage="** numbers only" />
```

В странице, код которой приводится в примере 15.9, используются валидаторы `requiredFieldValidator` и `typeValidator`, которые проверяют обязательное наличие числового значения.

Пример 15.9. Использование валидатора для проверки типа данных

```
ControlValidationType.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="TextBox1" />
      <span id="Error1" style="color: red;">*</span>
      <br />
      <input type="submit" />
    </div>
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <textBox id="TextBox1">
          <validators>
            <requiredFieldValidator
              errorMessage="** TextBox1 value missing" />
              <typeValidator type="Number" errorMessage="** numbers only" />
          </validators>
        </textBox>
        <validationErrorLabel id="Error1"
                              associatedControl="TextBox1" />
      </components>
    </page>
  </script>
</body>
</html>
```

Проверка принадлежности диапазону

Иногда бывает необходимо убедиться, что введенное значение не просто является числом, но и принадлежит определенному диапазону (например, некоторому интервалу дат или времени). Для решения подобных задач можно использовать элемент `<rangeValidator>`. Нижняя и верхняя границы диапазона устанавливаются с помощью свойств `lowerBound`

и upperBound. Следующая разметка демонстрирует проверку на принадлежность диапазону чисел от 1 до 6:

```
<rangeValidator lowerBound="1" upperBound="6" errorMessage="** 1 to 6 only" />
```

Пример 15.10 построен на основе предыдущего примера. Данные проверяются не только с помощью валидаторов requiredFieldValidator и typeValidator, но и с помощью rangeValidator.

Пример 15.10. Использование валидатора для проверки на принадлежность диапазону значений

```
ControlValidationRange.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="TextBox1" />
      <span id="Error1" style="color: red;">*</span>
      <br />
      <input type="submit" />
    </div>
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <textBox id="TextBox1">
          <validators>
            <requiredFieldValidator
              errorMessage="** TextBox1 value missing" />
            <typeValidator type="Number" errorMessage="** numbers only" />
            <rangeValidator lowerBound="1" upperBound="6"
                            errorMessage="** 1 to 6 only" />
          </validators>
        </textBox>
        <validationErrorLabel id="Error1"
                              associatedControl="TextBox1" />
      </components>
    </page>
```

```
  </script>
</body>
</html>
```

Собственные валидаторы

Для достижения большей гибкости приложения поддерживается возможность создания своих собственных валидаторов для проверки пользовательских данных. Сигнатура функции-валидатора имеет следующий вид:

```
function <name>(sender, args) {}
```

Первый аргумент функции – это элемент, вызвавший операцию проверки, но более важное значение имеет второй аргумент, где передается значение для проверки, доступ к которому можно получить с помощью метода `get_value()`. После проверки необходимо вызвать метод `set_isValue()`, которому в случае успеха передается значение `true`, а в случае неудачи – значение `false`.

Исключительно в демонстрационных целях рассмотрим случай, когда по некоторым причинам в текстовое поле допускается вводить только квадраты целых чисел. Данную проверку выполняет следующая функция:

```
function validateSquare(sender, args) {
  var value = args.get_value();
  args.set_isValid(Math.sqrt(value) == Math.floor(Math.sqrt(value)));
}
```

В xml-script элемент `<customValidator>` должен содержать атрибут `validateValue`, который будет ссылаться на функцию-валидатор:

Режим отображения элементов-валидаторов

До сих пор мы еще не рассматривали свойство `visibilityMode` элементов-валидаторов, определяющее режим отображения. Если быть более точным, это свойство элемента `<validationErrorLabel>`, которое может принимать одно из двух значений (определяются в виде перечисления `Sys.UI.VisibilityMode`):

- `Collapse`
- `Hide`

Этот режим отображения определяется с помощью стиля `display` (или, в JavaScript, *`element`*`.style.display`). Если не задан ни один из режимов, используется значение `"none"` (что делает сообщение об ошибке невидимым и освобождает занимаемое им место на странице для использования другими элементами). Это свойство определяет, как после загрузки страницы будет скрыт элемент управления `Label`, содержащий текст сообщения об ошибке.

```
<customValidator validateValue="validateSquare"
                 errormessage="** square numbers only" />
```

В примере 15.11 приводится полный программный код этой нестандартной функции-валидатора.

Пример 15.11. Использование собственного валидатора

```
ControlValidationCustom.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script type="text/javascript">
  function validateSquare(sender, args) {
    var value = args.get_value();
    args.set_isValid(Math.sqrt(value) == Math.floor(Math.sqrt(value)));
  }
  </script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="TextBox1" />
      <span id="Error1" style="color: red;">*</span>
      <br />
      <input type="submit" />
    </div>
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <textBox id="TextBox1">
          <validators>
            <requiredFieldValidator
              errorMessage="** TextBox1 value missing" />
            <typeValidator type="Number" errorMessage="** numbers only" />
            <customValidator validateValue="validateSquare"
                             errorMessage="** square numbers only" />
          </validators>
        </textBox>
        <validationErrorLabel id="Error1" associatedControl="TextBox1" />
      </components>
```

```
    </page>
  </script>
</body>
</html>
```

Программная проверка

Декларативный подход прекрасно зарекомендовал себя на практике, однако существует и программный подход к проверке данных (который добавляет проверку во время исполнения).

Впрочем, и при таком подходе по-прежнему требуется выполнить некоторые объявления, такие как показаны в следующем фрагменте:

```
<textBox id="TextBox1">
</textBox>
<validationErrorLabel id="Error1"
                      associatedControl="TextBox1" />
```

Валидатор можно создать с помощью программного кода JavaScript. Для этого необходимо пройти два этапа:

1. Добавить валидатор: *element*.get_validators().add(*validator*).
2. Если вы предполагаете использовать функцию обратного вызова (которая вызывается, когда возникает потребность в выполнении проверки), используется синтаксис: *element*.add_validated(*function*).

Невозможно сделать доступным для проверки элемент с помощью обычной операции new Sys.Preview.UI.*XXX*. Вместо этого необходимо использовать несколько непривычный синтаксис, который будет представлен в ходе обсуждения реализации предотвращения отправки формы:

```
var textbox = $get("TextBox1").control;
```

Сначала мы отыскиваем элемент управления (на стороне клиента) с помощью функции $get(), а затем получаем ссылку на него с помощью свойства control. В примере 15.12 приводится полный код страницы, использующей программный подход к реализации проверки. В этом примере выполняется проверка обязательного к заполнению поля, которая приводилась в примере 15.7, но на этот раз валидатор добавляется к текстовому полю из программного кода JavaScript.

Пример 15.12. Программный подход к построению собственного валидатора

```
ControlValidationCustomProgrammatic.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
```

```
    <script type="text/javascript">
      function pageLoad() {
        var textbox = $get("TextBox1").control;
        validator = new Sys.Preview.UI.RequiredFieldValidator();
        validator.set_errorMessage("** enter some data");
        textbox.get_validators().add(validator);
        textbox.add_validated(validationComplete);
      }

      function validationComplete(sender, args) {
      }
    </script>
  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server">
        <Scripts>
          <asp:ScriptReference Name="PreviewScript.js"
                               Assembly="Microsoft.Web.Preview" />
        </Scripts>
      </asp:ScriptManager>
      <div>
        <input type="text" id="TextBox1" />
        <span id="Error1" style="color: red;">*</span>
        <br />
        <input type="submit" />
      </div>
    </form>
    <script type="text/xml-script">
      <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
        <components>
          <textBox id="TextBox1">
          </textBox>
          <validationErrorLabel id="Error1"
                                associatedControl="TextBox1" />
        </components>
      </page>
    </script>
  </body>
  </html>
```

Этот подход можно использовать и для реализации более сложных валидаторов, включая свои собственные валидаторы. Синтаксис объявления собственной функции-валидатора имеет следующий вид:

```
validator.add_validateValue(validation function);
```

В примере 15.13 демонстрируется, как можно добавлять валидаторы к полю ввода декларативным и программным способами. Валидаторы, проверяющие наличие данных в поле и тип данных, добавляются декларативно, а авторский валидатор – программно. Результат получается тот же самый, что и в предыдущих примерах.

Пример 15.13. Добавление возможности проверки декларативным и программным способами

ControlValidationRequiredFieldProgrammatic.aspx

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
  <script language="JavaScript" type="text/javascript">
  function validateSquare(sender, args) {
    var value = args.get_value();
    args.set_isValid(Math.sqrt(value) == Math.floor(Math.sqrt(value)));
  }
  function pageLoad() {
    var textbox = $get("TextBox1").control;
    validator = new Sys.Preview.UI.CustomValidator();
    validator.set_errorMessage("Square numbers only");
    validator.add_validateValue(validateSquare);
    textbox.get_validators().add(validator);
    textbox.add_validated(validationComplete);
  }
  function validationComplete(sender, args) {
  }
  </script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="TextBox1" />
      <span id="Error1" style="color: red;">*</span>
      <br />
      <input type="submit" />
    </div>
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <textBox id="TextBox1">
          <validators>
            <requiredFieldValidator
              errorMessage="** TextBox1 value missing" />
            <typeValidator type="Number" errorMessage="** numbers only" />
```

```
          </validators>
        </textBox>
        <validationErrorLabel id="Error1"
                              associatedControl="TextBox1" />
      </components>
    </page>
  </script>
</body>
</html>
```

Групповая проверка

Элементы проверки могут быть сгруппированы вместе, что позволяет выполнять проверку нескольких элементов управления как единого целого. Группы проверяемых компонентов создаются с помощью элемента `<validationGroup>`. Каждый валидатор в группе выполняет свою проверку независимо от других, но вся группа проходит тестирование как один объект. Если какая-то из проверок в группе терпит неудачу, то неудачу терпит и вся группа. Естественно, с другой стороны, что если все проверки в группе выполнены успешно, то и вся группа считается успешно прошедшей проверку. Группировка особенно полезна, когда по некоторому условию бывает необходимо активировать и деактивировать отдельные наборы валидаторов. В следующем примере демонстрируется, как можно управлять видимостью элементов в случае успешного выполнения проверок во всех условных валидаторах.

Группа проверки имеет метод `isValid()`, который определяет успех или провал проверки. Он может использоваться в соединении с механизмом привязки данных для отображения сообщения, текст которого зависит от успешного или неудачного прохождения всех проверок в группе.

Прежде всего, необходимо создать элемент для отображения сообщения:

```
<div id="Errors">-no errors-</div>
```

Затем нужно связать свойство `visible` этого элемента с методом группы `isValid()`. Если все валидаторы в группе успешно выполняют проверку, элемент `<div>` становится видимым.

```
<label id="Errors">
  <bindings>
    <binding dataContext="group" dataPath="isValid" property="visible" />
  </bindings>
</label>
```

Чтобы сделать элемент `<div>` видимым в случае *неудачной* проверки, используется преобразователь `Invert`:

```
<binding dataContext="group" dataPath="isValid" property="visible"
         transform="Invert" />
```

Теперь, помимо самих валидаторов, у нас отсутствует только одно – собственно группа проверяемых компонентов. Группу представляет элемент <validationGroup>. Ей необходимо присвоить идентификатор (предшествующая разметка использует в качестве идентификатора группы значение "group"), а внутри группы определить все элементы формы, которые будут принимать участие в проверке, как показано ниже:

```
<validationGroup id="group" >
  <associatedControls>
    <reference component="TextBox1" />
    <reference component="TextBox2" />
  </associatedControls>
</validationGroup>
```

В примере 15.14 приводится код страницы, в которой выполняется групповая проверка. В странице присутствует элемент с текстом -no errors-, который отображается в случае, если все текстовые поля успешно прошли проверку. Для первого текстового поля определен валидатор, проверяющий обязательное заполнение поля, то есть элемент будет отображаться, только если в это поле были введены какие-либо данные. Второе текстовое поле требует, чтобы введенные данные представляли собой число, являющееся квадратом какого-либо целого числа. Результат работы страницы показан на рис. 15.6.

Пример 15.14. Использование групповой проверки на принадлежность диапазону, связанной с меткой

```
CustomValidationGroup.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
  <script language="JavaScript" type="text/javascript">
    function validateSquare(sender, args) {
      var value = args.get_value();
      args.set_isValid(Math.sqrt(value) == Math.floor(Math.sqrt(value)));
    }
  </script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
```

```
      Anything: <input type="text" id="TextBox1" />
      <span id="Error1" style="color: red;">*</span>
      <br />
      A square: <input type="text" id="TextBox2" />
      <span id="Error2" style="color: red;">*</span>
      <br />
      <input type="submit" />
    </div>
    <div id="Errors">-no errors-</div>
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <textBox id="TextBox1">
          <validators>
            <requiredFieldValidator
              errorMessage="** TextBox1 value missing" />
          </validators>
        </textBox>
        <validationErrorLabel id="Error1"
                              associatedControl="TextBox1" />
        <textBox id="TextBox2">
          <validators>
            <requiredFieldValidator
              errorMessage="** TextBox2 value missing" />
            <typeValidator type="Number" errorMessage="** numbers only" />
            <customValidator validateValue="validateSquare"
                             errorMessage="** square numbers only" />
          </validators>
        </textBox>
        <validationErrorLabel id="Error2"
                              associatedControl="TextBox2" />
        <validationGroup id="group">
          <associatedControls>
            <reference component="TextBox1" />
            <reference component="TextBox2" />
          </associatedControls>
```

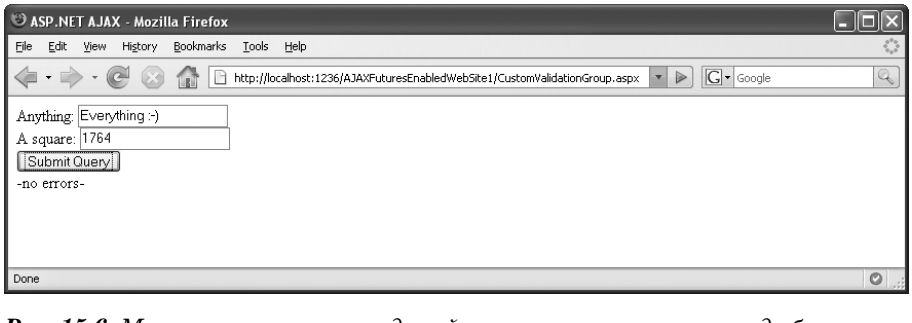


Рис. 15.6. *Метка становится видимой, только если все поля ввода были заполнены без ошибок*

```
        </validationGroup>
        <label id="Errors">
          <bindings>
            <binding dataContext="group" dataPath="isValid"
                     property="visible" />
          </bindings>
        </label>
      </components>
    </page>
  </script>
</body>
</html>
```

Предотвращение возможности отправки формы

Элементы управления, выполняющие проверку и входящие в состав пакета ASP.NET AJAX Futures, очень удобны, но они не включают в себя механизм отправки формы. Поэтому, даже если проверка терпит неудачу, пользователь все равно сможет отправить форму. Обычно это не представляет большой проблемы, так как все данные в форме должны пройти повторную проверку на сервере (поддержка JavaScript может быть отключена в броузере, поэтому вы никогда не должны полагаться на результаты проверки на стороне клиента). Однако с точки зрения удобства использования формы будет лучше, если отправить ее можно будет, только когда все поля заполнены корректными данными.

Добавление такой возможности требует написания некоторого объема программного кода, но это не составит особого труда. Мы возьмем за основу пример 15.7 (проверка обязательного заполнения поля) и добавим в него возможность предотвращения отправки формы. Вся хитрость заключается в том, чтобы перехватить событие, которое запускает тег `<form>`. Благодаря системе обработки событий мы можем запускать программный код JavaScript в момент отправки формы (событие `submit`). Если этот код вернет значение `false`, отправка формы будет прервана. Ниже показано, как следует изменить тег `<form>`:

```
<form id="form1" runat="server" onsubmit="return validateForm();">
```

Метод `validateForm()` должен возвращать значение `false`, если форма не готова к отправке, в противном случае он должен возвращать `true`. Это легко реализуется с помощью вспомогательного метода `get_isInvalid()`, который определяется как метод элемента управления текстового поля (используется свойство `control` ассоциированного элемента HTML):

```
function validateForm() {
  var textbox = $get("TextBox1").control;
  return !(textbox.get_isInvalid());
}
```

Полный исходный текст приводится в примере 15.15.

Пример 15.15. Использование групповой проверки на принадлежность диапазону, связанной с меткой

```
CustomValidationGroup.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
  <script type="text/javascript">
  function validateForm() {
    var textbox = $get("TextBox1").control;
    return !(textbox.get_isInvalid());
  }
  </script>
</head>
<body>
  <form id="form1" runat="server" onsubmit="return validateForm();">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="TextBox1" />
      <span id="Error1" style="color: red;">*</span>
      <br />
      <input type="submit" />
    </div>
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <textBox id="TextBox1">
          <validators>
            <requiredFieldValidator
              errorMessage="** TextBox1 value missing" />
          </validators>
        </textBox>
        <validationErrorLabel id="Error1" associatedControl="TextBox1" />
      </components>
    </page>
  </script>
</body>
</html>
```

Добавить в этот код большее число элементов формы не составит никакого труда – функция `validateForm()` в этом случае должна просто возвращать нечто вроде:

```
!(formelement1.get_isInvalid() || formelement2.get_isInvalid() || ...)
```

Подведение итогов

В этой главе мы познакомились с механизмом привязки данных и элементами проверки правильности данных, которые входят в состав пакета ASP.NET AJAX Futures. В сравнении с элементами проверки в ASP.NET они испытывают определенный недостаток функциональных возможностей. Кроме того, использование элементов управления ASP.NET AJAX выглядит немного необычно по сравнению с их серверными аналогами. Они не интегрированы в механизм отправки форм броузера, а это означает, что пользователь имеет возможность отправить форму, даже если в ней присутствуют ошибочные данные. И наконец, последнее и самое важное – элементы проверки ASP.NET AJAX работают только на стороне клиента и только при наличии поддержки JavaScript. Элементы проверки ASP.NET работают как на стороне клиента, так и на стороне сервера, и потому проверку, построенную на их основе, нельзя обойти, просто отключив поддержку JavaScript. Это делает их использование более предпочтительным, если у вас есть возможность выбирать. Однако следует отметить, что если веб-сайт использует платформу ASP.NET AJAX для организации всех эффектов на стороне клиента, то валидаторы ASP.NET AJAX прекрасно интегрируются с другими особенностями ASP.NET AJAX.

Для дополнительного чтения

http://blogs.msdn.com/mattgi/archive/2007/05/12/validators-update-available.aspx

Страница в блоге Мэтта Гиббса (Matt Gibbs); содержит информацию об обновленных версиях валидаторов, которые также работают внутри `UpdatePanel`

http://quickstarts.asp.net/Futures/ajax/doc/bindings.aspx

Предварительная версия документации Microsoft с описанием механизма привязки данных

http://quickstarts.asp.net/Futures/ajax/doc/validation.aspx

Предварительная версия документации Microsoft с описанием валидаторов

16

Использование клиентских аспектов поведения и компонентов

Обработка событий с использованием программного кода или с помощью xml-script помогает создать у пользователя ощущение интерактивности приложения, но иногда для реализации такого подхода приходится писать большие объемы кода. Это особенно актуально для случаев, когда необходимо определить специфическое поведение некоторого элемента управления в ответ на щелчок мыши или наведение указателя. К счастью, платформа ASP.NET AJAX предлагает вполне жизнеспособные альтернативы, с которыми мы познакомимся, и которые мы будем обсуждать в этой главе: *аспекты поведения* (*элементы behaviors*) и *компоненты* ASP.NET AJAX.

Аспекты поведения ASP.NET AJAX обеспечиваются функциональностью JavaScript и всегда подключаются к видимым элементам HTML-страницы. Компоненты ASP.NET AJAX, которые также содержат программный код JavaScript, могут иметь, а могут и не иметь графического представления. Примером компонента, не имеющего графического представления, может служить элемент управления `Timer`, обсуждавшийся в главе 6.

В этой главе мы займемся исследованием аспектов поведения и компонентов, входящих в состав пакета ASP.NET AJAX Futures, и продемонстрируем, как ими пользоваться.

Использование аспектов поведения

Аспекты поведения ASP.NET AJAX напоминают одноименные компоненты, добавленные компанией Microsoft в броузер Internet Explorer, в том смысле, что предопределенное поведение ASP.NET AJAX можно подключить к элементу HTML точно так же, как можно подключить аспект поведения Internet Explorer. Например, один из аспектов пове-

дения, поддерживаемый Internet Explorer, позволяет выполнять некоторые действия (изменять цвет или шрифт) при наведении указателя мыши на такой элемент, как кнопка.

Фактически пакет ASP.NET AJAX Futures поставляется с единственным аспектом поведения – `Sys.Preview.UI.ClickBehavior`. Остальные аспекты поведения определяются в дополнительных библиотеках ASP.NET AJAX Futures, которые можно подключить к приложению. Среди них:

- `Sys.Preview.UI.FloatingBehavior` (определен в *PreviewDragDrop.js*, будет обсуждаться позже в этой главе)
- `Sys.Preview.UI.OpacityBehavior` (определен в *PreviewGlitz.js*, будет обсуждаться в главе 19)
- `Sys.Preview.UI.LayoutBehavior` (также определен в *PreviewGlitz.js*)

Аспект поведения щелчок

Аспект `Sys.Preview.UI.ClickBehavior` связывает щелчок на элементе с выполняемым действием (это отражено в имени аспекта).

Пример, показанный в следующем разделе, демонстрирует эту возможность более детально. Данный пример имитирует реализацию интерфейса с вкладками – популярную особенность таких броузеров, как Firefox, Opera и Internet Explorer 7.

Реализация вкладок в этом примере имеет целью продемонстрировать лишь простейший способ работы с аспектами поведения. Если вы задумаете реализовать интерфейс с вкладками в своем приложении, используйте элемент управления Tabs из пакета Control Toolkit. Подробное обсуждение этой темы приведено в главе 12.

Две вкладки представлены двумя элементами `<div>`; пользователь может переключаться между ними с помощью элементов `<span>`:

```
<div>
  <span id="Show1" style="background-color: Fuchsia;">Tab 1</span>
  <span id="Show2" style="background-color: Fuchsia;">Tab 2</span>
</div>
<div id="Panel1" style="visibility: visible; position: absolute; top: 35px;
left: 10px">
  This is the first tab.<br />
  It is full of ASP.NET AJAX information.<br />
  Although it seems to be full of dummy text.
</div>
<div id="Panel2" style="visibility: hidden; position: absolute; top: 35px;
left: 10px">
  This is the second tab.<br />
  It is full of ASP.NET AJAX information as well.<br />
  Although it seems to be full of dummy text, too.
</div>
```

Остальная часть страницы содержит только декларативные элементы, благодаря чему удалось отказаться от программного кода. И снова нам на помощь придет xml-script. В первую очередь необходимо зарегистрировать два элемента , чтобы позднее обеспечить доступ к ним из аспектов поведения. Напомню, что в пространстве имен Sys.Preview.UI не существует клиентских элементов управления, которые служили бы представлением панелей , но имеется универсальный элемент <control>, который можно использовать, как показано ниже:

```
<control id="Panel1" />
<control id="Panel2" />
```

Аспекты поведения должны подключаться к отдельным элементам , которые образуют сами вкладки. Сначала необходимо зарегистрировать элементы:

```
<label id="Show1">
...
</label>
```

Затем в игру вступают вложенные элементы:

- Элемент <behaviors>, который будет содержать определения всех аспектов поведения, подключаемых к элементу.
- Элемент, реализующий каждый аспект поведения. В данном примере будет использоваться элемент <clickBehavior>.
- Внутри этого элемента необходимо определить вложенный элемент <click>, который идентифицирует элемент, связанный с данным аспектом поведения. (Некоторые аспекты поведения могут обслуживать сразу несколько элементов.) Ниже приводится разметка одного элемента , или одной вкладки:

```
<label id="Show1">
  <behaviors>
    <clickBehavior>
      <click>
      ...
      </click>
    </clickBehavior>
  </behaviors>
</label>
```

Теперь настало время добавить элементы <setPropertyAction> или <invokeMethodAction>, с которыми мы познакомились в главе 15. Когда пользователь щелкает на первой вкладке, первая панель становится видимой, а вторая – невидимой. Ниже приводится разметка, которая обеспечивает это поведение:

```
<label id="Show1">
  <behaviors>
    <clickBehavior>
      <click>
        <setPropertyAction target="Panel1" property="visible" value="true" />
```

```
        <setPropertyAction target="Panel2" property="visible" value="false" />
      </click>
    </clickBehavior>
  </behaviors>
</label>
```

Когда щелчок выполняется на втором элементе , первая панель становится невидимой, а вторая – видимой. Вся разметка, необходимая для реализации страницы с вкладками, приводится в примере 16.1.

Пример 16.1. Использование аспекта поведения щелчка

BehaviorClick.aspx

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server" ID="ScriptManager1">
      <Scripts>
        <asp:ScriptReference name="PreviewScript.js"
                             assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <span id="Show1" style="background-color: Fuchsia;">Tab 1</span>
      <span id="Show2" style="background-color: Fuchsia;">Tab 2</span>
    </div>
    <div id="Panel1" style="visibility: visible; position: absolute;
                                        top: 35px; left: 10px">
      This is the first tab.<br />
      It is full of ASP.NET AJAX information.<br />
      Although it seems to be full of dummy text.
    </div>
    <div id="Panel2" style="visibility: hidden; position: absolute;
                                        top: 35px; left: 10px">
      This is the second tab.<br />
      It is full of ASP.NET AJAX information, as well.<br />
      Although it seems to be full of dummy text, too.
    </div>
  </form>

  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <control id="Panel1" />
        <control id="Panel2" />
        <label id="Show1">
```

```
            <behaviors>
              <clickBehavior>
                <click>
                  <setPropertyAction target="Panel1"
                                     property="visible" value="true" />
                  <setPropertyAction target="Panel2"
                                     property="visible" value="false" />
                </click>
              </clickBehavior>
            </behaviors>
          </label>
          <label id="Show2">
            <behaviors>
              <clickBehavior>
                <click>
                  <setPropertyAction target="Panel1"
                                     property="visible" value="false" />
                  <setPropertyAction target="Panel2"
                                     property="visible" value="true" />
                </click>
              </clickBehavior>
            </behaviors>
          </label>
        </components>
      </page>
    </script>
  </body>
  </html>
```

На рис. 16.1 показана страница, созданная разметкой из примера 16.1.

В отличие от гиперссылки в данном примере при наведении указателя мыши на область, где можно выполнить щелчок, форма указателя не изменяется. Если вам необходимо будет добавить в свою страницу этот эффект, его можно реализовать с помощью стиля CSS `cursor: hand`:

```
<span id="Show1" style="background-color: Fuchsia;
                 cursor: hand">Tab 1</span>
<span id="Show2" style="background-color: Fuchsia;
                 cursor: hand">Tab 2</span>
```

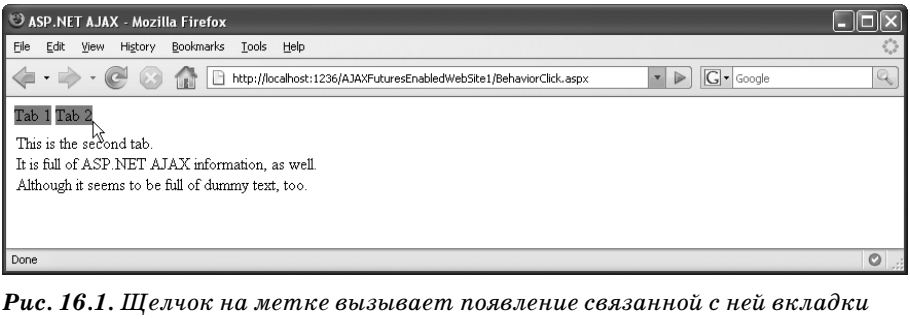


Рис. 16.1. *Щелчок на метке вызывает появление связанной с ней вкладки*

Аспект поведения «перетащить-и-отпустить»

В отличие от аспекта поведения щелчка, описанного выше, аспект поведения «перетащить-и-отпустить» (drag-and-drop), входящий в пакет Futures, не имеет самоописательного имени, такого как DragDropBehavior или похожего на него. Вместо этого он введен под именем FloatingBehavior. Особенность «перетащить-и-отпустить» довольно широко используется современными веб-порталами, но она не входит в состав библиотеки *PreviewScript.js*, которая распространяется вместе с пакетом ASP.NET AJAX Futures CTP. Эта функциональность реализована в дополнительном файле *PreviewDragDrop.js* (от термина «DragDrop»).

Реализовать возможность буксировки элементов интерфейса с помощью ASP.NET AJAX довольно просто. Прежде всего, вам потребуется элемент ASP.NET AJAX Panel – панель, которая будет буксироваться. HTML-элемент работал бы как и раньше, но при этом Panel позволяет разместить внутри все, что угодно, в чем вы сейчас убедитесь.

В этом примере мы будем использовать панель для создания небольшого прямоугольника, имитирующего дисплей, который отображает число входящих сообщений:

```
<asp:Panel CssClass="mailbox" ID="DragPanel" runat="server">
  <p>
    You currently have <asp:Label id="inbox" runat="server"></asp:Label>
    e-mail messages in your <a href="http://www.hotmail.com/">inbox</a>.
  </p>
</asp:Panel>
```

В данном случае «ящик для входящих сообщений» будет содержать случайное число новых сообщений электронной почты (примерно так отображается число сообщений в экране регистрации Windows XP). Ниже приводится фрагмент, который генерирует случайное число для отображения:

```
<script runat="server">
  protected void Page_Load(object sender, EventArgs e)
  {
    inbox.Text = new Random().Next(0, 100).ToString( );
  }
</script>
```

CSS-класс mailbox, на который ссылается элемент управления Panel, не содержит ничего экстраординарного, он должен включать лишь определение толщины рамки и ширину прямоугольника.

```
<style type="text/css">
.mailbox { border: solid 2px black; width: 150px; }
</style>
```

Теперь осталось добавить в xml-script элемент <floatingBehavior> и связать его с панелью. В свойство handle следует записать идентификатор элемента, который будет служить рукояткой для буксировки. В на-

шем случае вся панель будет исполнять роль такой рукоятки, поэтому используем ее идентификатор.

```
<script type="text/xml-script">
  <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
    <components>
      <control id="DragPanel">
        <behaviors>
          <floatingBehavior handle="DragPanel" />
        </behaviors>
      </control>
    </components>
  </page>
</script>
```

Полный программный код страницы приводится в примере 16.2.

Пример 16.2. Добавление аспекта поведения «перетащить-и-отпустить» к панели

```
DragDrop.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  protected void Page_Load(object sender, EventArgs e)
  {
    inbox.Text = new Random().Next(0, 100).ToString( );
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
  <style type="text/css">
  .box { border: solid 2px black; }
  .mailbox { border: solid 2px black; width: 150px; }
  </style>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference name="PreviewScript.js"
                             assembly="Microsoft.Web.Preview" />
        <asp:ScriptReference name="PreviewDragDrop.js"
                             assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <asp:Panel ID="ContentPanel" CssClass="box" runat="server">
      <h1>My Portal</h1>
```

```
        <p>
          Welcome to your personal portal, powered by Microsoft ASP.NET AJAX.
          The mail status window is freely draggable. Welcome to your personal
          portal, powered by ASP.NET AJAX.
          The mail status window is freely draggable. Welcome to your personal
          portal, powered by ASP.NET AJAX.
          The mail status window is freely draggable.
        </p>
        [...]
      </asp:Panel>
      <asp:Panel CssClass="mailbox" ID="DragPanel" runat="server">
        <p>
          You currently have <asp:Label id="inbox" runat="server"></asp:Label>
          mails in your <a href="http://www.hotmail.com/">inbox</a>.
        </p>
      </asp:Panel>

    </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <control id="DragPanel">
          <behaviors>
            <floatingBehavior handle="DragPanel" />
          </behaviors>
        </control>
      </components>
    </page>
  </script>
  </body>
  </html>
```

Запустите пример и посмотрите, как выглядит страница в окне броузера. Вы можете отбуксировать прямоугольник в любое место внутри породившей его страницы и оставить его там (но вы не можете, например, отбуксировать панель в нижнюю часть окна, расположенную за пределами HTML-разметки страницы). На рис. 16.2 видно, что сквозь перемещаемую панель в процессе буксировки видна лежащая ниже панель с текстом. Следует отметить, что панель вернется на свое первоначальное место при обновлении страницы. Далее в этой главе вы узнаете, как сохранять положение панели между сеансами броузера.

Использование расширителя «перетащить-и-отпустить»

Некоторые аспекты поведения также доступны в виде элементов управления (*web controls*). Так, аспект `FloatingBehavior` имеет двойника – `DragOverlayExtender`. Этот невизуальный элемент управления можно поместить в страницу для добавления («расширения») элементу управления возможности перетаскивания. Для работы с ним не требуется xml-script, а для доступа к свойствам расширителя можно использовать программный код на стороне сервера.

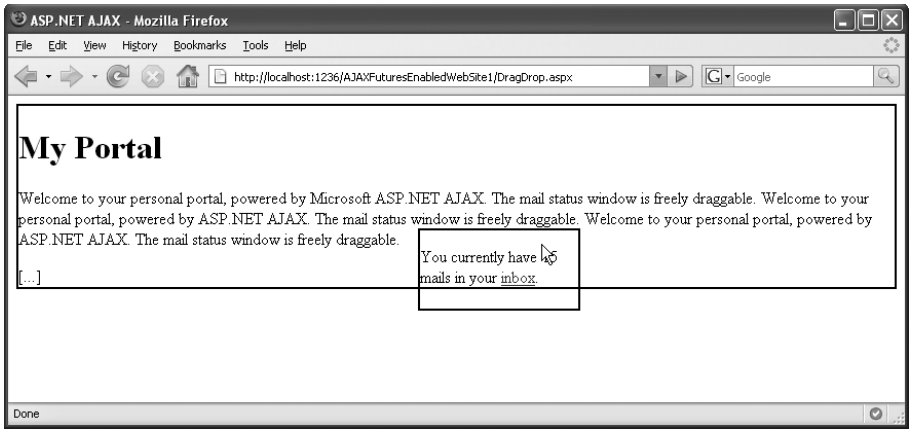


Рис. 16.2. *Прямоугольник можно перемещать в пределах страницы*

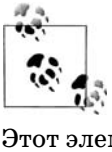
Вам потребуются следующие свойства расширителя:

`Enabled`

Активирует эффект

`TargetControlID`

Ссылка на панель, которую требуется сделать перемещаемой



На случай, если вы вдруг зададитесь вопросом, для чего этому компоненту необходимо свойство `Enabled`: оно позволяет включать и выключать эффект программным способом из сценария.

Этот элемент управления позволяет перетаскивать панель с сообщением внутри страницы (в пределах, описанных после примера 16.2):

```
<asp:DragOverlayExtender ID="DragOverlayExtender1" runat="server"
  TargetControlID="DragPanel" Enabled="true" />
```

Полный код страницы приводится в примере 16.3 (дополнительная панель с текстом используется для увеличения пространства для перемещения прямоугольника).

Пример 16.3. Добавление поведения «перетащить-и-отпустить» к панели с помощью расширителя

```
DragDropExtender.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  protected void Page_Load(object sender, EventArgs e)
  {
```

```
    inbox.Text = new Random().Next(0, 100).ToString( );
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
  <style type="text/css">
  .box { border: solid 2px black; }
  .mailbox { border: solid 2px black; width: 150px; }
  </style>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference name="PreviewScript.js"
                             assembly="Microsoft.Web.Preview" />
        <asp:ScriptReference name="PreviewDragDrop.js"
                             assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <asp:Panel ID="ContentPanel" CssClass="box" runat="server">
      <h1>My Portal</h1>
      <p>
        Welcome to your personal portal, powered by Microsoft ASP.NET AJAX.
        The mail status window is freely draggable. Welcome to your personal
        portal, powered by ASP.NET AJAX.
        The mail status window is freely draggable. Welcome to your personal
        portal, powered by ASP.NET AJAX.
        The mail status window is freely draggable.
      </p>
      [...]
    </asp:Panel>
    <asp:Panel CssClass="mailbox" ID="DragPanel" runat="server">
      <p>
        You currently have <asp:Label id="inbox" runat="server"></asp:Label>
        mails in your <a href="http://www.hotmail.com/">inbox</a>.
      </p>
    </asp:Panel>
    <asp:DragOverlayExtender ID="DragOverlayExtender1" runat="server"
                             TargetControlID="DragPanel" Enabled="true" />
  </form>
</body>
</html>
```

Индивидуализация поведения «перетащить-и-отпустить»

В предыдущих примерах имеется одно ограничение. Как уже было продемонстрировано, наша панель может свободно перемещаться по всей поверхности страницы. Однако если покинуть страницу и вер-

нуться к ней позже, вы увидите, что самое последнее положение панели не сохранилось. Это ограничение вполне преодолимо.

И снова ключевой окажется возможность использования уже имеющегося кода. В ASP.NET 2.0 имеются средства индивидуализации в виде свойств профиля (ссылки на дополнительную информацию о профилях в ASP.NET 2.0 вы найдете в разделе «Для дополнительного

Рекомендации по реализации поведения «перетащить-и-отпустить»

Реализовать поведение «перетащить-и-отпустить» на JavaScript достаточно сложно, особенно если необходимо обеспечить работоспособность программного кода во всех типах броузеров. Реализация в ASP.NET AJAX Futures CTP работает на удивление хорошо, однако здесь есть несколько аспектов, которые требуется учитывать. Во-первых, точка «сброса» элемента должна находиться в пределах информационного наполнения страницы. Это одна из причин, почему в примерах выше нам потребовалось добавлять ничего не значащий текст: обязательно должно быть некоторое пространство, в пределах которого можно будет перемещать прямоугольник. Без этого может возникать эффект «возврата», то есть когда производится попытка оставить прямоугольник за пределами области информационного наполнения, он немедленно перемещается в первоначальную позицию. Этот эффект был особенно характерен для старых версий ASP.NET AJAX Futures CTP.

Когда выполняется прокрутка страницы или изменяются размеры окна, ситуация ухудшается еще больше, поскольку расширитель не всегда точно учитывает позицию прокрутки. В пакете ASP.NET AJAX Control Toolkit имеется небольшой сценарий на JavaScript, который исправляет эту ошибку. Текст этого сценария (с небольшими изменениями) приводится ниже. Всякий раз, когда изменяется размер окна, он соответствующим образом устанавливает высоту области информационного наполнения:

```
<script type="text/javascript">
  function fixBodyHeight( ) {
    document.body.style.height =
      Math.max(document.documentElement.scrollHeight,
               document.body.scrollHeight) + "px";
  }

  function pageLoad( ) {
    fixBodyHeight( );
      $addHandler(window, "resize", fixBodyHeight);
  }
</script>
```

чтения» в конце этой главы, кроме того, обзор этих возможностей приводится в главе 7). Платформа ASP.NET AJAX поддерживает свойства профиля для некоторых из своих элементов управления, включая и DragDropExtender. Положение перемещаемой панели можно сохранить в свойстве ProfileProperty компонента DragOverlayExtender.

Чтобы иметь возможность сохранять данные о положении, нужно создать свойство профиля в файле *Web.config* в виде следующей разметки:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>
    <anonymousIdentification enabled="true" />
    <profile>
      <properties>
        <add name="DragPanelPosition" allowAnonymous="true" />
      </properties>
    </profile>
    [...]
  </system.web>

  <system.web.extensions>
    <scripting>
      <webServices>
        <profileService enabled="true"
                        writeAccessProperties="DragPanelPosition"
                        readAccessProperties="DragPanelPosition" />
        [...]
      </webServices>
    </scripting>
  </system.web.extensions>

</configuration>
```

Если не включить элемент <anonymousIdentification enabled="true" />, то профиль и возможность сохранять положение панели будут предоставлены только зарегистрированным пользователям (тем, кто зарегистрировался или как-то иначе прошел процедуру аутентификации).

Добавьте эти изменения в существующий файл *Web.config* вашего приложения. Затем вам необходимо активировать поддержку службы управления профилями в странице, добавив элемент <asp:ProfileService>. Не забудьте установить свойство AutoSave в значение true, чтобы измененная позиция панели автоматически сохранялась при выполнении каждой операции перетаскивания.

```
<asp:ProfileService ID="ProfileService1" runat="server" AutoSave="true" />
```

В заключение необходимо добавить в объявление DragDropExtender ссылку на свойство профиля, которое будет использоваться для сохранения положения прямоугольника. Значение свойства ProfileServiceID элемента DragDropExtender должно соответствовать идентификатору элемента управления ProfileService, только что добавленного в страницу.

```
<asp:DragOverlayExtender ID="DragOverlayExtender1" runat="server"
  TargetControlID="DragPanel" Enabled="true"
  ProfileProperty="DragPanelPosition" ProfileServiceID="ProfileService1" />
```

Теперь после повторной загрузки страницы элемент будет возвращаться в сохраненную позицию. Если приглядеться повнимательнее, можно заметить, что в самом начале страница отображается с панелью в позиции по умолчанию, и лишь потом она перемещается в сохраненную ранее позицию. В примере 16.4 приводится полный код этой страницы.

Пример 16.4. Добавление поведения «перетащить-и-отпустить» к панели с возможностью запоминания позиции

```
DragDropExtenderProfile.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  protected void Page_Load(object sender, EventArgs e)
  {
    inbox.Text = new Random().Next(0, 100).ToString( );
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
  <style type="text/css">
  .box { border: solid 2px black; }
  .mailbox { border: solid 2px black; width: 150px; }
  </style>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Scripts>
        <asp:ScriptReference name="PreviewScript.js"
                             assembly="Microsoft.Web.Preview" />
        <asp:ScriptReference name="PreviewDragDrop.js"
                             assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <asp:Panel ID="ContentPanel" CssClass="box" runat="server">
      <h1>My Portal</h1>
      <p>
        Welcome to your personal portal, powered by Microsoft ASP.NET AJAX.
        The mail status window is freely draggable. Welcome to your personal
        portal, powered by ASP.NET AJAX.
        The mail status window is freely draggable. Welcome to your personal
        portal, powered by ASP.NET AJAX.
```

```
        The mail status window is freely draggable.
      </p>
      [...]
    </asp:Panel>
    <asp:Panel CssClass="mailbox" ID="DragPanel" runat="server">
      <p>
        You currently have <asp:Label id="inbox" runat="server"></asp:Label>
        mails in your <a href="http://www.hotmail.com/">inbox</a>.
      </p>
    </asp:Panel>
    <asp:DragOverlayExtender ID="DragOverlayExtender1" runat="server"
                             TargetControlID="DragPanel" Enabled="true"
                             ProfileProperty="DragPanelPosition"
                             ProfileServiceID="ProfileService1" />
    <asp:ProfileService ID="ProfileService1" runat="server" AutoSave="true" />
  </form>
</body>
</html>
```

При первом запуске этого примера можно заметить, что страница отображается в броузере с заметной задержкой. Это происходит из-за того, что ASP.NET затрачивает некоторое время на создание базы данных, где будет храниться информация профиля. База данных профилей создается в каталоге App_Data веб-сайта в виде файла *ASPNETDB.MDF*. Если открыть его, можно увидеть запись с координатами панели в базе данных *aspnet_Profile* (рис. 16.3). Каждый раз, когда будет изменяться позиция панели, ASP.NET AJAX будет отправлять HTTP-запрос на обновление данных в профиле (рис. 16.4).

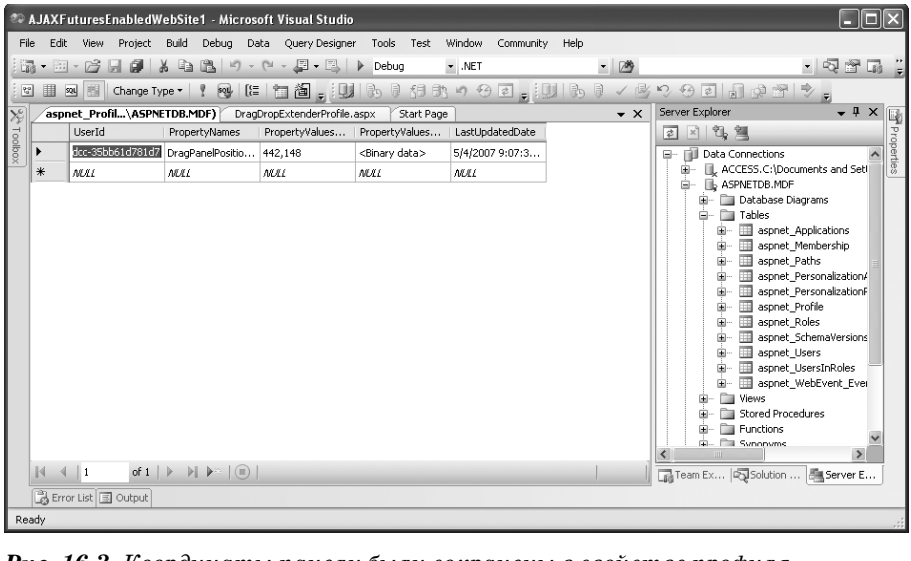


Рис. 16.3. *Координаты панели были сохранены в свойстве профиля*

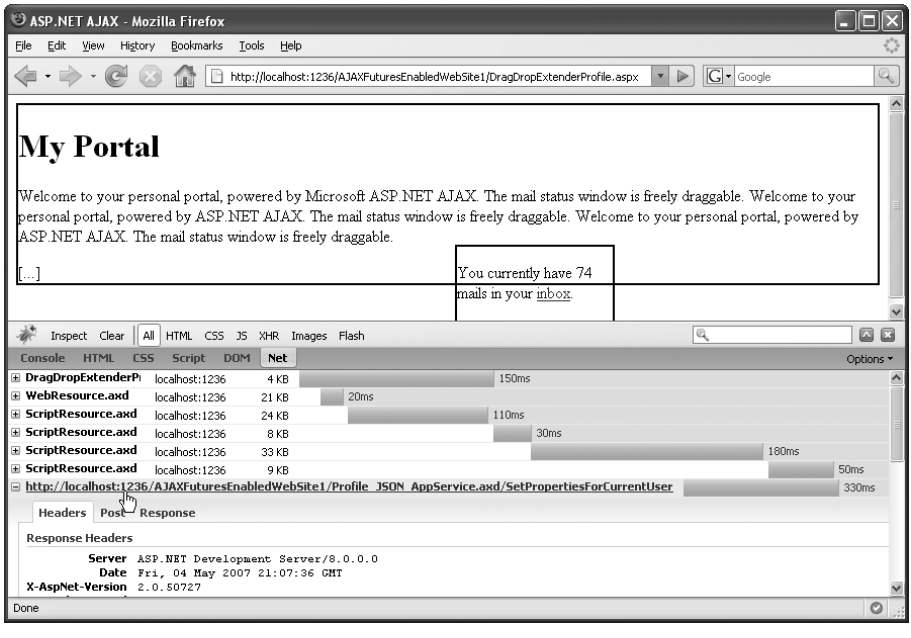


Рис. 16.4. *Для обновления профиля платформа ASP.NET AJAX отправляет в фоновом режиме HTTP-запрос*

Использование компонентов

Компоненты ASP.NET AJAX инкапсулируют самостоятельный программный код JavaScript, который не привязан к конкретным HTML-элементам страницы. Каждый компонент ASP.NET AJAX содержит ряд функциональных возможностей JavaScript для обеспечения использования в сценариях единого интерфейса. В отличие от аспекта поведения, который непременно должен быть привязан к конкретному элементу страницы, компонент может предложить более широкие функциональные возможности. В состав пакета Futures входит несколько компонентов, большинство из которых предназначены для работы с данными (в чем вы убедитесь в главе 17), но здесь мы рассмотрим только один компонент, обладающий большой практической ценностью: компонент таймера. Это эквивалент для xml-script серверного элемента управления `Timer`, демонстрировавшегося в главе 6. Он обеспечивает аналогичную функциональность, но не имеет официальной поддержки Microsoft, так же как и xml-script, и сам пакет Futures.

Использование компонента Timer

Помимо элемента управления `<asp:Timer>` (глава 6) существуют еще два альтернативных способа воспользоваться таймером, которые пре-

доставляет пакет ASP.NET AJAX Futures CTP – использовать xml-script или программный код JavaScript.

Начнем с варианта, основанного на xml-script. Элемент, используемый в качестве таймера, называется `<timer>` и обладает следующими свойствами:

`enabled`

Чтобы активировать таймер, в это свойство нужно записать значение `true`

`interval`

Длительность интервала времени в миллисекундах, через которое произойдет событие `tick`

Единица измерения – миллисекунды – обусловлена тем, что таймер из пакета ASP.NET AJAX Futures использует метод JavaScript `window.setInterval()`, который измеряет время в миллисекундах.

Остальная часть сценария не содержит ничего сложного. Событие `tick` должно быть обработано с помощью `<invokeMethodAction>` или `<setPropertyAction>`:

```
<timer id="Timer1" interval="5000" enabled="true">
  <tick>
    <invokeMethodAction ... />
    <setPropertyAction ... />
  </tick>
</timer>
```

Альтернативный, программный подход заключается в создании JavaScript-объекта таймера:

```
var timer = new Sys.Preview.Timer( );
```

Теперь с помощью методов доступа можно настроить таймер:

```
timer.set_enabled(true);
timer.set_interval(5000);
```

Не забудьте добавить обработчик события `tick`:

```
timer.add_tick(function( ) {
  // ...
});
```

И наконец, таймер нужно инициализировать, в противном случае он не будет работать:

```
timer.initialize();
```

Попробуем использовать этот таймер вместе с панелями-вкладками (из примера 16.1). В примере 16.5 используются оба способа добавления

таймера в страницу. Первый таймер создается с помощью xml-script и активизирует вторую вкладку каждые пять секунд. Второй таймер создается программно. Он активизирует первую вкладку, тоже через каждые пять секунд. Однако программный код задерживает создание второго таймера на 2.5 секунды с помощью функции `window.setTimeout()`. Таким образом, каждый из таймеров срабатывает через 5-секундный интервал, но со сдвигом относительно другого на 2.5 секунды, поэтому переключение панелей происходит каждые 2.5 секунды.

Пример 16.5. Два разных таймера сразу

```
Timer.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
  <script type="text/javascript">
  function pageLoad( ) {
    setTimeout("setupTimer( );", 2500);
  }

  function swapPanels( ) {
    $get("Panel1").style.visibility = "hidden";
    $get("Panel2").style.visibility = "visible";
  }

  function setupTimer( ) {
    swapPanels( );
    var timer = new Sys.Preview.Timer( );
    timer.set_enabled(true);
    timer.set_interval(5000);
    timer.add_tick(swapPanels);
    timer.initialize( );
  }
  </script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server" ID="ScriptManager1">
      <Scripts>
        <asp:ScriptReference name="PreviewScript.js"
                             assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div id="Panel1" style="visibility: visible; position: absolute;
                                     top: 35px; left: 10px">
      This is the first tab.<br />
      It is full of ASP.NET AJAX information.<br />
      Although it seems to be full of dummy text.
```

```
    </div>
    <div id="Panel2" style="visibility: hidden; position: absolute;
                                          top: 35px; left: 10px">
      This is the second tab.<br />
      It is full of ASP.NET AJAX information, as well.<br />
      Although it seems to be full of dummy text, too.
    </div>
  </form>

  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <control id="Panel1" />
        <control id="Panel2" />
        <timer id="Timer1" interval="5000" enabled="true">
          <tick>
            <setPropertyAction target="Panel1" property="visible"
                               value="true" />
            <setPropertyAction target="Panel2" property="visible"
                               value="false" />
          </tick>
        </timer>
      </components>
    </page>
  </script>
</body>
</html>
```

Подведение итогов

В этой главе рассматривались аспекты поведения, такие как щелчок и «перетащить-и-отпустить». Здесь также было дано введение в компоненты ASP.NET AJAX, на которые, как было показано, можно сослаться с помощью xml-script. Хотя сами компоненты реализованы на JavaScript, тем не менее xml-script предоставляет возможность декларативного добавления функциональности к элементам управления и веб-сайтам.

Для дополнительного чтения

http://quickstarts.asp.net/Futures/ajax/doc/behaviors.aspx

Предварительная версия документации Microsoft к аспектам поведения в ASP.NET AJAX Futures

17

Использование данных, размещенных на стороне сервера

Особенности ASP.NET AJAX, о которых говорилось до сих пор, позволяют получить существенную экономию сил и времени, затрачиваемых на разработку. А теперь наступило время познакомиться с еще более мощными функциональными возможностями, которыми обладает эта платформа.

В этой главе вы узнаете, как с помощью ASP.NET AJAX подключаться к базам данных и как привязывать данные из этих источников к элементам страницы. Рассматриваемые функциональные возможности не ограничены простейшими статическими элементами управления, такими как текстовые поля, и позволяют выполнять привязку данных со сложной структурой. С использованием ASP.NET AJAX вы сможете отображать данные в виде таблиц и списков HTML и, кроме того, как и в случае со многими особенностями, рассмотренными ранее, сможете создавать свои собственные источники данных.

В главе 16 рассказывалось о привязке данных к клиентским элементам управления. Но при этом не рассматривалась серверная часть. Эта глава дополняет сведения о привязке данных информацией об использовании данных, размещенных на стороне сервера. Здесь будет представлен пример веб-службы, которая извлекает данные из источника данных и возвращает их клиенту. После этого будет продемонстрировано, как с помощью клиентских элементов управления ASP.NET AJAX и разметки xml-script отобразить эти данные в формате HTML.

Использование элемента управления ListView

Лучший способ отобразить данные средствами ASP.NET AJAX заключается в использовании элемента управления `ListView` (в xml-script –

элемент `<listView>`). Этот элемент управления наделен возможностью выполнять обход *списка* (*list*) данных в цикле, чтобы пользователь мог *просматривать* (*view*) результаты – отсюда имя элемента управления.

Внутри элемента `<listView>` в xml-script можно определить два шаблона отображения:

`<layoutTemplate>`

Определяет шаблон представления всего списка данных

`<itemTemplate>`

Определяет шаблон представления отдельного элемента списка (item) данных

Существует еще целый ряд атрибутов (которые подробно будут рассматриваться в следующем разделе) и возможность привязки данных к элементам списка. В качестве целевого можно выбрать любой подходящий элемент HTML. Статические списки (нумерованные или маркированные), списки выбора (элемент `<select>`) и таблицы – наиболее часто используемые элементы, поскольку они прекрасно подходят для отображения списков данных.

Привязка данных к элементу управления ListView

Неупорядоченный список представляет наиболее очевидную форму отображения данных, получаемых из источника данных на стороне сервера. Следующий пример выполняет запрос на получение данных с сервера баз данных и отображает их в виде маркированного списка.

Прежде чем углубиться в xml-script, добавим разметку HTML, которая будет использоваться для отображения данных. В первую очередь нам потребуется контейнер, как правило, элемент `<div>`, который будет вмещать отображаемый список данных. Его разметка выглядит следующим образом:

```
<div id="output">
  здесь будет выводиться список производителей</div>
```

Затем в отдельный контейнер макета необходимо добавить два шаблона (макета и элемента списка данных). Этот контейнер будет объявлен невидимым (`display:none`). Контейнер макета сам не выполняет отображения данных, а выступает в роли контейнера для элементов HTML, которые используются для размещения и оформления данных. Фактически данные будут отображаться в только что описанном контейнере `output`, который вначале исполняет только роль заполнителя.

Внутри контейнера макета необходимо разместить несколько элементов (с соответствующими идентификаторами):

- Внешний контейнер, представляющий элемент `<layoutTemplate>`, как правило, это элемент `<div>`.
- Внутренний контейнер, соответствующий элементу `<itemTemplate>`.

- Отдельные элементы, исполняющие роль заполнителей для элементов данных (такие как элементы), получаемых из источника данных.

Если вы знакомы с элементом управления Repeater из ASP.NET, подобная схема размещения элементов должна быть вам знакома (например, контейнер макета здесь соответствует самому элементу управления Repeater). Разница состоит в том, что самый внешний контейнер (самый первый элемент , описанный выше) соответствует контейнеру для самого элемента Repeater.

Следующий фрагмент содержит пример использования неупорядоченного списка (элемент <ul>). В качестве внешнего контейнера используется элемент . Отдельные элементы списка данных отображаются в виде элементов <li> (его родительским элементом является элемент <ul>). В результате получается следующая разметка, которая исполняет роль шаблона:

```
<div style="display:none;">        <!-- скрывает заполнители -->
  <div id="vendorsLayout">         <!-- контейнер шаблона списка -->
    <ul id="vendorsItemParent">    <!-- контейнер шаблона элемента списка -->
      <li id="vendorsItem">
        <span id="vendorsName">здесь будет выведено название производителя
        </span>
      </li>
    </ul>
  </div>
</div>
```

В этом примере нельзя уменьшить число элементов, объединив два внешних контейнера – невидимый элемент и элемент контейнера шаблона списка (в данном примере – это элемент vendorsLayout). В противном случае выводимые данные будут невидимы, даже несмотря на то, что весь макет будет вставлен в элемент output. Поэтому нам необходим дополнительный элемент (соответствующий элементу <itemTemplate>), который не будет скрыт средствами CSS (невидимым будет только внешний элемент).

Прежде чем продолжить работу над страницей отображения данных, нам необходимо создать данные, с которыми мы будем работать. Для этого мы создадим веб-службу. Нам потребуется некоторый механизм, который будет отдавать данные в виде свойств объекта, возвращаемого веб-службой. Механизм привязки данных в ASP.NET AJAX для элемента listView не способен напрямую работать с наборами данных ADO.NET. Но имеются две возможности, которые используются наиболее часто:

- Объект DataTable, точнее DataRowColection внутри DataTable.
- Собственный класс, в котором все данные помещаются в члены класса.

Разработка собственного класса позволяет обеспечить более высокую гибкость, хотя обычно это ведет к увеличению объема программного кода. С другой стороны, объект DataRowCollection проще в использовании. Чтобы получить желаемую коллекцию записей со всеми данными, достаточно создать объект DataSet, а затем обратиться к свойству Table[0].Rows. Как и раньше, начиная с главы 1 мы будем использовать тестовую базу данных *AdventureWorks*. В этом примере будут запрашиваться значения полей AccountNumber и Name из таблицы Vendor. В примере 17.1 приводится программный код веб-службы, которая возвращает данные из *AdventureWorks* в виде объекта DataRowCollection. Не забывайте использовать атрибут [ScriptService] для всех веб-служб, которые будут использоваться ASP.NET AJAX, и атрибут [WebMethod] для отдельных методов веб-службы, которые будут доступны клиентам.

Если вы попытаетесь обратиться к этой веб-службе непосредственно из броузера, то скорее всего получите сообщение об ошибке «System.Data.DataRow cannot be serialized because it does not have a parameterless constructor» (невозможно сериализовать System.Data.DataRow, потому что он не имеет конструктора, вызываемого без аргументов). Это сообщение можно проигнорировать. Оно никак не сказывается на функциональности веб-службы при обращении к ней из клиентского сценария.

Пример 17.1. Веб-служба, возвращающая объект DataRowCollection

```
ListViewVendors.asmx

<%@ WebService Language="C#" Class="Vendors" %>

using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Data;
using System.Data.SqlClient;

[WebService(Namespace = "http://hauser-wenz.de/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.Web.Script.Services.ScriptService]
public class Vendors : System.Web.Services.WebService
{
  [WebMethod]public DataRowCollection GetVendors( )
  {
    SqlConnection conn = new SqlConnection(
      "server=(local)\\SQLEXPRESS; Integrated Security=true;
                                   Initial Catalog=AdventureWorks");
    conn.Open( );
    SqlCommand comm = new SqlCommand(
      "SELECT TOP 10 AccountNumber, Name FROM Purchasing.Vendor",
      conn);
    SqlDataAdapter adap = new SqlDataAdapter(comm);
```

```
        DataSet ds = new DataSet( );
        adap.Fill(ds);
        return ds.Tables[0].Rows;
      }
    }
```

Предварительная версия ASP.NET AJAX (Atlas) могла работать непосредственно с объектом DataTable. Однако ASP.NET AJAX требует, чтобы веб-служба возвращала объект DataRowCollection. В противном случае сервер будет посылать клиенту еще и общую информацию обо всех столбцах, что вызовет путаницу в компоненте ASP.NET AJAX, отвечающем за отображение этих данных.

При использовании DataRowCollection вы должны знать об одной проблеме в ASP.NET AJAX. Когда вы попытаетесь обратиться к веб-службе из JavaScript, то получите сообщение об ошибке, указывающей на невозможность разрешить циклическую ссылку. Данная ошибка вызвана неспособностью ASP.NET выполнить сериализацию объектов DataTable и DataRowCollection без дополнительной помощи. Такая «дополнительная помощь» обеспечивается парой конвертеров, которые необходимо поместить в файл *Web.config*. Загляните в раздел <webServices> (если этого раздела не существует, то создайте его в узле <scripting>, вложенном в узел <system.web.extensions>). Иногда бывает необходимо сделать это при переносе программного кода, написанного под Atlas, на платформу ASP.NET AJAX.

Теперь добавим следующую разметку:

```
<jsonSerialization maxJsonLength="500000000">
  <converters>
    <add name="DataSetConverter"
type="Microsoft.Web.Preview.Script.Serialization.Converters.DataSetConverter,
Microsoft.Web.Preview"/>
    <add name="DataRowConverter"
type="Microsoft.Web.Preview.Script.Serialization.Converters.DataRowConverter,
Microsoft.Web.Preview"/>
    <add name="DataTableConverter"
type="Microsoft.Web.Preview.Script.Serialization.Converters.DataTableConverter,
Microsoft.Web.Preview"/>
  </converters>
</jsonSerialization>
```

Здесь сначала устанавливается максимальная длина строки в формате JSON, которая может получиться в процессе сериализации. Вы можете использовать значение больше или меньше указанного, в зависимости от конкретных требований. Далее добавляются три конвертера – для объектов DataSet, DataRow и DataTable. Они будут выполнять преобразование объектов данных, которые мы собираемся использовать.

Как вариант, можно написать веб-службу, которая вместо объекта DataRowCollection будет возвращать массив данных. Т.к. веб-служба, которую мы разрабатываем, будет использовать поля AccountNumber и Name из *AdventureWorks*, то при использовании нашего собственного типа данных этот тип должен содержать два строковых свойства с именами AccountNumber и Name. В следующем фрагменте демонстрируется, как можно реализовать такой тип данных:

```
public class Vendor
{
  string _AccountNumber;
  string _Name;

  public string AccountNumber
  {
    get
    {
      return _AccountNumber;
    }
    set
    {
      _AccountNumber = value;
    }
  }

  public string Name
  {
    get
    {
      return _Name;
    }
    set
    {
      _Name = value;
    }
  }

  public Vendor(string AccountNumber, string Name)
  {
    this._AccountNumber = AccountNumber;
    this._Name = Name;
  }

  public Vendor( )
  {
  }
}
```

Пустой конструктор public Vendor(){} должен присутствовать обязательно, чтобы класс мог быть сериализован. Если опустить этот конструктор и попытаться использовать класс в веб-службе, то при попытке загрузить файл *.asmx* непосредственно из броузера вы получите сообщение об ошибке. Однако веб-служба

останется работоспособной, и ее можно будет вызывать из сценария JavaScript. Этот дополнительный конструктор облегчает тестирование веб-службы с помощью броузера, хотя и не добавляет никакой дополнительной функциональности, которая могла бы использоваться в сценариях.

Ниже приводится измененная версия веб-службы, использующая пользовательский тип данных, которая выполняет запрос к таблице Purchasing.Vendors в базе данных *AdventureWorks* и выбирает из нее записи с данными (в нашем примере 10 первых записей), как и предыдущий пример:

```
[WebMethod]
public Vendor[] GetVendors( )
{
  SqlConnection conn = new SqlConnection(
    "server=(local)\\SQLEXPRESS; Integrated Security=true;
                                  Initial Catalog=AdventureWorks");
  conn.Open( );
  SqlCommand comm = new SqlCommand(
    "SELECT TOP 10 AccountNumber, Name FROM Purchasing.Vendor",
    conn);
  SqlDataReader dr = comm.ExecuteReader( );
```

Теперь вместо того, чтобы возвращать объект DataRowCollection, программа проходит по списку и для каждой записи в таблице создает элемент Vendor. Затем получившийся список преобразуется в массив и возвращается клиенту:

```
List<Vendor> v = new List<Vendor>( );

while (dr.Read( ))
{
  v.Add(new Vendor(
    dr["AccountNumber"].ToString( ),
    dr["Name"].ToString( )));
  }
  return v.ToArray( );
}
```

В этом примере используется конструкция, впервые появившаяся в .NET Framework 2.0: обобщение (generics). Чтобы использовать обобщения, необходимо импортировать соответствующие пространства имен (System.Collections, где реализована поддержка класса List, и System.Collections.Generic). В примере 17.2 приводится полный программный код версии веб-службы, которая использует сконструированный нами тип данных Vendor.

Пример 17.2. Эта веб-служба возвращает результат нашего собственного типа

```
ListViewVendorsCustom.asmx

<%@ WebService Language="C#" Class="Vendors" %>
```

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Data;
using System.Data.SqlClient;
using System.Collections;
using System.Collections.Generic;

public class Vendor
{
  string _AccountNumber;
  string _Name;

  public string AccountNumber
  {
    get
    {
      return _AccountNumber;
    }
    set
    {
      _AccountNumber = value;
    }
  }

  public string Name
  {
    get
    {
      return _Name;
    }
    set
    {
      _Name = value;
    }
  }

  public Vendor(string AccountNumber, string Name)
  {
    this._AccountNumber = AccountNumber;
    this._Name = Name;
  }

  public Vendor( )
  {
  }
}

[WebService(Namespace = "http://hauser-wenz.de/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.Web.Script.Services.ScriptService]
public class Vendors : System.Web.Services.WebService
{
  [WebMethod]
```

```
    public Vendor[] GetVendors( )
    {
      SqlConnection conn = new SqlConnection(
        "server=(local)\\SQLEXPRESS; Integrated Security=true;
                                      Initial Catalog=AdventureWorks");
        conn.Open( );
      SqlCommand comm = new SqlCommand(
        "SELECT TOP 10 AccountNumber, Name FROM Purchasing.Vendor",
        conn);
      SqlDataReader dr = comm.ExecuteReader( );
      List<Vendor> v = new List<Vendor>( );
      while (dr.Read( ))
      {
        v.Add(new Vendor(
        dr["AccountNumber"].ToString( ),
        dr["Name"].ToString( )));
      }
      return v.ToArray( );
    }
  }
```

В исходных текстах примеров, которые можно загрузить с сайта книги, имеются оба варианта веб-службы (в файлах *ListViewVendors.asmx* и *ListViewVendorsCustom.asmx*) – в первом используется объект `DataRowCollection`, а во втором – нестандартный тип данных. Для опробования следующих далее примеров вы можете использовать любую из этих версий – они полностью взаимозаменяемы.

Теперь вернемся к странице ASP.NET, откуда вызывается веб-служба. Веб-службы детально рассматривались в главе 5, поэтому я лишь напомню, что следует предпринять, чтобы ими можно было пользоваться. Во-первых, в xml-script следует поместить ссылку на файл *.asmx*. В результате на стороне клиента будет создан объект-посредник, представляющий удаленную веб-службу. Это значит, что локальный объект будет обладать теми же методами, что и удаленная служба, – вызовы локальных методов будут автоматически преобразованы в вызовы удаленных методов. Эти вызовы производятся асинхронно (как и другие запросы `XMLHttpRequest`, которые мы применяли в главе 3). Для обработки полученных результатов используется функция обратного вызова, которая вызывается после того, как веб-служба вернет данные.

Включая в страницу элемент управления ASP.NET AJAX `ScriptManager`, в первую очередь следует добавить ссылку на файл *.asmx* веб-службы. Как это сделать, показано ниже:

```
<asp:ScriptManager runat="server">
  <Services>
    <asp:ServiceReference Path="ListViewVendors.asmx" />
  </Services>
</asp:ScriptManager>
```

После загрузки страницы следует обратиться к веб-службе. Однако понятие «после загрузки страницы» несколько сбивает с толку. Например, следующий фрагмент работать не будет:

```
<script type="text/javascript">
  window.onload = function( ) {
    Vendors.GetVendors(callComplete);
  }
</script>
```

Событие load в HTML-страницы возникает сразу после загрузки всей разметки HTML-страницы. Однако вполне возможно, что к этому моменту библиотека ASP.NET AJAX и объект-посредник еще не были полностью загружены. Поэтому данный сценарий может потерпеть неудачу, выдав такое сообщение об ошибке: «Vendors not defined» (символ Vendors не определен). По этой причине лучше задержать исполнение программного кода. Для этой цели можно было бы использовать метод window.setTimeout() или подождать, пока пользователь щелкнет на кнопке, чтобы получить данные, написав что-нибудь вроде нижеследующего (функция loadVendors() будет реализована на следующем шаге):

```
<input type="button" value="Load Vendors" onclick="loadVendors( );" />
```

Но еще лучше использовать специальный метод pageLoad(), который предоставляется платформой ASP.NET AJAX:

```
<script type="text/javascript">
  function pageLoad( ) {
    Vendors.GetVendors(callComplete);
  }
</script>
```

Вслед за этим можно вызвать веб-службу:

```
<script type="text/javascript">
  function loadVendors( ) {
    Vendors.GetVendors(callComplete, callError);
  }
```

Полученные результаты будут переданы первой функции обратного вызова (или ошибка – второй). В функции обратного вызова необходимо сделать следующее:

1. Получить ссылку на элемент, в котором будут отображаться данные (в данном примере – это).
2. Обратиться к свойству control и вызвать его метод set_data(), чтобы передать результат вызова веб-службы.

Следуя этим требованиям, получаем следующий программный код:

```
  function callComplete(result) {
    $get("output").control.set_data(result);
  }
</script>
```

Единственное, что осталось сделать, – это написать разметку xml-script. Здесь есть некоторые сложности, но начинается она относительно просто. Сначала создается элемент <script>, затем внутри него – элемент <page>, а внутри этого элемента – элемент <components>:

```
<script type="text/xml-script">
  <page xmlns="http://schemas.microsoft.com/xml-script/2005">
    <components>
    ...
    </components>
  </page>
</script>
```

Теперь внутрь элемента <components> можно поместить элемент <listView>. В этом теге необходимо определить значения некоторых атрибутов:

itemTemplateParentElementId

Идентификатор элемента, родительского по отношению к отдельным элементам списка. Звучит странно, но в данном примере это относится к элементу <ul>.

id

Идентификатор элемента, куда помещается результат.

Следующая разметка получается в случае использования неупорядоченного списка:

```
<listView itemTemplateParentElementId="vendorsItemParent" id="output">
...
</listView>
```

Внутри <listView> должны быть определены шаблон списка и шаблон элемента списка. Первый из них определяется просто – достаточно сослаться на внешний элемент :

```
<listView itemTemplateParentElementId="vendorsItemParent" id="output">
  <layoutTemplate>
    <template layoutElement="vendorsLayout" />
  </layoutTemplate>
  ...
</listView>
```

А вот определение элемента <itemTemplate> выглядит немного сложнее. На этот раз необходимо сослаться на отдельный элемент списка. В нашем примере – это элемент <li>.

```
<listView itemTemplateParentElementId="vendorsItemParent" id="output">
  <layoutTemplate>
    <template layoutElement="vendorsLayout" />
  </layoutTemplate>
  <itemTemplate>
    <template layoutElement="vendorsItem">
```

```
      ...
      </template>
    </itemTemplate>
  </listView>
```

Внутри элемента `<template>` следует определить привязку данных к каждому элементу списка. Поскольку в этом примере будет выводиться текст, можно использовать элемент `<label>`, который имеет свое представление в ASP.NET AJAX в виде элемента управления `Label`. В коде разметки нужно определить значения следующих двух свойств:

`dataPath`

Имя свойства класса, которое будет служить источником данных

`property`

Свойство элемента управления `Label`, куда должны записываться данные

В результате мы приходим к следующей разметке:

```
<listView itemTemplateParentElementId="vendorsItemParent" id="output">
  <layoutTemplate>
    <template layoutElement="vendorsLayout" />
  </layoutTemplate>
  <itemTemplate>
    <template layoutElement="vendorsItem">
      <label id="vendorsName">
        <bindings>
          <binding dataPath="Name" property="text" />
        </bindings>
      </label>
    </template>
  </itemTemplate>
</listView>
```

Это немалый труд, причем все придется делать без поддержки механизма IntelliSense. Но результат стоит того. Заключительный шаг состоит в том, чтобы загрузить библиотеку JavaScript – *PreviewScript.js* из пакета Futures, которая содержит фактическую клиентскую функциональность:

```
<asp:ScriptManager runat="server">
  <Services>
    <asp:ServiceReference Path="ListViewVendors.asmx" />
  </Services>
  <Scripts>
    <asp:ScriptReference Name="PreviewScript.js"
                         Assembly="Microsoft.Web.Preview" />
  </Scripts>
</asp:ScriptManager>
```

В примере 17.3 приводится полный код разметки и сценариев для этой страницы.

Пример 17.3. Привязка данных к HTML-списку

```
ListViewUnorderedList.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script type="text/javascript">
  function loadVendors( ) {
    Vendors.GetVendors(callComplete, callError);
  }

  function callComplete(result) {
    $get("output").control.set_data(result);
  }

  function callError(result) {
    $get("output").innerHTML = "Error: " + result.get_message( );
  }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server">
      <Services>
        <asp:ServiceReference Path="ListViewVendors.asmx" />
      </Services>
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <input type="button" value="Load Vendors" onclick="loadVendors( );" />
    <div id="output">
      здесь будет выводиться список производителей</div>
    <div style="display:none;">
      <div id="vendorsLayout">
        <ul id="vendorsItemParent">
          <li id="vendorsItem"><span id="vendorsName">
            здесь будет выводиться список производителей</span></li>
        </ul>
      </div>
    </div>
  </form>

<script type="text/xml-script">
  <page xmlns="http://schemas.microsoft.com/xml-script/2005">
    <components>
      <listView itemTemplateParentElementId="vendorsItemParent" id="output">
```

```
        <layoutTemplate>
          <template layoutElement="vendorsLayout" />
        </layoutTemplate>
        <itemTemplate>
          <template layoutElement="vendorsItem">
            <label id="vendorsName">
              <bindings>
                <binding dataPath="Name" property="text" />
              </bindings>
            </label>
          </template>
        </itemTemplate>
      </listView>
    </components>
  </page>
</script>

</body>
</html>
```

На рис. 17.1 показано содержимое страницы после щелчка на кнопке Load Vendors (загрузить список производителей).

Далее коротко описывается, что происходит в действительности:

1. Когда производится щелчок на кнопке, вызывается веб-служба.
2. Как только веб-служба вернет данные, исполняется функция обратного вызова.
3. Программный код JavaScript в цикле обходит набор данных, полученный от веб-службы.

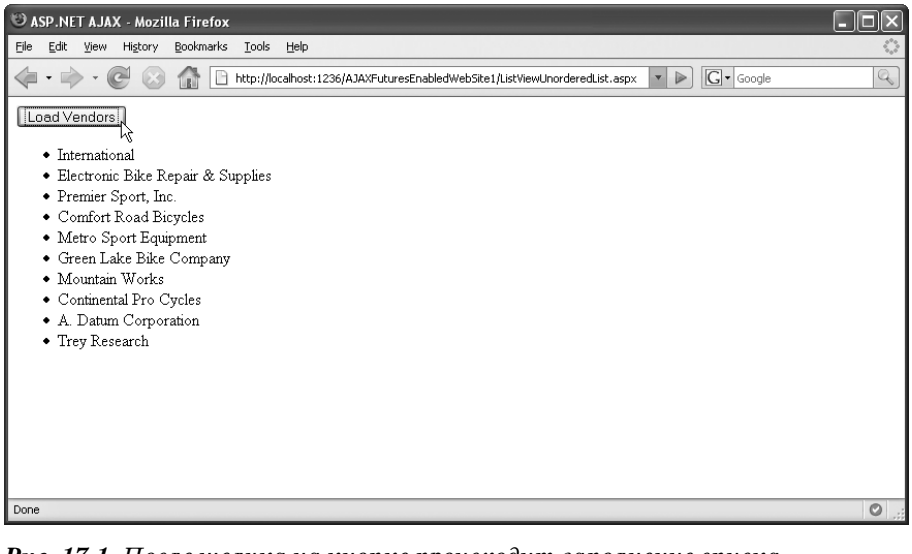


Рис. 17.1. *После щелчка на кнопке происходит заполнение списка*

4. В соответствии с данными в xml-script происходит заполнение шаблонов данными и создается список в невидимом элементе `<div>`.
5. Список копируется (с помощью функций DOM) в место назначения – элемент `<div>` с идентификатором `output`.

Привязка данных к HTML-таблице

Для отображения данных вместо неупорядоченного списка можно использовать HTML-таблицу – версию `GridView` в ASP.NET для ASP.NET AJAX. Для этого придется немного изменить HTML-разметку. Вместо элементов `<ul>` и `<li>` будут использоваться элементы `<table>` и `<tr>`. Кроме того, так как таблица может отображать данные в нескольких колонках, можно использовать все данные, получаемые от веб-службы, включая поля `Name` *и* `AccountNumber`.

Для каждой записи с данными будет создаваться строка в таблице (`<tr>`). Внутри строки будут созданы две ячейки (`<td>`), по одной для каждой колонки, возвращаемой веб-службой.

Использование списков выбора HTML

К сожалению, подход к созданию списка, продемонстрированный в примере 17.3, не годится для работы с HTML-элементом списка `<select>`. Давайте посмотрим, как выглядит типичный элемент `<select>`:

```
<select>
  <option value="1">one</option>
  <option value="2">two</option>
  <option value="3">three</option>
</select>
```

Внутри элементов `<option>` не допускается вставлять другие элементы разметки HTML. Например, вы могли бы попробовать нечто подобное:

```
<select>
  <option value="1"><span id="text1">one</span></option>
  <option value="2">two</option>
  <option value="3">three</option>
</select>
```

Однако это не сработает. Поэтому для заполнения списка данными невозможно использовать подход, продемонстрированный в примере 17.3. Однако, чтобы заполнить список динамически, можно использовать другой прием ASP.NET AJAX, который описывается в этой книге: клиентский элемент `Select` из пакета Futures поддерживает возможность привязки данных!

Ниже приводится (скрытый) макет с элементом <table>, для которого ASP.NET AJAX поддерживает возможность привязки данных, расположенных на стороне сервера:

```
<div style="display: none;">
  <div id="vendorsLayout">
    <table id="vendorsItemParent">
      <tr><th>Account Number</th><th>Name</th></tr>
      <tr id="vendorsItem">
        <td><span id="vendorsAccountNumber">номер счета производителя</
span></td>
        <td><span id="vendorsName">название производителя</span></td>
      </tr>
    </table>
  </div>
</div>
```

Однако существует одна проблема. Броузеры семейства Mozilla отобразят таблицу, а окно броузера Internet Explorer останется пустым. Internet Explorer совершенно по-особенному интерпретирует HTML-таблицы, сгенерированные динамически. Этот результат интересен в свете того, что исторически Internet Explorer весьма терпимо относится к неправильно оформленному коду разметки HTML.

Таким образом, чтобы заставить работать HTML-таблицу с привязанными данными, необходимо создавать ее с разделами <thead> и <tbody>. Раздел <tbody> является родительским для каждого элемента списка данных, который отображается с помощью элемента <tr>.

Можно также добавить необязательный элемент <tfoot>, который нужно было бы вставить перед элементом <tbody>. В данном примере элемент <tfoot> не требуется.

```
<table>
  <thead>
    <tr><th>Account Number</th><th>Name</th></tr>
  </thead>
  <tbody id="vendorsItemParent">
    <tr id="vendorsItem">
      <td id="vendorsAccountNumber">номер счета производителя</td>
      <td id="vendorsName">название производителя</td>
    </tr>
  </tbody>
</table>
```

В xml-script необходимо добавить дополнительную привязку для нового элемента-заполнителя. В остальном пример работает точно так же, как и раньше: после щелчка на кнопке вызывается веб-служба, полученные результаты вставляются в элемент vendorsLayout и затем копируются в элемент output. В примере 17.4 приводится полный код, в котором жирным шрифтом выделены произведенные изменения.

Пример 17.4. Привязка данных к HTML-таблице

```
ListViewTable.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
    function loadVendors( ) {
      Vendors.GetVendors(callComplete, callError);
    }

    function callComplete(result) {
      $get("output").control.set_data(result);
    }

    function callError(result) {
      $get("output").innerHTML = "Error: " + result.get_message( );
    }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server">
      <Services>
        <asp:ServiceReference Path="ListViewVendors.asmx" />
      </Services>
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <input type="button" value="Load Vendors" onclick="loadVendors( );" />
    <div id="output">
      здесь будет выводиться список производителей</div>
    <div style="display: none;">
      <div id="vendorsLayout">
        <table>
          <thead>
            <tr><th>Account Number</th><th>Name</th></tr>
          </thead>
          <tbody id="vendorsItemParent">
            <tr id="vendorsItem">
              <td id="vendorsAccountNumber">номер счета производителя</td>
              <td id="vendorsName">название производителя</td>
            </tr>
          </tbody>
        </table>
      </div>
```

```
    </div>
  </form>

  <script type="text/xml-script">
    <page xmlns="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <listView itemTemplateParentElementId="vendorsItemParent" id="output">
          <layoutTemplate>
            <template layoutElement="vendorsLayout" />
          </layoutTemplate>
          <itemTemplate>
            <template layoutElement="vendorsItem">
              <label id="vendorsAccountNumber">
                <bindings>
                  <binding dataPath="AccountNumber" property="text" />
                </bindings>
              </label>
              <label id="vendorsName">
                <bindings>
                  <binding dataPath="Name" property="text" />
                </bindings>
              </label>
            </template>
          </itemTemplate>
        </listView>
      </components>
    </page>
  </script>

</body>
</html>
```

Результат работы страницы показан на рис. 17.2.

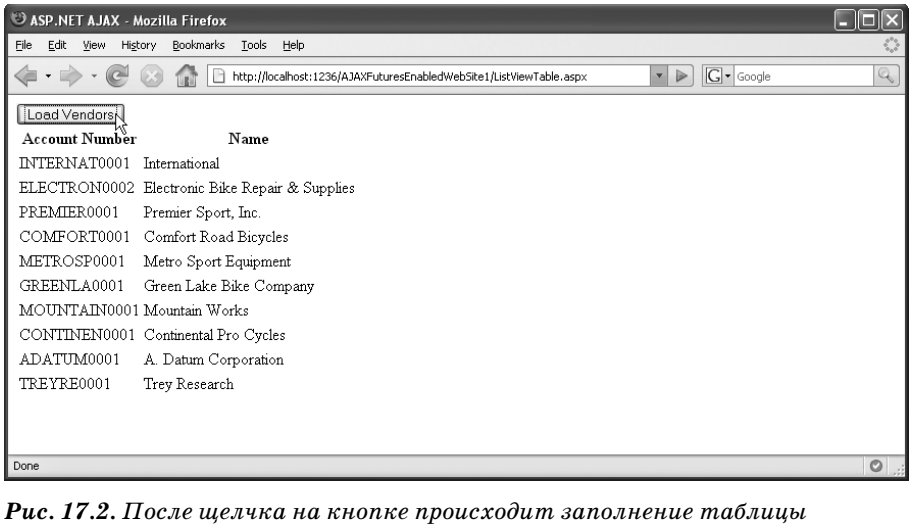


Рис. 17.2. *После щелчка на кнопке происходит заполнение таблицы*

Создание собственного источника данных

Для обеспечения большей гибкости доступа к данным или при невозможности придерживаться структуры, предоставляемой имеющимся источником данных, можно реализовать свой собственный источник данных в виде класса ASP.NET. Поскольку ASP.NET AJAX в значительной степени зависит от веб-служб, вам необходимо будет реализовать свой класс, наследующий класс `DataService`. Реализация этого класса находится в пространстве имен `Microsoft.Web.Preview.Services`, которое определено в пакете Futures. В своем классе необходимо будет реализовать методы по умолчанию, используемые для работы с объектом данных. Список этих методов находится в перечислении `System.ComponentModel.DataObjectMethodType` и включает в себя следующие методы:

- `Delete`
- `Insert`
- `Select`
- `Update`

Отображение данных из собственного источника данных

Для демонстрации мы сначала реализуем метод `SELECT` веб-службы, который, как и раньше, будет извлекать данные из таблицы `Purchasing.Vendors` в базе данных *AdventureWorks*.

Как и в предыдущих примерах, вы можете реализовать метод, возвращающий требуемые данные. С помощью атрибута `[DataObjectMethod(DataObjectMethodType.Select)]` вы объявляете задаваемый метод методом, выполняющим «выборку». Фактическое имя метода не имеет значения. В качестве типа данных значения, возвращаемого методом, снова можно использовать свой собственный тип, как показано в примере 17.5.

Пример 17.5. Метод, возвращающий нестандартный тип данных

ListViewVendorsDataServiceCustomType.asmx, фрагмент

```
[DataObjectMethod(DataObjectMethodType.Select)]
public Vendor[] GetVendors( )
{
  SqlConnection conn = new SqlConnection(
    "server=(local)\\SQLEXPRESS; Integrated Security=true; Initial
Catalog=AdventureWorks");
  conn.Open( );
  SqlCommand comm = new SqlCommand(
    "SELECT TOP 10 AccountNumber, Name FROM Purchasing.Vendor",
    conn);
  SqlDataReader dr = comm.ExecuteReader( );
  List<Vendor> v = new List<Vendor>( );

  while (dr.Read( ))
```

```
    {
      v.Add(new Vendor(
      dr["AccountNumber"].ToString( ),
      dr["Name"].ToString( )));
    }
    return v.ToArray( );
  }
```

Как вариант, можно возвращать тип DataTable, для чего требуется меньше программного кода, как показано в примере 17.6. Выбор этого типа возвращаемого значения требует, чтобы метод возвращал DataTable, а не DataRowCollection!

Пример 17.6. Метод, возвращающий DataTable

```
ListViewVendorsDataService.asmx

<%@ WebService Language="C#" Class="VendorsDataService" %>
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Data;
using System.Data.SqlClient;
using Microsoft.Web.Preview.Services;using System.ComponentModel;

[WebService(Namespace = "http://hauser-wenz.de/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.Web.Script.Services.ScriptService]
public class VendorsDataService : DataService
{[DataObjectMethod(DataObjectMethodType.Select)]
  public DataTable GetVendors( )
  {
    SqlConnection conn = new SqlConnection(
      "server=(local)\\SQLEXPRESS; Integrated Security=true;
InitialCatalog=AdventureWorks");
    conn.Open( );
    SqlCommand comm = new SqlCommand(
      "SELECT TOP 10 AccountNumber, Name FROM Purchasing.Vendor",
      conn);
    SqlDataAdapter adap = new SqlDataAdapter(comm);
    DataSet ds = new DataSet( );
    adap.Fill(ds);
    return ds.Tables[0];
  }
}
```

Эти файлы не содержат методов, явно помеченных атрибутом [WebMethod]. Однако если обратиться к одной из этих веб-служб непосредственно из броузера, можно увидеть, что они имеют два метода: GetData() и SaveData() (рис. 17.3). Оба метода ожидают получить массив parameters с дополнительной информацией. Платформа ASP.NET AJAX автоматически генерирует обязательные параметры, поэтому вы про-

Рис. 17.3. *Методы, объявленные в базовом классе*

сто вызываете методы с именами, зафиксированными в `DataObjectMethodType` — `Delete`, `Insert`, `Select` и `Update`.

Пример 17.6 существует в двух видах: с использованием `DataTable` и с пользовательским типом данных. В исходных текстах примеров, которые можно загрузить с сайта книги, варианты хранятся в файлах *ListViewVendorsDataServiceCustom.asmx* и *ListViewVendorsDataService.asmx*. Определение пользовательского типа находится в файле с добавлением .txt к имени (*.asmx.txt*), чтобы избежать конфликта имен с файлом *ListViewVendorsCustom.asmx*.

На стороне ASP.NET требуется наличие разметки HTML, в которой находится определение шаблона, и разметки xml-script, обеспечивающей привязку данных. Первая составляющая – это HTML-таблица. Не забывайте использовать разделы `<thead>` и `<tbody>`, чтобы данные удовлетворяли требованиям броузера Internet Explorer.

Следующая разметка HTML исполняет роль шаблона, к которому с помощью ASP.NET AJAX выполняется привязка данных, получаемых из нестандартного источника данных:

```
<div id="output">
  vendor list goes here</div>
<div style="display: none;">
  <div id="vendorsLayout">
    <table>
    <thead>
      <tr><th>Account Number</th><th>Name</th></tr>
    </thead>
      <tbody id="vendorsItemParent">
        <tr id="vendorsItem">
          <td id="vendorsAccountNumber">номер счета производителя</td>
          <td id="vendorsName">название производителя</td>
        </tr>
      </tbody>
```

```
      </table>
    </div>
  </div>
```

Однако в xml-script необходимо внести некоторые изменения по сравнению с предыдущим примером. Начинается все как обычно:

```
<script type="text/xml-script">
  <page xmlns="http://schemas.microsoft.com/xml-script/2005">
    <components>
    ...
    </components>
  </page>
</script>
```

Затем следует указать ссылку на источник данных. Поскольку это необычная веб-служба, объект ScriptManager не может использоваться для ссылки на эту веб-службу. Вместо него следует использовать элемент <dataSource>. В этом элементе необходимо определить значения URL и идентификатора – идентификатор нам потребуется позже!

```
<script type="text/xml-script">
  <page xmlns="http://schemas.microsoft.com/xml-script/2005">
    <components>
    <dataSource id="vendorSource"
                serviceURL="ListViewVendorsDataService.asmx" />
      ...
    </components>
  </page>
</script>
```

Теперь настало время элемента управления ListView и его расширения – элемента <listView>. Самый важный шаг – это привязка источника данных из предыдущего фрагмента кода к элементу управления ListView. В свойства dataPath и property должно быть записано значение data, а свойство dataContext должно ссылаться на идентификатор элемента <dataSource>:

```
<listView id="vendorsList" itemTemplateParentElementId="vendorsItemParent"
          targetElement="output">
  <bindings>
    <binding dataContext="vendorSource" dataPath="data" property="data" />
  </bindings>
  ...
</listView>
```

Элементы <layoutTemplate> и <itemTemplate> остались без изменений, привязка данных выполняется к элементу <table> и вложенным в него подэлементам.

Однако мы упустили одну вещь. Данные мы привязали, но еще не загрузили. Источник данных поддерживает свойство с именем autoLoad. Если этому свойству присвоить значение true, метод Select источника данных будет вызван автоматически.

В примере 17.7 приводится полный код, решающий эту задачу.

Пример 17.7. Отображение данных, полученных из нестандартного источника данных

```
ListViewDataService.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server">
      <Scripts>
        <asp:ScriptReference Name="PreviewScript.js"
                             Assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div id="vendorsList">
      здесь будет выводиться список производителей</div>
    <div style="display: none;">
      <div id="vendorsLayout">
        <table>
          <thead>
            <tr><th>Account Number</th><th>Name</th></tr>
          </thead>
          <tbody id="vendorsItemParent">
            <tr id="vendorsItem">
              <td id="vendorsAccountNumber">номер счета поставщика</td>
              <td id="vendorsName">название поставщика</td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
  </form>

  <script type="text/xml-script">
    <page xmlns="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <dataSource id="vendorSource"
                    serviceURL="ListViewVendorsDataService.asmx"
                    autoLoad="true" />
        <listView id="vendorsList"
                  itemTemplateParentElementId="vendorsItemParent">
          <bindings>
            <binding dataContext="vendorSource" dataPath="data"
                     property="data" />
```

```
            </bindings>
            <layoutTemplate>
              <template layoutElement="vendorsLayout" />
            </layoutTemplate>
            <itemTemplate>
              <template layoutElement="vendorsItem">
                <label id="vendorsAccountNumber">
                  <bindings>
                    <binding dataPath="AccountNumber" property="text" />
                  </bindings>
                </label>
                <label id="vendorsName">
                  <bindings>
                    <binding dataPath="Name" property="text" />
                  </bindings>
                </label>
              </template>
            </itemTemplate>
          </listView>
        </components>
      </page>
    </script>

  </body>
  </html>
```

Нам удалось избежать необходимости писать программный код (исключение составляет класс `DataService` веб-службы) и обойтись одними декларациями. Этот пример выводит первые 10 записей из таблицы `Purchasing.Vendors` в виде HTML-элемента `<table>`. Таким образом, результат работы этого сценария идентичен тому, что приводится на рис. 17.2.

Управление данными

Отображение данных – это лишь первый шаг. Далее вполне логично было бы реализовать остальные методы, определяемые в перечислении `System.ComponentModel.DataObjectMethodType`. После этого можно было бы листать данные, изменять их и многое другое. Это порождает целый набор новых возможностей. Достичь более высокой производительности приложения и эффективности процесса разработки можно было бы при использовании элемента управления ASP.NET – `GridView` (или любого другого подходящего элемента управления для работы с данными). Если вас волнует проблема полного обновления страницы, которая присуща элементу управления `GridView`, вернитесь к главе 6, где был представлен способ преодоления этого ограничения – разумеется, с использованием возможностей ASP.NET AJAX.

Подведение итогов

В этой главе рассказывалось о том, как организовать со стороны клиента доступ к данным, расположенным на сервере. Для отображения информации, находящейся на сервере, мы реализовали веб-службу, а затем воспользовались механизмом привязки данных ASP.NET AJAX и специальными клиентскими элементами управления, такими как `ListView`.

Для дополнительного чтения

http://quickstarts.asp.net/Futures/ajax/doc/data.aspx

Предварительная версия документации компании Microsoft об использовании служб доступа к данным из ASP.NET AJAX

http://astoria.mslivelabs.com/

Проект компании Microsoft под кодовым названием «Astoria», цель которого состоит в том, чтобы обеспечить в приложениях возможность представлять данные в виде служб

http://download.microsoft.com/download/5/9/c/59cd0dc5-4691-4c3e-840c-66d865f27692/listview.xps

Спецификация нового (серверного) элемента управления в .NET Framework 3.5, который также называется `ListView`

18

Использование внешних веб-служб

В главах 5 и 17 обсуждалось взаимодействие веб-служб ASP.NET и ASP.NET AJAX. Объект `XMLHttpRequest`, сердце всех Ajax-приложений, работает в ограниченной среде и ему недоступно все, что находится вне пределов текущего домена. Для случая, когда необходимо получать данные от внешней веб-службы (то есть от службы, исполняющейся на другом сервере), существует единственное решение: создать посредника на своем сервере и затем обращаться к нему из своих сценариев JavaScript. В свою очередь, это позволяет создавать *гибридные* (*mashups*) веб-приложения. Гибридными называют такие приложения, которые используют данные из разных источников, чтобы «растолочь и размешать» их («mix 'n mash») и приготовить из них что-то новое. В этой главе будет показано, как сделать доступными для JavaScript данные от внешних веб-служб. После этого вы сможете с помощью JavaScript скомбинировать различные внешние источники в нечто совершенно новое.

Пакет ASP.NET AJAX Futures поддерживает реализацию посредников для работы с внешними веб-службами с использованием технологии, которая называется мост веб-службы (*web service bridge*). В следующих разделах будут созданы страницы, принимающие данные от наиболее популярных коммерческих веб-служб: поисковой службы Google и веб-службы электронной коммерции Amazon. Приемы, демонстрируемые в этой главе, без труда можно будет распространить на случаи использования веб-служб, работающих по протоколу SOAP.

Данные для этой технологии сохраняются в файле с новым расширением *.asbx*, который регистрируется шаблоном веб-сайта ASP.NET AJAX Futures в файле *Web.config*. Файлы с этим расширением могут содержать разметку XML, которая предоставляет информацию о локальном классе-посреднике (на стороне сервера), с помощью которого организован доступ к веб-службе. Программный код JavaScript страницы просто соединяется с этим файлом *.asbx*, а он берет на себя все за-

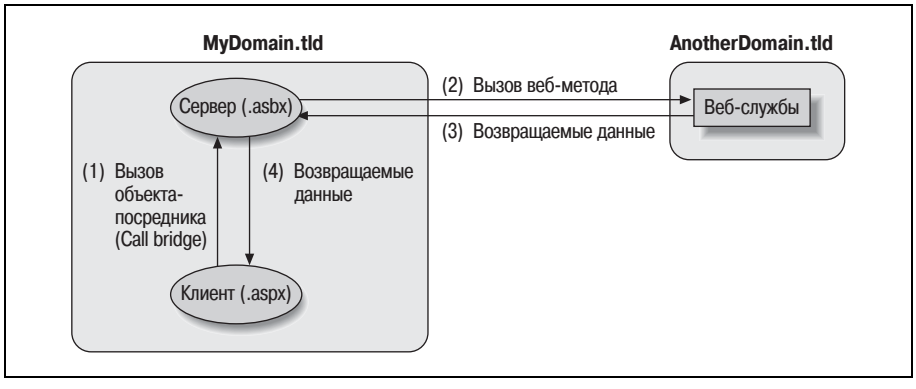


Рис. 18.1. *Клиентская страница обращается к внешней веб-службе через мост*

боты по обеспечению взаимодействий с внешней службой. Этот механизм демонстрируется на рис. 18.1.

Если вы используете шаблон веб-сайта ASP.NET AJAX Futures CTP, значит, файл *Web.config* уже содержит все необходимые настройки. Прежде чем двинуться дальше, я должен предупредить, что версии ASP.NET AJAX Futures, выпущенные в январе, мае и июле 2007 года содержат ошибку, которую можно найти в следующей строке:

```
<add extension="*.asbx"
type="Microsoft.Web.Preview.Services.BridgeBuildProvider"/>
```

Ошибка здесь – в символе *. Эту ошибку сложно заметить сразу, поскольку она проявляется, только когда некоторый сценарий сообщает, что «Для расширения .asbx отсутствует встроенный провайдер». Исправляется проблема следующим образом:

```
<add extension=".asbx"
type="Microsoft.Web.Preview.Services.BridgeBuildProvider"/>
```

Внесите это исправление на всех своих сайтах, построенных на базе ASP.NET Futures. Вам также потребуется исправлять эту ошибку во всех вновь создаваемых сайтах с поддержкой Futures. (Как вариант, можно изменить шаблон файла *Web.config*, чтобы при создании новых сайтов данная ошибка уже была исправлена. Шаблон файла *Web.config* находится в каталоге установки ASP.NET Futures, которым по умолчанию является каталог *%programdir%\Microsoft ASP.NET\ASP.NET Futures July 2007\v1.2.61025\web_config*.)

Использование веб-службы Google

Веб-служба Google предоставляет удобный программный доступ к поисковой машине с использованием интерфейсов SOAP и REST. В нашем примере при реализации моста будет использоваться интерфейс SOAP.

Чтобы иметь возможность пользоваться веб-службой Google, необходимо пройти регистрацию. Для отправки запроса перейдите на страницу *http://www.google.com/apis/soapsearch*. Google передаст вам 32-байтовый ключ лицензии, который необходимо будет отправлять вместе с каждым запросом к службе.

На момент написания этих строк компания Google временно прекратила выдачу новых ключей. Если у вас уже имеется ключ, то вы сможете работать с веб-службой. Если нет – не волнуйтесь, в разделе «Использование веб-службы Amazon» (в этой же главе) рассматриваются два примера приложений, для которых новые ключи еще выпускаются.

Регистрация расширения .asbx вручную

Без запуска пакета установки ASP.NET AJAX расширение *.asbx* не будет распознаваться веб-сервером IIS. В этом случае запустите консоль управления IIS и поставьте расширению *.asbx* в соответствие файл *aspnet_isapi.dll*, что позволит выполнять HTTP-команды GET, POST и HEAD. Кроме того, в файлы *Web.config* всех сайтов, использующих мост к веб-службам, нужно добавить следующую разметку, чтобы обеспечить корректное распознавание файлов моста:

```
<compilation>
  <buildProviders>
  ...
    <add extension=".asbx" type="Microsoft.Web.Preview.Services.
BridgeBuildProvider"/>
  </buildProviders>
</compilation>
...
<httpHandlers>
  ...
  <add verb="GET,HEAD,POST" path="*.asbx" type="
System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=1.2.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" validate="false"/>
```

Если вы используете IIS 7, добавьте в узел `<handlers>`, расположенный внутри узла `<system.webServer>` в файле *Web.config*, следующие элементы:

```
<add name="ASBXHandler" verb="GET,HEAD,POST" path="*.asbx"
preCondition="integratedMode"
type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=1.2.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"/>
```

Конечно, было бы глупо сохранять этот (секретный!) ключ лицензии в программном коде JavaScript в странице. Также не рекомендуется помещать ключ в программный код ASP.NET на стороне сервера.

Однако ключ лицензии можно поместить в раздел `<appSettings>` файла *Web.config*, как демонстрируется в следующем фрагменте:

```
<appSettings>
  <add key="GoogleLicenseKey" value="***" />
</appSettings>
```

Очевидно, что файл *Web.config*, входящий в состав исходных текстов примеров к этой главе, доступных для загрузки, *не* содержит ключ лицензии. Вам необходимо получить свой ключ и занести его в файл. Кроме того, вам следует использовать возможность шифрования записей в файле *Web.config*, чтобы зашифровать секретный ключ. Обсуждение данной особенности выходит далеко за рамки нашей темы, поэтому мы не будем использовать ее в примерах из этой главы.

На веб-сайте Google API можно найти доступный для загрузки Google Web API Developer's Kit (набор инструментальных средств разработчика Google Web API). В его состав входит WSDL-файл с именем *GoogleSearch.wsdl*, содержащий описание интерфейса веб-службы. Эта информация может использоваться инструментом *wsdl.exe* (входит в состав .NET Framework SDK) для создания класса-посредника.

Загрузите файл *GoogleSearch.wsdl* или извлеките его из Google API SDK. Кроме того, содержимое файла *.wsdl* можно просмотреть в броузере, обратившись по адресу: *http://api.google.com/GoogleSearch.wsdl*, а затем скопировать его и вставить в локальный файл. Откройте окно Command (Командная строка) и запустите следующую команду (используйте символ верхнего регистра при наборе слова «Google», так как оно определяет чувствительное к регистру символов название пространства имен, которое будет использовано позднее):

```
wsdl.exe /namespace:Google GoogleSearch.wsdl
```

Чтобы иметь возможность запускать команду `wsdl.exe` из командной строки, вам может потребоваться добавить в переменную PATH путь к каталогу, где находятся утилиты .NET SDK. По умолчанию утилиты располагаются в каталоге *%windir%\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin*.

Эта команда создаст файл *GoogleSearch.cs*, в котором будут находиться определения класса и методов для вызова поисковой службы Google. Как показано на рис. 18.2, вы получите несколько предупреждений, однако в данном случае их можно совершенно спокойно игнорировать. Кроме того, обратите внимание на то, что мы используем пространство имен для класса, чтобы избежать потенциальных конфликтов с другими классами в веб-приложении.

```
Visual Studio 2005 Command Prompt
C:\Documents and Settings\Christian\My Documents>wsdl.exe /namespace:Google
GoogleSearch.wsdl
Microsoft (R) Web Services Description Language Utility
[Microsoft (R) .NET Framework, Version 2.0.50727.42]
Copyright (C) Microsoft Corporation. All rights reserved.
Warning: This web reference does not conform to WS-I Basic Profile v1.1.
R2706: A wsdl:binding in a DESCRIPTION MUST use the value of "literal" for the u
se attribute in all soapbind:body, soapbind:fault, soapbind:header and soapbind:
headerfault elements.
  -  Input element soapbind:body of operation 'doGetCachedPage' on portType 'Goo
gleSearchBinding' from namespace 'urn:GoogleSearch'.
  -  Output element soapbind:body of operation 'doGetCachedPage' on portType 'Go
ogleSearchBinding' from namespace 'urn:GoogleSearch'.
  -  Input element soapbind:body of operation 'doSpellingSuggestion' on portType
 'GoogleSearchBinding' from namespace 'urn:GoogleSearch'.
  -  Output element soapbind:body of operation 'doSpellingSuggestion' on portTyp
e 'GoogleSearchBinding' from namespace 'urn:GoogleSearch'.
  -  Input element soapbind:body of operation 'doGoogleSearch' on portType 'Goog
leSearchBinding' from namespace 'urn:GoogleSearch'.
  -  Output element soapbind:body of operation 'doGoogleSearch' on portType 'Goo
gleSearchBinding' from namespace 'urn:GoogleSearch'.

For more details on the WS-I Basic Profile v1.1, see the specification
at http://www.ws-i.org/Profiles/BasicProfile-1.1.html.

Writing file 'C:\Documents and Settings\Christian\My Documents\GoogleSearchServi
ce.cs'.

C:\Documents and Settings\Christian\My Documents>_
```

Рис. 18.2. *Создание класса-посредника для доступа к веб-службе Google*

Поместите файл *GoogleSearchService.cs* со сгенерированным классом в каталог *App_Code* приложения с поддержкой ASP.NET Futures. (Если каталог *App_Code* еще не существует, создайте его.) Это позволит вам использовать класс, не прибегая к ручной компиляции.

Следующий шаг состоит в том, чтобы создать программную обертку вокруг класса-посредника, посредством которой будет вызываться метод поиска. Мы не будем глубоко погружаться в прикладной интерфейс веб-службы Google, а отметим лишь, что веб-служба объявляет метод `doGoogleSearch()`, который принимает два аргумента: ключ лицензии Google и строку поиска. Обертка просто будет вызывать этот метод и возвращать полученные результаты, как показано в примере 18.1. Создайте файл *GoogleSearchServiceWrapper.cs* с определением класса в каталоге *App_Code*, удалите весь программный код, который уже имеется в файле, и затем скопируйте в него программный код из примера 18.1.

Пример 18.1. Класс-обертка для доступа к веб-службе Google

```
GoogleSearchServiceWrapper.cs

using Google;

public class GoogleSearchServiceWrapper
{
  public GoogleSearchResult Search(string licenseKey, string query)
  {
    GoogleSearchService gss = new GoogleSearchService();
    return gss.doGoogleSearch(
      licenseKey,
```

```
        query,
        0,     // смещение первого результата
        10,    // максимальное число результатов
        false, // следует ли отбрасывать похожие результаты
        "",    // набор условий Google, ограничивающих поиск
        false, // следует ли отбрасывать содержимое,
               // предназначенное только для совершеннолетних
        "",    // ограничение поиска по языку
        "",    // игнорируется, так же как и следующий параметр
        "");
    }
  }
```

Теперь можно использовать полученный мост к веб-службе. Для активации моста к веб-службе следует предоставить всю необходимую информацию о веб-службе в виде файла *.asbx*. Создайте в корневом каталоге вашего веб-сайта XML-файл с именем *Google.asbx*.

Добавьте в файл *.asbx* пространство имен, где будет размещаться мост к веб-службе (атрибут `namespace`) и имя класса реализации моста (атрибут `className`).

```
<bridge namespace="OReilly.AspNetAJAX" className="Google" >
```

В элементе `<proxy>` должно быть указано имя класса-обертки и каталог, где его можно будет найти:

```
<proxy type="GoogleSearchServiceWrapper, App_Code" />
```

Затем необходимо перечислить все методы веб-службы, включая имена аргументов. Все аргументы, перечисленные здесь, позднее смогут использоваться в сценариях JavaScript. Не забудьте, что ключ лицензии хранится в файле *Web.config*. Из этого следует, что значение аргумента, в котором передается ключ лицензии, не может быть установлено из JavaScript. Вместо этого мы можем загружать ключ из раздела `<appSettings>` во время исполнения, используя следующий синтаксис:

```
<parameter name="licenseKey" value="% appsettings : GoogleLicenseKey %"
serverOnly="true" />
```

Атрибут `serverOnly="true"` обеспечивает недоступность аргумента `licenseKey` для программного кода JavaScript, поэтому значение ключа всегда будет извлекаться из файла *Web.config*.

На этом все. Полный программный код из файла моста приводится в примере 18.2.

Пример 18.2. Реализация моста к веб-службе Google

```
Google.asbx

<?xml version="1.0" encoding="utf-8" ?>
<bridge namespace="OReilly.AspNetAJAX" className="Google" >
  <proxy type="GoogleSearchServiceWrapper, App_Code" />
  <method name="Search">
```

```
    <input>
      <parameter name="licenseKey"
                 value="% appsettings : GoogleLicenseKey %"
                 serverOnly="true" />
      <parameter name="query" />
    </input>
  </method>
</bridge>
```

Теперь все, что осталось сделать, это написать страницу *.aspx*, использующую мощь платформы ASP.NET AJAX. На странице нам потребуются текстовое поле, куда будет вводиться поисковый запрос, кнопка, запускающая поиск, и некоторые заполнители, где будут отображаться результаты поиска.

Разметка может выглядеть примерно так:

```
<div>
  <input type="text" id="Query" />
  <input type="button" value="Search" onclick="Search();" />
</div>
<div>
  <p>Approx. <span id="Count">0</span> results.</p>
  <ul id="Results">
  </ul>
</div>
```

Безусловно, страница должна также содержать элемент управления `ScriptManager`. В его подэлементе `<Services>` следует указать ссылку на веб-службу – файл *.asbx*. Кроме того, нужно загрузить библиотеку сценариев *PreviewScript.js* (глава 15), поскольку она будет использоваться программным кодом JavaScript.

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Services>
    <asp:ServiceReference Path="~/Google.asbx" />
  </Services>
  <Scripts>
    <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                         Name="PreviewScript.js" />
  </Scripts>
</asp:ScriptManager>
```

Этот фрагмент загрузит реализацию моста и сделает доступным в JavaScript наше пространство имен `OReilly.AspNetAJAX`. После этого можно вызвать метод обертки `Search()`, как если бы это была обычная локальная веб-служба. Обратите внимание на то, как передаются параметры, – в виде массива, в котором имена параметров служат индексами:

```
OReilly.AspNetAJAX.Google.Search(
  { "query": query },
  callComplete, callError
);
```

Данные, получаемые от веб-службы, – это JavaScript-представление объектов SOAP, возвращаемых сервером. В случае поиска в Google в возвращаемых данных имеется свойство (или подэлемент) с именем `resultElements`, которое содержит массив отдельных URL, удовлетворяющих условию поиска. Каждый URL, кроме всего прочего, имеет свойства `title` и `URL`, которые мы будем отображать на странице.

Полный код страницы в примере 18.3 содержит некоторые другие интересные эффекты на JavaScript. Например, при получении от веб-службы результаты динамически добавляются в список (HTML-элемент `<ul>`). При запуске нового запроса этот список очищается вспомогательной функцией `clearList()`. Результаты поиска в Google благодаря мосту ASP.NET AJAX к веб-службе (рис. 18.1) видны на локальной странице (рис. 18.3).

Пример 18.3. Вызов веб-службы Google

```
Google.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
  function clearList() {
    var list = $get("Results");
    while (list.firstChild != null) {
```

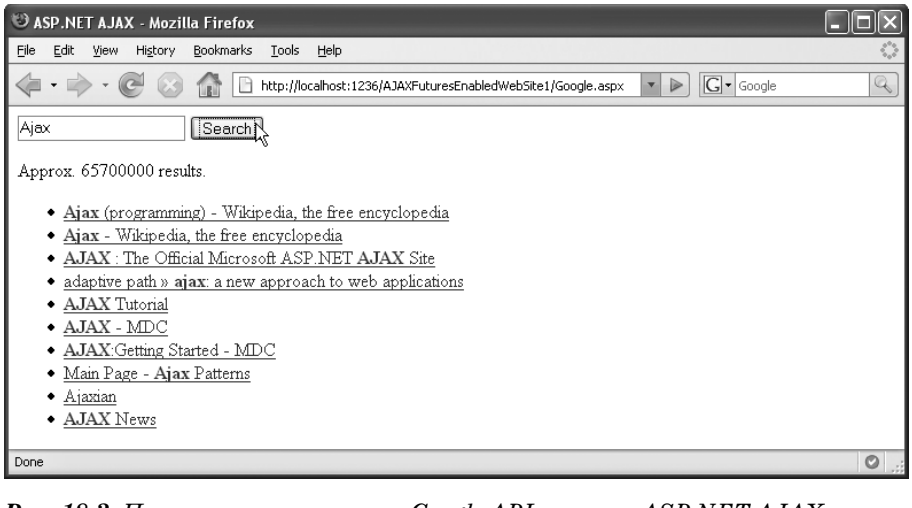


Рис. 18.3. *Поиск с использованием Google API и моста ASP.NET AJAX к веб-службам*

```
        list.removeChild(list.firstChild);
      }
    }

    function Search() {
      var query = new Sys.Preview.UI.TextBox($get('Query'))
      $get("Button").disabled = true;
      clearList();
      OReilly.AspNetAJAX.Google.Search(
        { "query": query.get_text() },
        callComplete, callError
      );
      new Sys.Preview.UI.Label($get('Count')).set_text("...");
    }

    function callComplete(result) {
      new
  Sys.Preview.UI.Label($get('Count')).set_text(result.estimatedTotalResultsCount);
      if (result.resultElements != null) {
        for (var i = 0; i < result.resultElements.length; i++) {
          var page = result.resultElements[i];
          var li = document.createElement("li");
          var a = document.createElement("a");
          a.setAttribute("href", page.URL);
          a.innerHTML = page.title;
          li.appendChild(a);
          $get("Results").appendChild(li);
        }
      }
      $get("Button").disabled = false;
    }

    function callError(result) {
      window.alert("Error! " + result.get_message());
      new Sys.Preview.UI.Label($get('Count')).set_text("0");
      $get("Button").disabled = false;
    }
    </script>

  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server">
        <Services>
          <asp:ServiceReference Path="~/Google.asbx" />
        </Services>
        <Scripts>
          <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                               Name="PreviewScript.js" />
        </Scripts>
      </asp:ScriptManager>
      <div>
        <input type="text" id="Query" />
```

```
      <input type="button" id="Button" value="Search" onclick="Search();" />
    </div>
    <div>
      <p>Approx. <span id="Count">0</span> results.</p>
      <ul id="Results">
      </ul>
    </div>
  </form>
</body>
</html>
```

При работе над этой книгой мне несколько раз пришлось столкнуться с перебоями в работе сервера веб-служб Google, поэтому не нужно сразу делать вывод об ошибках в программе, если вам не удастся немедленно получить результаты. Это может быть вызвано проблемами на удаленном сервере. В приложении А вы найдете сведения об отладке Ajax-приложений и способы изучения HTTP-трафика.

Использование веб-службы Amazon

В предыдущем разделе было продемонстрировано, как можно использовать веб-службу Google – относительно несложную в обращении, не использующую свои собственные типы данных для входных аргументов и с простым методом, который делает все, что нужно. В этом разделе будет рассматриваться более сложная веб-служба Amazon. Она поддерживает несколько типов, которые все вместе образуют поисковый запрос. И снова нас будут интересовать не конкретные подробности реализации веб-службы Amazon, а способы, которыми ASP.NET AJAX сможет использовать эти данные.

Вам снова понадобится лицензионный ключ (компанияAmazon называет его *ключом доступа* (*access key*)). Как и в случае с Google, для этого придется пройти регистрацию. URL сайта с документацией, где можно найти описание веб-службы Amazon: *http://www.amazon.com/gp/aws/landing.html*. Точно так же как и с ключом Google, вам следует поместить ключ доступа в раздел `<appSettings>` файла *Web.config* (ключ компании Amazon имеет длину 20 байт).

Файл примера, который можно загрузить с сайта книги, не содержит этот ключ, поэтому вам необходимо поместить его туда самостоятельно:

```
<appSettings>
  <add key="AmazonAccessKey" value="***" />
</appSettings>
```

Как и в примере Google, следующий шаг заключается в использовании утилиты *wsdl.exe* для создания класса-посредника из описания веб-службы Amazon в формате WSDL. Загрузить WSDL-файл можно по адресу: *http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl*.

Для создания класса посредника в виде файла *AWSECommerceService.cs* откройте окно Command (Командная строка) и запустите команду:

```
wsdl.exe /namespace:Amazon
http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl
```

Скопируйте получившийся файл *.cs* в каталог *App_Code*. Если были получены сообщения об ошибках, что может быть вызвано сбоями в работе сервера, – просто возьмите файл *AWSECommerceService.cs* из исходных текстов примеров, которые можно загрузить с сайта книги.

Реализация класса-обертки на этот раз выглядит немного сложнее, так как веб-служба использует ряд своих собственных объектов. Создайте файл класса с именем *AWSECommerceServiceWrapper.cs* в каталоге *App_Code*. В классе необходимо в первую очередь создать экземпляр класса `ItemSearchRequest`, посредством которого можно будет передать строку поиска (что искать), индекс поиска (где искать) и группу ответов (количество возвращаемых ответов).

```
public Amazon.Items Search(string accessKey, string query)
{
  ItemSearchRequest searchRequest = new ItemSearchRequest();
  searchRequest.Keywords = query;
  searchRequest.ResponseGroup = new string[] { "Small" };
  searchRequest.SearchIndex = "Books";
```

Следующий шаг заключается в создании экземпляра класса `ItemSearch`, посредством которого веб-службе передается ключ доступа и только что созданный объект класса `ItemSearchRequest`:

```
ItemSearch search = new ItemSearch();
search.AWSAccessKeyId = accessKey;
search.Request = new ItemSearchRequest[1] { searchRequest };
```

Наконец, необходимо создать экземпляр главного класса `AWSECommerceService` и вызвать метод `ItemSearch()`, передав ему в качестве аргумента объект класса `ItemSearch`. Возвращаемые данные представляют собой массив ответов на все переданные поисковые запросы (существует возможность за одно обращение отправить сразу несколько запросов).

Поскольку мы отправляем всего один запрос, то ожидается всего один результат:

```
  AWSECommerceService awse = new AWSECommerceService();
  ItemSearchResponse searchResponse = awse.ItemSearch(search);
  return searchResponse.Items[0];
}
```

В примере 18.4 содержится полный код класса *AWSECommerceServiceWrapper.cs*.

Пример 18.4. Класс-обертка для обращения к веб-службе Amazon

```
AWSECommerceServiceWrapper.cs

using Amazon;
```

```
public class AWSECommerceServiceWrapper
{
  public Amazon.Items Search(string accessKey, string query)
  {
    ItemSearchRequest searchRequest = new ItemSearchRequest();
    searchRequest.Keywords = query;
    searchRequest.ResponseGroup = new string[] { "Small" };
    searchRequest.SearchIndex = "Books";

    ItemSearch search = new ItemSearch();
    search.AWSAccessKeyId = accessKey;
    search.Request = new ItemSearchRequest[1] { searchRequest };

    AWSECommerceService awse = new AWSECommerceService();
    ItemSearchResponse searchResponse = awse.ItemSearch(search);
    return searchResponse.Items[0];
  }
}
```

Остальная часть этого демонстрационного приложения более или менее та же самая, что и в примере с Google. Файл *Amazon.asbx* исполняет роль моста к внешней веб-службе. Значение `accessKey` считывается из файла *Web.config*, а параметр запроса поступает от клиентского приложения. В примере 18.5 приводится содержимое XML-разметки для файла *Amazon.asbx*.

Пример 18.5. Мост к веб-службе Amazon

```
Amazon.asbx

<?xml version="1.0" encoding="utf-8" ?>
<bridge namespace="OReilly.AspNetAJAX " className="Amazon" >
  <proxy type="AWSECommerceServiceWrapper, App_Code" />
  <method name="Search">
    <input>
      <parameter name="accessKey"
                 value="% appsettings : AmazonAccessKey %"
                 serverOnly="true" />
      <parameter name="query" />
    </input>
  </method>
</bridge>
```

Мало того что отправка данных веб-службе Amazon стала сложнее, получение данных от нее – тоже достаточно трудоемкая процедура. Возвращаемые оберткой данные (массив элементов типа `Amazon.Item`) содержат список книг. Наиболее интересные данные в этом массиве находятся в еще одном специфическом объекте – свойстве `ItemAttributes`.

В примере 18.6 демонстрируется страница ASP.NET, которая содержит программный код, извлекающий названия и имена авторов всех книг, удовлетворяющих критериям поиска, и помещающий результаты в элемент `<ul>`. Результат показан на рис. 18.4.

Пример 18.6. Вызов веб-службы Amazon

Amazon.aspx

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script type="text/javascript">
    function clearList() {
      var list = $get("Results");
      while (list.firstChild != null) {
        list.removeChild(list.firstChild);
      }
    }

    function Search() {
      var query = new Sys.Preview.UI.TextBox($get('Query'))
      $get("Button").disabled = true;
      clearList();
      OReilly.AspNetAJAX.Amazon.Search(
        { "query": query.get_text() },
        callComplete, callError
      );
      new Sys.Preview.UI.Label($get('Count')).set_text("...");
    }

    function callComplete(result) {
      new Sys.Preview.UI.Label($get('Count')).set_text(result.TotalResults);
      if (result.Item != null) {
        for (var i = 0; i < result.Item.length; i++) {
          var article = result.Item[i];
          var author = (article.ItemAttributes.Author != null ?
            join(article.ItemAttributes.Author) + ": " : "");
          var title = article.ItemAttributes.Title;
          var li = document.createElement("li");
          var liText = document.createTextNode(author + title);
          li.appendChild(liText);
          $get("Results").appendChild(li);
        }
      }
      $get("Button").disabled = false;
    }

    function callError(result) {
      window.alert("Error! " + result.get_message());
      new Sys.Preview.UI.Label($get('Count')).set_text("0");
      $get("Button").disabled = false;
    }
```

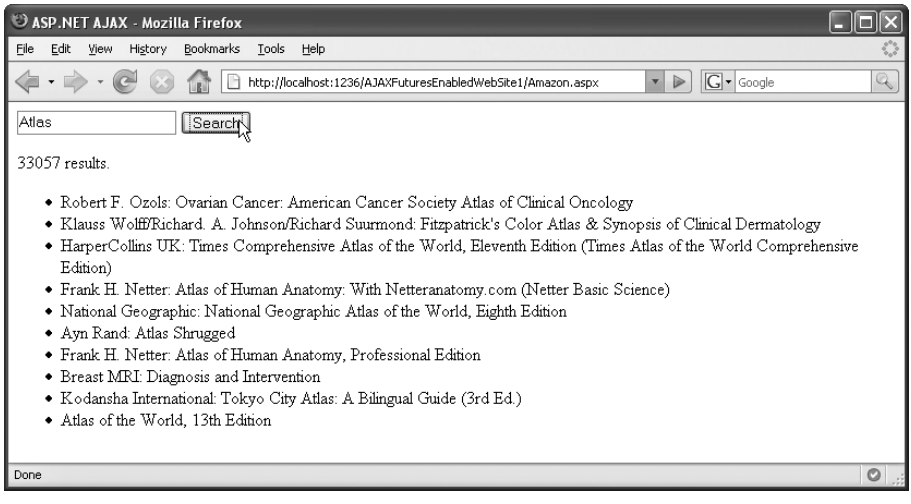


Рис. 18.4. *Поиск в каталоге Amazon с использованием моста ASP.NET AJAX*

```
    function join(a) {
      var s = "";
      for (var i=0; i < a.length - 1; i++) {
        s += a[i] + "/";
      }
      s += a[a.length - 1];
      return s;
    }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Services>
        <asp:ServiceReference Path="~/Amazon.asbx" />
      </Services>
      <Scripts>
        <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                             Name="PreviewScript.js" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="Query" />
      <input type="button" id="Button" value="Search" onclick="Search();" />
    </div>
    <div>
      <p>
        <span id="Count">0</span> results.</p>
      <ul id="Results">
```

```
        </ul>
      </div>
    </form>
  </body>
  </html>
```

Интересно, что и Google, и Amazon предоставляют доступ к своим веб-службам через интерфейсы SOAP и REST. Оба интерфейса обеспечивают одни и те же функциональные возможности. Интерфейс REST используется значительно чаще, чем SOAP. Одна из причин – повышенная сложность при использовании SOAP. Однако благодаря мосту ASP.NET AJAX к веб-службе большая часть этих сложностей снимается.

Преобразование результатов веб-службы с помощью XSLT

Данные, возвращаемые веб-службой, обычно представляют собой XML-документ (по крайней мере, при использовании SOAP или REST). В странице ASP.NET AJAX этот XML-документ представлен в виде объекта JavaScript, из которого можно извлечь всю необходимую информацию и отобразить ее с помощью HTML-элементов.

Другой способ преобразования данных веб-службы из формата XML в HTML заключается в использовании XSL-преобразований (XSLT). Описание принципов использования XSLT выходит за рамки этой книги, но в разделе «Для дополнительного чтения» в конце этой главы я приведу ссылки на ряд замечательных источников информации. Современные веб-броузеры (Mozilla, Internet Explorer, Opera 9+) обеспечивают поддержку XSLT в JavaScript, но эти реализации несовместимы между собой. Поэтому лучший способ заключается в выполнении преобразований на стороне сервера. Для этого можно написать свой собственный программный код .NET или воспользоваться компонентами ASP.NET AJAX. В этом разделе мы выполним преобразование результатов поиска, возвращаемых веб-службой Google, в разметку HTML, которая затем будет отображаться в веб-странице.

Мост ASP.NET AJAX к веб-службе поддерживает два встроенных преобразователя, которые позволяют конвертировать объекты в другой формат. Класс `Microsoft.Web.Preview.Services.XmlBridgeTransformer` конвертирует объект в формат XML, а класс `Microsoft.Web.Preview.Services.XsltBridgeTransformer` выполняет XSL-преобразование данных из формата XML в любой другой формат (обычно – HTML).

Как и прежде, начнем работу с подготовки моста, класса-обертки и сценария JavaScript, который приводит в действие механизм поиска. Сценарий JavaScript отправляет поисковый запрос из страницы мосту, который вызывает метод класса-обертки, который в свою очередь выполняет поиск. Результаты поиска возвращаются классу-обертке в ви-

де объекта (как мы уже видели ранее, мы можем работать с объектом как с массивом). Затем класс-обертка передает данные мосту. Однако на этот раз мост не будет отправлять данные странице в том виде, в каком он их получил.

В отличие от примеров, приводившихся ранее в этой главе, на этот раз мост будет дважды выполнять преобразование данных результата. Первое преобразование конвертирует объект результата в XML. Второе – применяет XSLT-преобразование, чтобы конвертировать XML в HTML. На выходе преобразования получается фактическая разметка HTML, которая затем будет использоваться для отображения списка результатов. Затем мост посылает полученную разметку HTML странице, где одной строкой JavaScript она просто вставляется в контейнер, специально предназначенный для этого.

Мы будем использовать измененные версии трех файлов, которые ранее приводились в примерах поиска в Google. Кроме того, нам потребуется еще один дополнительный файл с расширением *.xsl*, содержащий разметку XSLT. Этот файл содержит описание преобразования, которое выполняется мостом для конвертирования XML в HTML.

В корневом каталоге веб-приложения создайте новый файл XSLT с именем *Google.xsl*. В этом файле будут находиться инструкции XSLT преобразования результатов поиска Google в разметку HTML.

Как и в предыдущем примере с Google, мы будем отображать результаты в виде HTML-списка `<ul>`. Таким образом, XSL-преобразование должно обойти результаты поиска, полученные в виде XML, что можно сделать с помощью XSL-цикла `for each`. Здесь есть одно небольшое препятствие. Каждый отдельный результат поиска находится внутри элемента `<resultElement>`, а преобразователь ASP.NET AJAX XML конвертирует его в элемент `<ResultElement>`. Механизм преобразований XSL различает символы верхнего и нижнего регистров, поэтому вместо имени `resultElement` следует использовать `ResultElement`:

```
<xsl:for-each select="//resultElements/ResultElement">
...
</xsl:for-each>
```

Для каждого отдельного элемента результатов создается элемент `<li>`. Текст данного элемента – это ссылка (элемент `<a>`), которая указывает на веб-страницу для данного конкретного результата. Процессоры XSLT выполняют экранирование сущностей HTML, но веб-служба Google возвращает заголовок страницы в виде HTML (поскольку найденные совпадения выделены жирным шрифтом). Поэтому нам необходимо использовать атрибут `disable-output-encoding` для заголовков.

Другой момент: так как нам необходимо создать фрагмент HTML, в элемент XSLT – `<xsl: output>` следует включить атрибут `omit-xml-declaration`, чтобы предотвратить создание результата, окруженного тегами `<?xml ?>`. Содержимое XSLT-файла приводится в примере 18.7.

Пример 18.7. XSLT-файл для работы с веб-службой Google

```
Google.xsl

<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" encoding="utf-8" omit-xml-declaration="yes" />

  <xsl:template match="/">
    <p>
      Approx. <xsl:value-of select="//estimatedTotalResultsCount" /> matches!
      <ul>
        <xsl:for-each select="//resultElements/ResultElement">
    <li>
    <a>
    <xsl:attribute name="href">
    <xsl:value-of select="URL" />
    </xsl:attribute>
    <xsl:value-of select="title" disable-output-escaping="yes" />
    </a>
    </li>
        </xsl:for-each>
      </ul>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

Наш следующий шаг – определить точку входа в преобразование XSL. В ней задавалось бы преобразование объекта в XML (с помощью `XmlBridgeTransformer`) и преобразование XML в HTML (с помощью `XsltBridgeTransformer`). Для этой цели можно было бы создать новый метод в классе-обертке, выполняющем поиск с помощью веб-службы, который создает объект с результатами поиска для последующего преобразования, но во всем этом не содержится новой важной логики для реализации.

Можно было бы использовать новый файл моста, но тогда пришлось бы использовать иное имя класса, в противном случае Visual Studio будет выдавать предупреждения во время компиляции (хотя сама страница по-прежнему будет работать).

В предварительных версиях ASP.NET AJAX внутри реализации моста в элементе `<method>` можно было воспользоваться атрибутом `serverName` для перенаправления запросов методу класса-обертки:

```
<method name="SearchXslt" serverName="Search">
...
</method>
```

Здесь объявляется метод с именем `SearchXslt()`, который будет доступен в JavaScript, но он всего лишь исполняет существующий метод `Search()` класса-обертки.

Последние версии ASP.NET AJAX больше не поддерживают такую возможность. Таким образом, необходимо выполнить перенаправление на стороне сервера, изменив содержимое файла *GoogleSearchServiceWrapper.cs*, как показано в примере 18.8.

Пример 18.8. Измененная версия класса-обертки для доступа к службе поиска Google

```
GoogleSearchServiceWrapper.cs

using Google;

public class GoogleSearchServiceWrapper
{
  public GoogleSearchResult Search(string licenseKey, string query)
  {
    GoogleSearchService gss = new GoogleSearchService();
    return gss.doGoogleSearch(
      licenseKey,
      query,
      0,
      10,
      false,
      "",
      false,
      "",
      "",
      "");
  }

  public GoogleSearchResult SearchXslt(string licenseKey, string query)
  {
    return Search(licenseKey, query);
  }
}
```

Но вернемся к файлу с определением моста. Внутри элемента `<method>` элемент `<input>` остался без изменений, поскольку входные параметры не изменились. Однако здесь появился новый элемент `<transforms>`, который определяет два преобразователя. Преобразователю XSLT должен быть предоставлен для работы файл XSL. В примере 18.9 приводится разметка XML для измененного файла моста.

Пример 18.9. Измененная версия моста для доступа к веб-службе Google

```
Google.asbx

<?xml version="1.0" encoding="utf-8" ?>
<bridge namespace="OReilly.AspNetAJAX" className="Google" >
  <proxy type="GoogleSearchServiceWrapper, App_Code" />
```

```
  <method name="Search">
    <input>
      <parameter name="licenseKey"
                 value="% appsettings : GoogleLicenseKey %"
                 serverOnly="true" />
      <parameter name="query" />
    </input>
  </method>
  <method name="SearchXslt" serverName="Search">
    <input>
      <parameter name="licenseKey"
                 value="% appsettings : GoogleLicenseKey %"
                 serverOnly="true" />
      <parameter name="query" />
    </input>
    <transforms>
      <transform type="Microsoft.Web.Preview.Services.XmlBridgeTransformer" />
     <transform type="Microsoft.Web.Preview.Services.XsltBridgeTransformer">
        <data>
          <attribute name="stylesheetFile" value="~/Google.xsl" />
        </data>
      </transform>
    </transforms>
  </method>
</bridge>
```

Теперь все, что осталось сделать, – это обратиться к мосту. Поскольку мост возвращает фрагмент разметки HTML, то результаты, полученные от веб-службы, можно просто записать в свойство `innerHTML` контейнера `<div>`. Это существенно упрощает сценарий JavaScript.

В примере 18.10 приводится полная страница ASP.NET с разметкой и программным кодом JavaScript. Результат работы этой страницы идентичен результату работы примера 18.3 и показан на рис. 18.3.

Пример 18.10. Вызов веб-службы Google с XSL-преобразованием результатов

```
GoogleXslt.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
    function clearList() {
      $get("Results").innerHTML = "";
    }

    function Search() {
      var query = new Sys.Preview.UI.TextBox($get('Query'))
```

```
      $get("Button").disabled = true;
      clearList();
      OReilly.AspNetAJAX.Google.SearchXslt(
        { "query": query.get_text() },
        callComplete, callError
      );
    }

    function callComplete(result) {
      $get("Results").innerHTML = result;
      $get("Button").disabled = false;
    }

    function callError(result) {
      window.alert("Error! " + result.get_message());
      $get("Button").disabled = false;
    }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Services>
        <asp:ServiceReference Path="~/Google.asbx" />
      </Services>
      <Scripts>
        <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                             Name="PreviewScript.js" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="Query" />
      <input type="button" id="Button" value="Search" onclick="Search();" />
    </div>
    <div id="Results">
    </div>
  </form>
</body>
</html>
```

Безусловно, такой подход пригоден и для работы с веб-службой Amazon. Прежде всего, необходимо изменить файл моста. В примере 18.11 приводится новый код разметки.

Пример 18.11. Измененная версия моста для доступа к веб-службе Amazon

```
Amazon.asbx

<?xml version="1.0" encoding="utf-8" ?>
<bridge namespace="OReilly.AspNetAJAX" className="Amazon" >
  <proxy type="AWSECommerceServiceWrapper, App_Code" />
  <method name="Search">
    <input>
```

```
      <parameter name="accessKey"
                 value="% appsettings : AmazonAccessKey %"
                 serverOnly="true" />
      <parameter name="query" />
    </input>
  </method>
  <method name="SearchXslt" serverName="Search">
    <input>
      <parameter name="accessKey"
                 value="% appsettings : AmazonAccessKey %"
                 serverOnly="true" />
      <parameter name="query" />
    </input>
    <transforms>
      <transform type="Microsoft.Web.Preview.Services.XmlBridgeTransformer" />
     <transform type="Microsoft.Web.Preview.Services.XsltBridgeTransformer">
        <data>
          <attribute name="stylesheetFile" value="~/Amazon.xsl" />
        </data>
      </transform>
    </transforms>
  </method>
</bridge>
```

Кроме того, в классе-обертке необходимо реализовать фиктивную функцию для перенаправления обращений `SearchXslt()` методу `Search()`, как показано в примере 18.12.

Пример 18.12. Измененная версия класса-обертки для доступа к веб-службе Amazon

```
AWSECommerceServiceWrapper.cs

using Amazon;

public class AWSECommerceServiceWrapper
{
  public Amazon.Items Search(string accessKey, string query)
  {
    ItemSearchRequest searchRequest = new ItemSearchRequest();
    searchRequest.Keywords = query;
    searchRequest.ResponseGroup = new string[] { "Small" };
    searchRequest.SearchIndex = "Books";

    ItemSearch search = new ItemSearch();
    search.AWSAccessKeyId = accessKey;
    search.Request = new ItemSearchRequest[1] { searchRequest };

    AWSECommerceService awse = new AWSECommerceService();
    ItemSearchResponse searchResponse = awse.ItemSearch(search);
    return searchResponse.Items[0];
  }

  public Amazon.Items SearchXslt(string accessKey, string query)
```

```
    {
      return Search(accessKey, query);
    }
  }
```

Затем необходимо описать XSL-преобразование, которое будет конвертировать XML-документ, полученный от SOAP-службы Amazon, в HTML. В примере 18.13 представлен один из возможных вариантов.

Пример 18.13. Таблица стилей для преобразования результатов вызова веб-службы Amazon в HTML

```
Amazon.xsl

<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" encoding="utf-8" omit-xml-declaration="yes" />

  <xsl:template match="/">
    <p>
      <xsl:value-of select="//TotalResults" /> matches found!
      <ul>
        <xsl:for-each select="//Items/Item/ItemAttributes">
          <li>
            <xsl:value-of select="Author" />:
            <xsl:value-of select="Title" />
          </li>
        </xsl:for-each>
      </ul>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

Наконец, для запуска трансформации используется программный код, очень похожий на пример 18.10. В примере 18.14 содержится полный листинг.

Пример 18.14. Вызов веб-службы Amazon с XSL-преобразованием результатов

```
AmazonXslt.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
    function clearList() {
      $get("Results").innerHTML = "";
```

```
    }

    function Search() {
      var query = new Sys.Preview.UI.TextBox($get('Query'))
      $get("Button").disabled = true;
      clearList();
      OReilly.AspNetAJAX.Amazon.SearchXslt(
        { "query": query.get_text() },
        callComplete, callError
      );
    }

    function callComplete(result) {
      $get("Results").innerHTML = result;
      $get("Button").disabled = false;
    }

    function callError(result) {
      window.alert("Error! " + result.get_message());
      $get("Button").disabled = false;
    }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
      <Services>
        <asp:ServiceReference Path="~/Amazon.asbx" />
      </Services>
      <Scripts>
        <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                             Name="PreviewScript.js" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="Query" />
      <input type="button" id="Button" value="Search" onclick="Search();" />
    </div>
    <div id="Results">
    </div>
  </form>
</body>
</html>
```

На рис. 18.5 показан результат работы этой страницы. Здесь можно заметить некоторые недостатки отображения. Например, элементы списка с названиями изданий, для которых авторы не указаны, такие как энциклопедии и прочие справочники, начинаются с символа двоеточия. Этот недостаток проще было бы устранить с помощью программного кода, но то же самое можно сделать и с помощью XSL.

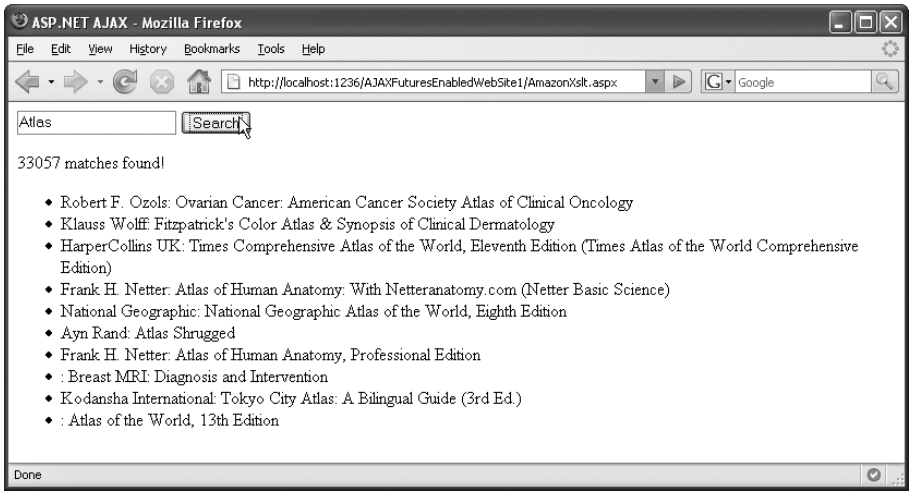


Рис. 18.5. *Поиск в каталоге Amazon с использованием моста ASP.NET AJAX и XSL-преобразования*

Использование веб-службы Yahoo! (а также REST и XPath)

В последнем примере этой главы мы используем еще одну веб-службу, на этот раз веб-службу Yahoo!. В настоящее время эта служба не предоставляет интерфейс SOAP и использует исключительно интерфейс REST.

Коротко о REST

Название REST происходит от английского «REpresential State Transfer» (передача данных о состоянии). Концепция REST детально описана Роем Филдингом (Roy Fielding) в его докторской диссертации (Университет штата Калифорния) «Architectural Styles and the Design of Network-based Software Architectures» («Архитектурные стили и проектирование архитектур сетевого ПО»). (Филдинг является одним из авторов спецификаций HTTP.) Теоретические основы REST не имеют отношения к нашим примерам, поэтому мы не будем углубляться в детали. Нам достаточно знать, что REST-интерфейс вызова веб-служб работает примерно следующим образом: серверу передается запрос GET (!), все данные запроса (какой метод вызывается, какие аргументы передаются) становятся частью строки URL. Возвращаемые данные обычно имеют формат XML, но могут использоваться и другие форматы, например простой текст, CSV или JSON.

Таким образом, на стороне сервера вместо запроса POST SOAP будет посылаться запрос GET. Пакет ASP.NET AJAX Futures поддерживает такую возможность, но сначала нам нужно зарегистрироваться на сайте Yahoo!, чтобы получить возможность использовать службу.

Yahoo! предполагает регистрацию каждого приложения и, как результат, предоставит вам несколько ключей, если вы используете несколько приложений. Чтобы получить ключ для своего приложения, перейдите на страницу *http://search.yahooapis.com/webservices/register_application*. Чтобы получить доступ к этой странице, вы должны быть зарегистрированы на сервере Yahoo!. Если у вас еще нет учетной записи Yahoo!, то вы можете получить ее совершенно бесплатно. Страница содержит анкету, как показано на рис. 18.6, где необходимо ввести основные сведения о приложении. Веб-страница, содержащая ключ приложения, показана на рис. 18.7 (перед публикацией этот рисунок был отредактирован по вполне понятным причинам).

Сохраните новый ключ в файле *Web.config* под именем `YahooApplicationID`. С учетом предыдущих примеров, файл *Web.config* теперь должен содержать раздел, который выглядит следующим образом:

```
<appSettings>
  <add key="AmazonAccessKey" value="***" />
  <add key="GoogleLicenseKey" value="***" />
  <add key="YahooAppID" value="***" />
</appSettings>
```

Строка URL вызова службы поиска Yahoo! выглядит следующим образом (на рис. 18.8 показано, как выглядит ответ в формате XML):

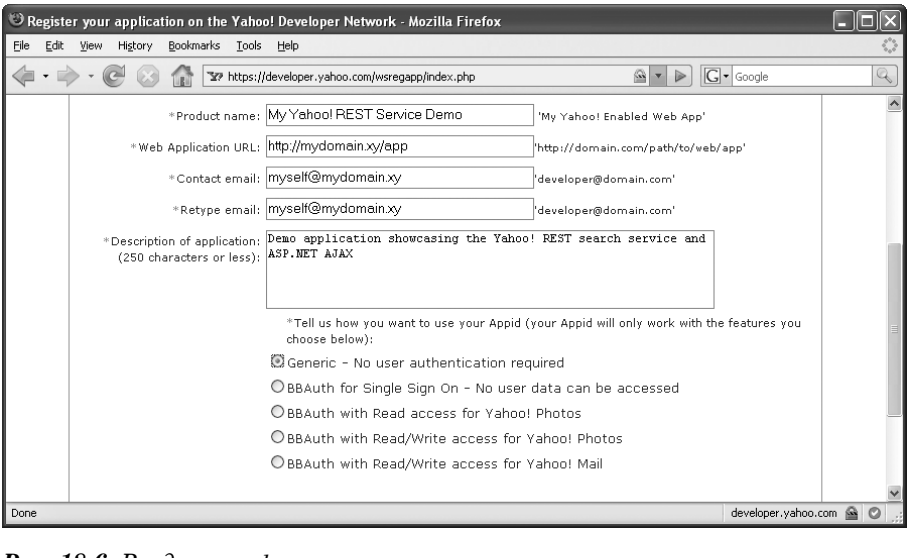


Рис. 18.6. *Введите информацию о своем приложении*

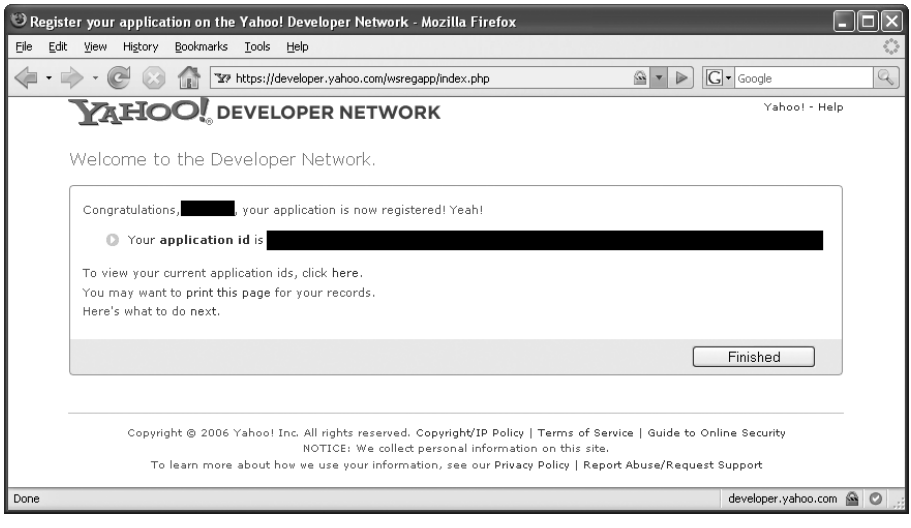


Рис. 18.7. *Вновь созданный ключ приложения (по очевидным причинам сам ключ здесь не показан)*

```
http://api.search.yahoo.com/WebSearchService/V1/
webSearch?appid=***&output=xml&query=***
```

Эта строка URL содержит базовую часть (*http://api.search.yahoo.com/WebSearchService/V1/webSearch*) и ряд аргументов, входящих в строку запроса. Мост для этого примера (в файле *.asbx*) настроен на отправку запросов приложения по этому URL.

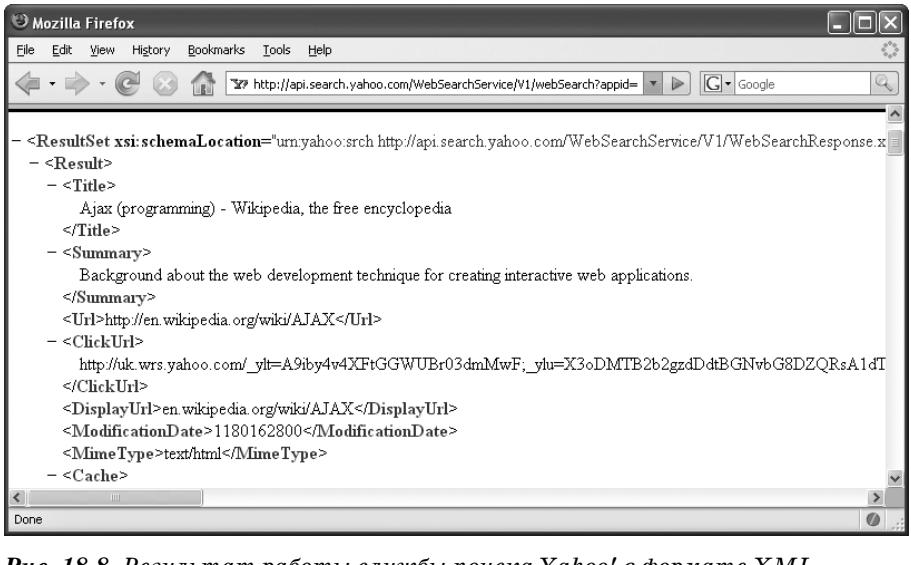


Рис. 18.8. *Результат работы службы поиска Yahoo! в формате XML*

Для начала нужно выбрать корректный тип класса-посредника. На этот раз нам не нужно использовать класс-обертку для работы с интерфейсом SOAP, созданный на основе WSDL-описания службы, но нам потребуется провайдер, который входит в состав ASP.NET AJAX Futures: `Microsoft.Web.Preview.Services.BridgeRestProxy`. Свойство `serviceURL` элемента `<proxy>` должно содержать базовую часть URL службы поиска Yahoo!:

```
<?xml version="1.0" encoding="utf-8" ?>
  <bridge namespace="OReilly.AspNetAJAX" className="Yahoo">
    <proxy type="Microsoft.Web.Preview.Services.BridgeRestProxy"
serviceUrl="http://api.search.yahoo.com/WebSearchService/V1/webSearch" />
```

Как и в предыдущих примерах, метод определяется с помощью элемента `<method>`. Значения аргументов `appid` и `output` будут определяться на стороне сервера, а аргумент `query` (фактическая строка поиска) будет предоставляться сценарием JavaScript:

```
<method name="Search">
  <input>
    <parameter name="appid"
               value="% appsettings : YahooAppID %"
               serverOnly="true" />
    <parameter name="output"
               value="xml"
               serverOnly="true" />
    <parameter name="query" />
</input>
```

Этот мост фактически уже может работать. Если вызвать метод `OReilly.AspNetAJAX.Yahoo.Search()`, то будет получен XML-документ, напоминающий тот, что показан на рис. 18.8. После этого XML-документ можно проанализировать в JavaScript и отобразить данные в веб-странице.

Синтаксический анализ XML-документа в JavaScript не только может оказаться достаточно утомительной задачей, которая не одинаково хорошо будет работать во всех типах броузеров, но и просто здесь не нужен. В нашем распоряжении имеется еще одна удобная возможность ASP.NET AJAX Futures – `XPathBridgeTransformer`. Этот подход чем-то напоминает `XsltBridgeTransformer`, но на этот раз XML-документ преобразуется платформой ASP.NET AJAX не в HTML, а в объект JavaScript.

Основная идея заключается в использовании запросов `XPath` для доступа к конкретным узлам XML-документа. Результаты запросов `XPath` преобразуются в объекты JavaScript и могут использоваться в клиентском сценарии.

Как правило, для этого потребуется выполнить следующие действия:

1. Использовать `<attribute name="selector" value="XPath expression" />`, чтобы запросить один или более узлов XML-документа.

2. Необязательно: Если необходимо использовать пространство имен, следует определить его с помощью элемента <dictionary> (в следующем примере показано, как это сделать).
3. С помощью элемента <dictionary> отобрать узлы из результата запроса XPath и преобразовать их в свойства объекта JavaScript.

Ниже показано, как выглядит XML-документ, полученный от службы поиска Yahoo! (сокращенная версия):

```
<ResultSet xsi:schemaLocation="urn:yahoo:srch
http://api.search.yahoo.com/WebSearchService/V1/WebSearchResponse.xsd">
  <Result>
    <Title>Ajax (programming) - Wikipedia, the free encyclopedia</Title>
    <Url>http://en.wikipedia.org/wiki/AJAX</Url>
  </Result>
  <Result>
  ...
  </Result>
</ResultSet>
```

Чтобы выбрать отдельные результаты, можно использовать XPath-запрос ResultSet/Result или просто Result. Однако следует учесть одну вещь. В XML-документе используется пространство имен для организации информации: urn:yahoo:srch. Это пространство имен также следует определить в файле моста. Делается это следующим образом:

```
<transforms>
  <transform type="Microsoft.Web.Preview.Services.XPathBridgeTransformer">
    <data>
      <attribute name="selector" value="y:Result" />
      <dictionary name="namespaceMapping">
        <item name="y" value="urn:yahoo:srch" />
      </dictionary>
```

Теперь префикс 'y' будет обозначать пространство имен urn:yahoo:srch. Предыдущий фрагмент выбирает все узлы Result. Теперь с помощью элемента <dictionary> можно получить доступ к конкретным узлам и свойствам. В этом примере нас интересуют названия и URL, которые описываются элементами <Title> и <Url>. Следующий фрагмент отбирает эти элементы, а также объявляет соответствующие им имена свойств JavaScript (resultTitle и resultUrl):

```
      <dictionary name="selectedNodes">
        <item name="resultTitle" value="y:Title" />
        <item name="resultUrl" value="y:Url" />
      </dictionary>
    </data>
  </transform>
</transforms>
</method>
</bridge>
```

В примере 18.15 приводится полный код разметки моста.

Пример 18.15. Мост доступа к веб-службе Yahoo! с использованием интерфейса REST

```
Yahoo.asbx

<?xml version="1.0" encoding="utf-8" ?>
<bridge namespace="OReilly.AspNetAJAX" className="Yahoo">
  <proxy type="Microsoft.Web.Preview.Services.BridgeRestProxy"
serviceUrl="http://api.search.yahoo.com/WebSearchService/V1/webSearch" />
  <method name="Search">
    <input>
      <parameter name="appid"
                 value="% appsettings : YahooAppID %"
                 serverOnly="true" />
      <parameter name="output"
                 value="xml"
                 serverOnly="true" />
      <parameter name="query" />
    </input>
    <transforms>
      <transform type="Microsoft.Web.Preview.Services.XPathBridgeTransformer">
        <data>
          <attribute name="selector" value="y:Result" />
          <dictionary name="namespaceMapping">
            <item name="y" value="urn:yahoo:srch" />
          </dictionary>
          <dictionary name="selectedNodes">
            <item name="resultTitle" value="y:Title" />
            <item name="resultUrl" value="y:Url" />
          </dictionary>
        </data>
      </transform>
    </transforms>
  </method>
</bridge>
```

Теперь мост на месте, а остальная часть приложения делается по аналогии с предыдущими примерами. Создайте новый файл ASP.NET и добавьте в него ссылки на файл *.asbx* и библиотеку Futures в элемент управления `ScriptManager`:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Services>
    <asp:ServiceReference Path="~/Yahoo.asbx" />
  </Services>
  <Scripts>
    <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                         Name="PreviewScript.js" />
  </Scripts>
</asp:ScriptManager>
```

Затем добавьте элементы пользовательского интерфейса, которые дадут пользователю возможность вводить строку поиска. По щелчку на

кнопке отправки формы необходимо выполнить следующий программный код:

```
function Search() {
  var query = new Sys.Preview.UI.TextBox($get('Query'))
  $get("Button").disabled = true;
  clearList();
  OReilly.AspNetAJAX.Yahoo.Search(
    { "query": query.get_text() },
    callComplete, callError
  );
}
```

Функция `callComplete()` автоматически получает результат обращения к веб-службе в первом аргументе. Этот аргумент представляет собой массив JavaScript со всеми результатами поиска (по умолчанию веб-служба Yahoo! возвращает десять результатов, если их не было меньше). Каждый из элементов этого массива представляет собой объект со свойствами `resultTitle` и `resultUrl`, в соответствии с определениями в файле моста. Эти данные легко извлекать и легко добавлять в страницу:

```
function callComplete(result) {
  for (var i = 0; i < result.length; i++) {
    var li = document.createElement("li");
    var a = document.createElement("a");
    a.setAttribute("href", result[i].resultUrl);
    a.innerHTML = result[i].resultTitle;
    li.appendChild(a);
    $get("Results").appendChild(li);
  }
  $get("Button").disabled = false;
}
```

В примере 18.16 приводится полный код этого примера, а на рис. 18.9 показаны результаты (обратите внимание, какое почетное место занимает домашняя страница ASP.NET AJAX при поиске по слову «Ajax» ;-)).

Пример 18.16. Вызов веб-службы Yahoo! с использованием интерфейса REST

```
Yahoo.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
  function clearList() {
    $get("Results").innerHTML = "";
  }
```

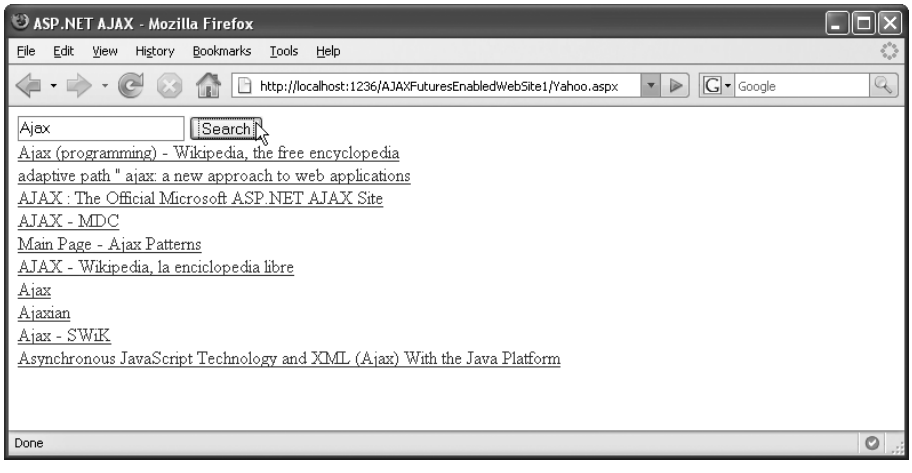


Рис. 18.9. *Результат обращения к веб-службе Yahoo!*

```
    function Search() {
      var query = new Sys.Preview.UI.TextBox($get('Query'))
      $get("Button").disabled = true;
      clearList();
      OReilly.AspNetAJAX.Yahoo.Search(
        { "query": query.get_text() },
        callComplete, callError
      );
    }

    function callComplete(result) {
      for (var i = 0; i < result.length; i++) {
        var li = document.createElement("li");
        var a = document.createElement("a");
        a.setAttribute("href", result[i].resultUrl);
        a.innerHTML = result[i].resultTitle;
        li.appendChild(a);
        $get("Results").appendChild(li);
      }
      $get("Button").disabled = false;
    }

    function callError(result) {
      window.alert("Error! " + result.get_message());
      $get("Button").disabled = false;
    }
    </script>

  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server">
        <Services>
```

```
        <asp:ServiceReference Path="~/Yahoo.asbx" />
      </Services>
      <Scripts>
        <asp:ScriptReference Assembly="Microsoft.Web.Preview"
                             Name="PreviewScript.js" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <input type="text" id="Query" />
      <input type="button" id="Button" value="Search" onclick="Search();" />
    </div>
    <div id="Results">
    </div>
  </form>
</body>
</html>
```

Одна из разновидностей веб-служб, которая достаточно часто используется для создания гибридных веб-приложений (mash-ups), – это карты. Существует множество компаний, предлагающих прикладной интерфейс доступа к географическим картам, включая Google, Yahoo! и Microsoft. В случае, если такие веб-службы предоставляют интерфейс SOAP или REST, для включения возможности отображения карт в своих веб-приложениях вы можете использовать методики, описанные в этой главе. Но самое интересное, что многие из этих служб обладают специальным интерфейсом, построенным на базе технологий Ajax: вам достаточно будет написать немного программного кода на JavaScript с HTML-разметкой, и карта будет включена в вашу страницу, причем с вашей стороны вообще не потребуется писать серверные сценарии.

Кэширование запросов на стороне моста

Возможно, у вас появится желание рассмотреть возможность кэширования запросов на стороне моста к веб-службе. Для реализации такой возможности вам достаточно будет добавить следующий элемент XML в файл моста *.asbx* (в любом месте, внутри узла `<bridge>`):

```
<caching >
  <cache type="Microsoft.Web.Preview.Services.BridgeCache" />
</caching>
```

Это поможет существенно повысить производительность обращений к веб-службе, но может оказаться и существенным препятствием на этапе разработки и тестирования, поэтому кэширование лучше отключить до того момента, пока веб-приложение не будет закончено и отлажено.

Подведение итогов

В этой главе были продемонстрированы интереснейшие возможности пакета ASP.NET AJAX Futures: вызов внешних веб-служб, преодоление ограничений системы безопасности, заложенных в объект `XMLHttpRequest`, с помощью моста на стороне сервера.

Для дополнительного чтения

http://www.amazon.com/gp/aws/landing.html

Регистрация и описание веб-службы на сайте электронной коммерции Amazon

http://www.google.com/apis

Регистрация и описание поисковой веб-службы на сайте Google

http://developer.yahoo.com

Сайт разработчиков Yahoo! API

http://www.w3schools.com/xsl/

Учебное руководство по XSLT, включающее ссылки на другие документы с описанием XSL

Fitzgerald, Michael. Learning XSLT (O'Reilly)

Прекрасное введение в технологию XSLT

http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Докторская диссертация (Университет штата Калифорния) Роя Филдинга (Roy Fielding) «Architectural Styles and the Design of Network-based Software Architectures» («Архитектурные стили и проектирование архитектур сетевого ПО»), где описываются принципы REST

19

Использование анимационных эффектов

Плавные переходы между страницами или элементами радуют глаз, но их реализация может быть весьма нетривиальной, и иногда эффект достигается только при использовании целого ряда преобразований. Например, можно добиться визуального изменения прозрачности элемента или его положения за счет постепенного изменения значений соответствующих параметров, создав таким образом иллюзию анимации. Если использовать число, изменяющееся в диапазоне от 0 до 100, в качестве значения степени непрозрачности элемента, то возникает эффект появления элемента, когда он из полностью прозрачного становится полностью непрозрачным.

К счастью, в ASP.NET AJAX Futures имеется встроенная реализация нескольких анимационных эффектов. Эта реализация находится в библиотеке *PreviewGlitz.js* (включена в сборку пакета Futures).

В этой главе вы узнаете, как можно использовать анимационные эффекты ASP.NET AJAX для изменения степени прозрачности элементов и их положения. Кроме этого, вы узнаете о других анимационных эффектах и работе с ними.

Использование анимационных эффектов

Поскольку реализация анимационных эффектов находится во внешней библиотеке, файл *PreviewGlitz.js* придется подключать вручную ко всем страницам, где эти эффекты используются. Кроме того, этот файл зависит от файла *PreviewScript.js* – «базовой» библиотеки пакета Futures. Существует несколько возможных способов подключения этого файла. Лучший заключается в использовании элемента ASP.NET AJAX `ScriptReference`, как показано в следующем фрагменте:

```
<asp:ScriptManager runat="server" ID="ScriptManager1">
  <Scripts>
    <asp:ScriptReference Name="PreviewGlitz.js"
                         Assembly="Microsoft.Web.Preview" />
  </Scripts>
</asp:ScriptManager>
```

В табл. 19.1 перечислены анимационные эффекты, реализованные в файле *PreviewGlitz.js*.

Таблица 19.1. Анимационные эффекты, реализованные в библиотеке PreviewGlitz.js

Анимационный эффект	Описание
Sys.Preview.UI.Effects.PropertyAnimation	Изменяет значение свойства (например, расстояние от левого или верхнего края) элемента
Sys.Preview.UI.Effects.InterpolatedAnimation	Изменяет значение свойства и интерполирует (вычисляет) значения для промежуточных шагов анимации
Sys.Preview.UI.Effects.DiscreteAnimation	Изменяет значение в соответствии с заданным списком значений
Sys.Preview.UI.Effects.NumberAnimation	Изменяет числовое значение
Sys.Preview.UI.Effects.ColorAnimation	Изменяет цвет элемента
Sys.Preview.UI.Effects.LengthAnimation	Изменяет число и округляет до целого значения на каждом промежуточном шаге анимации
Sys.Preview.UI.Effects.CompositeAnimation	Объединяет несколько анимационных эффектов в одно целое
Sys.Preview.UI.Effects.FadeAnimation	Изменяет степень прозрачности элемента

Все эти анимационные эффекты можно использовать декларативно – в xml-script, и большинство из них доступны программно. В следующих примерах вы познакомитесь с обоими способами.

Каждый анимационный эффект имеет метод `play()`, запускающий анимацию. В своей работе этот метод использует несколько свойств, определенных в классе. Наибольший интерес для нас представляют следующие три свойства:

`_duration`

Продолжительность анимации (в секундах)

`_fps`

Число шагов анимации (кадров) в секунду

_target

Целевой элемент анимации

Всякий раз, когда выполняется очередной шаг анимации, вызывается метод setValue(). Выполняемые этим методом действия зависят от его реализации. Это может быть метод отдельного анимационного эффекта или метод setValue() класса Sys.Preview.UI.Effects.Animation, базового для всех анимационных эффектов. В зависимости от вида анимации, реализация метода может производить весьма сложные вычисления или просто переходить к следующему элементу массива.

Для реализации альфа-прозрачности (графическая концепция, определяющая степень прозрачности, которая позволяет реализовать такие эффекты, как полупрозрачность) Internet Explorer использует фильтр DirectX – DXImageTransform.Microsoft.Alpha, в то время как остальные броузеры, такие как Mozilla, Firefox и другие, обладают встроенной поддержкой эффекта прозрачности.

Использование анимации для создания эффекта плавного исчезновения

Существует возможность создавать довольно интересный эффект плавного исчезновения, изменяя степень прозрачности элемента. Для начала рассмотрим реализацию этого эффекта программным способом. В функции pageLoad() создадим новый объект Sys.Preview.UI.Effects.FadeAnimation:

```
var ani = new Sys.Preview.UI.Effects.FadeAnimation();
```

Затем, в качестве целевого, устанавливаем элемент метки (элемент), который будет присутствовать на странице:

```
ani.set_target($get("Label1").control);
```

По умолчанию эффект этой анимации заключается в плавном исчезновении элемента. Однако в перечислении Sys.Preview.UI.Effects.FadeEffect определены два варианта: FadeIn и FadeOut, которые могут передаваться методу set_effect() для изменения эффекта:

```
ani.set_effect(Sys.Preview.UI.Effects.FadeEffect.FadeOut);
```

Затем определим протяженность анимации по времени. По умолчанию протяженность составляет одну секунду – в следующем фрагменте эта величина утраивается:

```
ani.set_duration(3);
```

И в заключение мы запускаем эффект анимации:

```
ani.play();
```

Полный код страницы приводится в примере 19.1.

Пример 19.1. Использование эффекта плавного исчезновения

```
FadeAnimation.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="JavaScript" type="text/javascript">
    function pageLoad() {
      var ani = new Sys.Preview.UI.Effects.FadeAnimation();
      ani.set_target($get("Label1").control);
      ani.set_effect(Sys.Preview.UI.Effects.FadeEffect.FadeOut);
      ani.set_duration(3);
      ani.play();
    }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server" ID="ScriptManager1">
      <Scripts>
        <asp:ScriptReference name="PreviewScript.js"
                             assembly="Microsoft.Web.Preview" />
        <asp:ScriptReference name="PreviewGlitz.js"
                             assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <label id="Label1"
             style="display: inline-block; background-color: Red;">
        See me fading ...</label>
    </div>
  </form>

  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <label id="Label1" />
      </components>
    </page>
  </script>
</body>
</html>
```

Обратите внимание на используемый стиль CSS `display: inline-block`. Без этого объявления Internet Explorer не будет воспроизводить анимационный эффект (по причинам, которые я так и не смог определить).

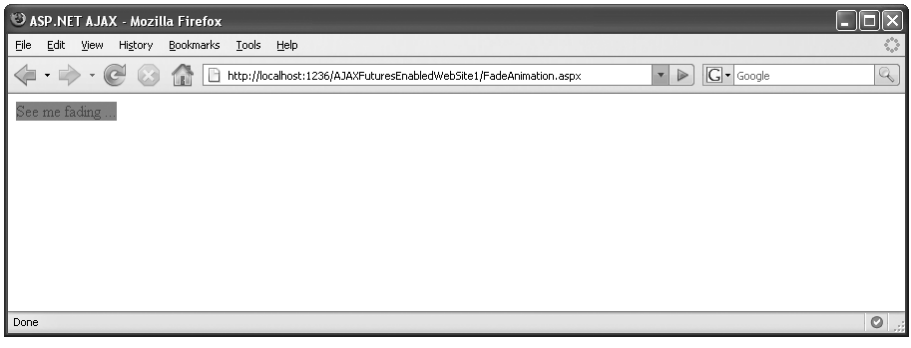


Рис. 19.1. *Метка постепенно исчезает, сливаясь с цветом фона*

После загрузки страницы элемент будет плавно исчезать в течение трех секунд. На рис. 19.1 показано, как выглядит исчезающая метка.

Естественно, этот эффект может быть реализован и декларативным способом. Как всегда, создается элемент xml-script, имя которого повторяет имя класса, с соблюдением правил использования символов верхнего и нижнего регистров, вследствие чего имя класса `FadeAnimation` превращается в элемент `<fadeAnimation>`. Очень важно определить значение атрибута ID этого элемента, потому что позднее он потребуется, чтобы запустить анимацию.

Запустить анимацию можно не только из программного кода, но и с помощью xml-script, как показано ниже:

```
<application>
  <load>
    <invokeMethodAction target="ani" method="play" />
  </load>
</application>
```

Этот подход достаточно подробно был описан в главе 17.

В примере 19.2 приводится полный код страницы, где наиболее важные элементы выделены жирным шрифтом. В этом примере мы явно не задаем продолжительность, поэтому анимация будет длиться в течение одной секунды.

Пример 19.2. Реализация анимационного эффекта с использованием xml-script

```
FadeAnimationDeclarative.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
```

```
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server" ID="ScriptManager1">
      <Scripts>
        <asp:ScriptReference name="PreviewScript.js"
                             assembly="Microsoft.Web.Preview" />
        <asp:ScriptReference name="PreviewGlitz.js"
                             assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
    <label id="Label1" style="display: inline-block; background-color: Red;">
    See me fading ...</label>
    </div>
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <label id="Label1" />
        <fadeAnimation id="ani" target="Label1" effect="FadeOut" />
        <application>
          <load>
            <invokeMethodAction target="ani" method="play" />
          </load>
        </application>
      </components>
    </page>
  </script>
</body>
</html>
```

Использование анимации для перемещения элемента

Эффект изменения степени прозрачности элемента может использоваться лишь для узкого круга задач. Другая (более универсальная) разновидность анимации, реализованная платформой ASP.NET AJAX, – та, что наращивает значение числа с определенным шагом. Возможность изменения числового значения может использоваться разными способами – обычно для изменения значения свойства элемента. Один из примеров, который сразу приходит на ум, – это непрерывное изменение значений свойств `left` и `top`.

Класс ASP.NET AJAX `Sys.Preview.UI.Effects.NumberAnimation` производит изменение числа от начального значения до конечного. Определяя протяженность анимации и количество кадров в секунду, вы задаете количество промежуточных шагов и длительность анимации.

В качестве примера мы опять возьмем элемент управления `Label`. Программный код создает экземпляр класса `Sys.Preview.UI.Effects.NumberAnimation` и устанавливает все необходимые свойства, кроме числа кадров в секунду, – в этом случае по умолчанию используется значение 25.

```
var ani = new Sys.Preview.UI.Effects.NumberAnimation();
ani.set_target($get("Label1").control);
ani.set_startValue(0);
ani.set_endValue(300);
ani.set_duration(3);
ani.set_integralValues(true);
```

В данном случае анимационный эффект продолжается три секунды, со скоростью 25 кадров в секунду, следовательно, на каждом шаге число увеличивается на 4. (25 кадров в секунду на протяжении трех секунд дает 75 шагов анимации, а поскольку число изменяется от 0 до 300, получаем, что на каждом шаге оно увеличивается на 4.) Таким образом, все получаемые значения будут целыми числами. Однако бывают случаи, когда при заданной продолжительности и величине интервала промежуточные результаты не являются целыми значениями. Поскольку координаты метки задаются только целыми числами, получающиеся промежуточные значения должны округляться. Класс `NumberAnimation` предоставляет такую возможность с помощью свойства `integralValues`.

Поскольку класс `NumberAnimation` является универсальным, в нем не делается никаких предположений относительно того, как будут использоваться промежуточные значения, поэтому он не реализует метод, который можно было бы использовать непосредственно для преобразования числовых значений в свойство `element`. Вместо этого необходимо присвоить свойству `setValue` класса `NumberAnimation` функцию, которая будет выполнять необходимые действия. У такого подхода есть свои преимущества – вы можете манипулировать числовыми значениями, как вам требуется. Например, некоторые броузеры (к ним относятся броузеры, основанные на Mozilla) принимают для позиционирования только величины, включающие единицы измерения, такие как `"20px"` вместо обычного числа `"20"`, таким образом ваша функция `setMethod()` может добавлять к числам единицы измерения.

Единственная сложность заключается в получении ссылки на элемент, который должен быть анимирован, без необходимости написания специального программного кода (например, с помощью `document.getElementById()` или `$get()` и фиксированного идентификатора). Класс анимации позволяет получить ссылку на целевой объект с помощью метода `get_target()`, а свойство `element` результата обеспечит доступ к ассоциированному DOM-элементу (с помощью метода `get_element()`). Вы можете использовать эту ссылку в своей реализации функции `setValue()` и затем запустить анимацию. Ваш код мог бы выглядеть примерно так:

```
ani.setValue = function(value) {
  this.get_target().get_element().style.left = value + "px";
  this.get_target().get_element().style.top = value + "px";
}
ani.play();
```

Вместо установки позиции метки вручную можно было бы использовать метод `Sys.UI.DomElement.setLocation()`, предоставляемый платформой ASP.NET AJAX, который, кроме всего прочего, заботится о подстановке единиц измерения:

```
ani.setValue = function(value) {
  Sys.UI.DomElement.setLocation(
    this.get_target().get_element(),
    value,
    value);
}
ani.play();
```

В примере 19.3 приводится полный код страницы, которая анимирует элемент управления `Label`, перемещая его по странице.

Пример 19.3. Перемещение элемента с помощью анимационного эффекта

```
NumberAnimation.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
  <script type="text/javascript">
    function pageLoad() {
      var ani = new Sys.Preview.UI.Effects.NumberAnimation();
      ani.set_target($get("Label1").control);
      ani.set_startValue(0);
      ani.set_endValue(300);
      ani.set_duration(3);
      ani.set_integralValues(true);
      ani.setValue = function(value) {
      Sys.UI.DomElement.setLocation(
        this.get_target().get_element(),
        value,
        value);
      }
      ani.play();
    }
  </script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server" ID="ScriptManager1">
      <Scripts>
        <asp:ScriptReference name="PreviewScript.js"
                             assembly="Microsoft.Web.Preview" />
        <asp:ScriptReference name="PreviewGlitz.js"
                             assembly="Microsoft.Web.Preview" />
```

```
        </Scripts>
      </asp:ScriptManager>
      <div>
        <label id="Label1" style="background-color: Red; position: relative;">
          See me moving ...</label>
      </div>
    </form>
    <script type="text/xml-script">
      <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
        <components>
          <label id="Label1" />
        </components>
      </page>
    </script>
  </body>
  </html>
```

Как только страница будет полностью загружена, элемент `label` начинает перемещаться по странице под углом 45 градусов. Обратите внимание, как используется свойство CSS `position: relative` для обеспечения такой возможности – вам необходимо только задать позицию элемента. Результат работы страницы приводится на рис. 19.2.

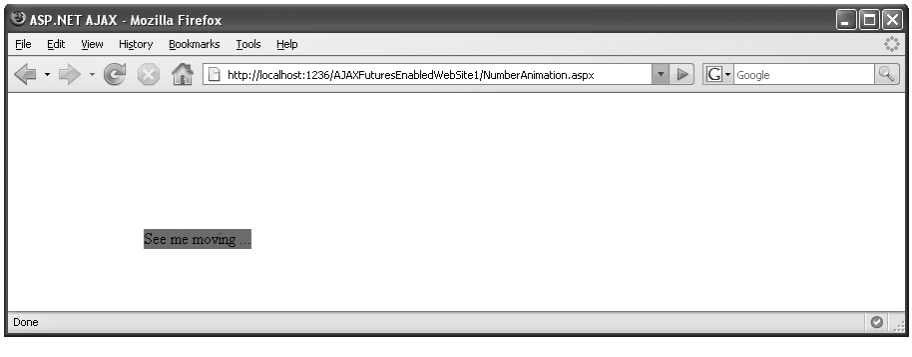


Рис. 19.2. *Элемент перемещается по экрану*

Использование LengthAnimation для перемещения элемента

Предыдущий пример также может быть реализован декларативно. Как отмечалось ранее, чтобы гарантировать воспроизведение анимационного эффекта во всех броузерах, свойствам `top` и `left` должны присваиваться значения, содержащие единицы измерения. При использовании класса `NumberAnimation` это возможно только в случае создания своего собственного метода `setValue()`.

Однако платформа ASP.NET AJAX предоставляет еще один класс – `LengthAnimation`, который способен решить эту задачу более непосредственным способом.

Этот класс похож на класс NumberAnimation, за исключением следующих двух отличий:

- Получаемое на каждом шаге значение всегда округляется.
- Значение свойства unit (по умолчанию "px") добавляется к числовому значению.

Таким образом, для организации перемещения элемента класс LengthAnimation выглядит «предпочтительнее», чем класс NumberAnimation из предыдущего примера. Хотя оба класса прекрасно справляются с задачей, и потому оба здесь приведены.

Тем не менее использование класса LengthAnimation для перемещения элемента Label имеет свои особенности. Свойства left и top являются частью стиля элемента, который напрямую не доступен, как обычные свойства. Однако доступ к стилевой информации, а следовательно, и к этим двум значениям, может быть получен с помощью аспекта поведения <layoutBehavior>.

Еще один полезный аспект поведения – <opacityBehavior>, который может использоваться для управления прозрачностью элемента, если у вас появится необходимость самостоятельно реализовать эффект плавного исчезновения или других эффектов, связанных с изменением видимости элемента.

Для организации перемещения элемента Label по странице добавьте аспект поведения <layoutBehavior> к метке в xml-script и присвойте ему свой идентификатор:

```
<label id="Label1">
  <behaviors>
    <layoutBehavior id="Label1Style" />
  </behaviors>
</label>
```

Затем создайте элемент анимации, а лучше два, поскольку нам требуется изменять два стилевых свойства:

```
<lengthAnimation id="ani1" target="Label1Style" duration="3" property="left"
  startValue="0" endValue="300" />
<lengthAnimation id="ani2" target="Label1Style" duration="3" property="top"
  startValue="0" endValue="300" />
```

В разделе <application><load> необходимо запустить оба эффекта анимации. В примере 19.4 приводится получившийся код страницы.

Пример 19.4. Создание эффекта перемещения элемента с помощью xml-script

```
LengthAnimation.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager runat="server" ID="ScriptManager1">
      <Scripts>
        <asp:ScriptReference name="PreviewScript.js"
                             assembly="Microsoft.Web.Preview" />
        <asp:ScriptReference name="PreviewGlitz.js"
                             assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <label id="Label1" style="background-color: Red; position: relative;">
        See me moving ...</label>
    </div>
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <label id="Label1">
          <behaviors>
            <layoutBehavior id="Label1Style" />
          </behaviors>
        </label>
        <lengthAnimation id="ani1" target="Label1Style" duration="3"
          property="left" startValue="0" endValue="300" />
        <lengthAnimation id="ani2" target="Label1Style" duration="3"
          property="top" startValue="0" endValue="300" />
        <application>
          <load>
            <invokeMethodAction target="ani1" method="play" />
            <invokeMethodAction target="ani2" method="play" />
          </load>
        </application>
      </components>
    </page>
  </script>
</body>
</html>
```

Группировка анимационных эффектов

Когда желаемое достигается за счет одновременного привлечения более чем одного анимационного эффекта, результат, который дает разметка, не всегда получается таким, как ожидалось: вы получаете серию анимационных эффектов, которые запускаются по очереди (в то время, как предполагалось получить параллельное исполнение). В предыдущем примере (пример 19.4) описываются две разных анимации, одна

из них изменяет горизонтальную координату, а вторая – вертикальную, и каждую из них пришлось определить отдельно, включая продолжительность действия эффекта.

Однако ситуацию можно упростить, сгруппировав анимационные эффекты с помощью класса `Sys.Preview.UI.Effects.CompositeAnimation`. Группировка позволяет гарантировать параллельное исполнение объединенных анимационных эффектов.

Группировка может быть выполнена с помощью элемента xml-script `<compositeAnimation>`. Внутрь этого элемента добавляется элемент `<animations>`, в пределах которого вставляются определения xml-script для всех анимационных эффектов, которые должны исполняться одновременно. После этого можно определить значения атрибутов `id` и `duration` для элемента `<compositeAnimation>`, которые будут относиться ко всей группе целиком:

```
<compositeAnimation id="ani" duration="3">
  <animations>
    <lengthAnimation target="Label1Style" property="left"
                     startValue="0" endValue="300" />
    <lengthAnimation target="Label1Style" property="top"
                     startValue="0" endValue="300" />
    <fadeAnimation target="Label1" effect="FadeOut" />
  </animations>
</compositeAnimation>
```

Запустить такой композитный эффект можно с помощью элемента `<invokeMethod>`:

```
<application>
  <load>
    <invokeMethodAction target="ani" method="play" />
  </load>
</application>
```

В примере 19.5 приводится полный код страницы, содержащей сгруппированный анимационный эффект.

Пример 19.5. Группировка анимационных эффектов, воспроизводимых на странице

```
CompositeAnimation.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
```

```
    <asp:ScriptManager runat="server" ID="ScriptManager1">
      <Scripts>
        <asp:ScriptReference name="PreviewScript.js"
                             assembly="Microsoft.Web.Preview" />
        <asp:ScriptReference name="PreviewGlitz.js"
                             assembly="Microsoft.Web.Preview" />
      </Scripts>
    </asp:ScriptManager>
    <div>
      <label id="Label1" style="display: inline-block; background-color: Red;
             position: relative;">
        See me fading and moving ...</label>
    </div>
  </form>
  <script type="text/xml-script">
    <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <components>
        <label id="Label1">
          <behaviors>
            <layoutBehavior id="Label1Style" />
          </behaviors>
        </label>
        <compositeAnimation id="ani" duration="3">
          <animations>
            <lengthAnimation target="Label1Style" property="left"
                             startValue="0" endValue="300" />
            <lengthAnimation target="Label1Style" property="top"
                             startValue="0" endValue="300" />
            <fadeAnimation target="Label1" effect="FadeOut" />
          </animations>
        </compositeAnimation>
        <application>
          <load>
            <invokeMethodAction target="ani" method="play" />
          </load>
        </application>
      </components>
    </page>
  </script>
</body>
</html>
```

Данная группа объединяет три отдельных анимационных эффекта, заканчивающихся одновременно (рис. 19.3):

- Плавное исчезновение элемента
- Перемещение элемента вправо
- Перемещение элемента вниз

Несмотря на то, что в реальности круг применения анимационных эффектов не очень велик, платформа ASP.NET AJAX создает очень удоб-

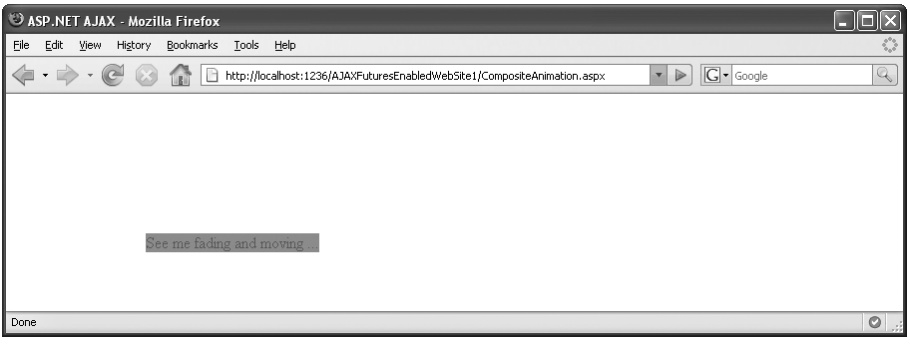


Рис. 19.3. *Метка перемещается и одновременно плавно исчезает*

ную среду для добавления некоторых оживляющих визуальных эффектов в веб-приложения. Поскольку эти возможности реализованы в отдельном файле JavaScript, сама библиотека *PreviewScript.js* не перегружена поддержкой данной функциональности по умолчанию.

Подведение итогов

Пакет ASP.NET AJAX Futures предлагает несколько анимационных эффектов, которые могут использоваться для оживления или изменения элементов. Все анимационные эффекты могут быть применены как программно, так и декларативно, что обеспечивает максимальную гибкость в процессе разработки.

Для дополнительного чтения

http://blogs.msdn.com/phaniraj/archive/2007/03/15/howto-sys-preview-ui-effectsfadeanimation.aspx

Статья в блоге об эффекте `FadeAnimation`

20

Кнопки Назад/Вперед и закладки

Ajax-приложения обладают многими неоспоримыми преимуществами перед «классическими» веб-приложениями, но не следует забывать и о недостатках, которые обусловлены самой концепцией. Один из самых главных недостатков заключается в том, что такие приложения не будут работать без поддержки JavaScript и без использования достаточно современных броузеров. По некоторым подсчетам, это исключает от 5 до 10 процентов пользователей (это число может существенно отличаться в зависимости от целевой аудитории). Одни из самых досадных недостатков – это несовершенство поддержки закладок («избранное» – в Internet Explorer) и кнопок Вперед/Назад – двух фундаментальных особенностей в броузерах.

Нарушение таких традиционных понятий не только расценивается пользователями как неуважение, но и снижает ценность приложения. В 2006 году был запущен сайт Live.com. Модуль поисковой машины этого сайта использовал массу эффектов, свойственных технологиям Ajax. Так много, что кнопка Назад оказалась ненужной. В ответ незамедлительно последовала негативная реакция общественности, и в результате поддержка кнопки Назад была добавлена в течение нескольких дней. Это потребовало написания значительного объема хитроумного программного кода на JavaScript и привело к появлению щелкающих звуков в Internet Explorer (из-за скрытого фрейма, содержимое которого приходилось перезагружать в отдельных случаях). В ответ последовала новая волна отрицательных откликов, и наконец, в третьей версии раздражающие недостатки были ликвидированы. Теперь Live.com работает, как ожидается, и обеспечивает надлежащую поддержку закладок и кнопок Вперед и Назад.

Причина отсутствия возможности устанавливать закладки кроется в том, что Ajax-приложения изо всех сил стремятся избежать полной перезагрузки страницы. Состояние одной и той же страницы может

изменяться, но URL – нет. Поэтому, как правило, невозможно установить закладку на страницу, обладающую поддержкой Ajax, так как состояние этой страницы не является частью строки URL. В результате кнопки Назад и Вперед в веб-броузерах также оказываются неработоспособными. Навигация происходит только между разными URL. Но строка URL не меняется, и поэтому в журнале броузера не создаются новые записи, что делает невозможным перемещение с помощью этих кнопок.

Поскольку эти две проблемы тесно связаны между собой, так же тесно связаны и их решения. Суть состоит в том, чтобы изменять строку URL всякий раз, когда изменяется состояние страницы. Но изменение строки URL в большинстве случаев приводит к перезагрузке страницы. Поэтому данный эффект было бы желательно устранить в первую очередь. Единственное исключение из этого правила – изменение хеша URL (часть строки URL, расположенная после символа #). Например, если предположить, что была загружена страница *Page.aspx* и сценарий JavaScript преобразовал строку URL в *Page.aspx#anyData*, то перезагрузки страницы не произойдет (в некоторых броузерах будет выполнена прокрутка страницы в самое начало, и от этого эффекта избавиться очень трудно).

Исправление недостатков программным способом

Понимание причины – отсутствие изменений в строке URL – приводит к простому решению проблемы установки закладок: всякий раз, когда изменяется текущее состояние страницы, информация о состоянии должна помещаться в хеш строки URL текущей страницы. Всякий раз, когда страница загружается после использования закладки, к странице должны применяться данные из хеша URL, чтобы привести ее в состояние, соответствующее моменту установки закладки. Для этого потребуются две части программного кода на JavaScript. Первая часть, что приводится ниже, сохраняет информацию об изменениях состояния странице в хеше:

```
function somethingChanged() {
  location.hash = "#" + getCurrentPageState();
}
```

Вторая часть запускается при первоначальной загрузке страницы. Если в хеше строки URL присутствует какая-либо информация, она должна быть применена к текущей странице:

```
function pageLoad() {
  if (location.hash.length > 0) {
    applyPageState(location.hash);
  }
}
```

Вам необходимо реализовать функции `getCurrentPageState()` и `applyPageState()`, что может превратиться в труднопреодолимое препятствие. Основная сложность заключается в том, как сериализовать текущее состояние страницы и поместить его в строку URL, а затем выполнить обратное преобразование и применить информацию о состоянии к странице. Задача не имеет универсального решения, но вышеприведенный подход к ее решению – общий для всех приложений.

Предупреждение об опасности: ловушки с закладками

При проверке степени безопасности страниц, использующих методику сериализации состояния, я нередко обнаруживаю, что для этих целей используется формат JSON. В некоторых веб-страницах, поддерживающих закладки, сериализация состояния страницы выполнена посредством записи элементов страницы и их значений или содержимого в виде строки формата JSON, примерно так, как показано ниже:

```
{"TextBox1":"My Data","Label1":"Some other data","DropDownList1":"2"}
```

Затем эти данные кодируются в формат URL (с помощью JavaScript-функции `escape()`) и добавляются в конец строки URL. До этого места данный подход еще не вызывает никаких проблем. Сложности начинаются, когда дело доходит до восстановления состояния страницы в соответствии с данными, находящимися в строке URL. Самый простой способ снова преобразовать строку JSON в объект JavaScript – это воспользоваться функцией `eval()`, как показано ниже:

```
var o = eval("(" + location.hash.substring(1) + ")");
```

Потратим минуту времени на изучение этого кода. Данные из строки URL интерпретируются как программный код на JavaScript. Это классический пример атаки типа межсайтовый скриптинг (Cross-Site Scripting – XSS), представляющей серьезную угрозу. Атакующий может разработать вредоносный программный код на JavaScript, добавить его в конец URL страницы и затем передать этот URL жертве. При обращении по такому URL броузер жертвы выполнит вредоносный код JavaScript.

Вы всегда должны исходить из предположения, что информация в строке URL может быть изменена злоумышленником. Поэтому не используйте ее для помещения важной или секретной информации и будьте особенно осторожны при применении этой информации к странице.

Исправить недостатки в поддержке кнопок Вперед и Назад несколько сложнее. Дело в том, что здесь броузеры ведут себя по-разному. Броузеры семейства Mozilla автоматически создают новую запись в журнале посещений при изменении хеша в строке URL.

Например, представим, что Ajax-приложение последовательно загружает три URL: *Page.aspx#1*, *Page.aspx#2* и *Page.aspx#3*. Если текущей является страница *Page.aspx#3* и пользователь щелкнет на кнопке Назад, броузер загрузит страницу *Page.aspx#2*. Здесь фактически изменяется только строка URL, а сама страница не перезагружается, поэтому функция `pageLoad()` не выполняется. Все, что остается делать, – это периодически проверять строку URL и при ее изменении применять новую информацию к странице.

В случае с Internet Explorer придется выполнить ряд дополнительных действий. Броузер компании Microsoft создает новую запись в журнале посещений, только когда отправляется HTTP-запрос. Изменение хеша строки URL не порождает такого запроса, отправки которого нам, конечно, хотелось бы избежать, поскольку это вступает в противоречие с одной из наших основных целей – избегать полной перезагрузки страниц.

Так как это довольно распространенная проблема, существует и способ ее решения. Если страница содержит плавающий фрейм (HTML-элемент `<iframe>`, который мы собираемся использовать), и этот фрейм отправляет HTTP-запрос, то информация об этом запросе записывается в глобальный журнал посещений броузера для содержащей запрос страницы.

Это ведет нас прямиком к решению проблемы с кнопками Вперед/Назад. Для Internet Explorer сценарий JavaScript динамически создает HTML-элемент `<iframe>` и добавляет его в страницу. С помощью CSS этот фрейм делается невидимым, чтобы избежать появления посторонних и ненужных визуальных эффектов.

Всякий раз, когда изменяется состояние страницы, в плавающий фрейм загружается новая страница, благодаря чему создается новая запись в журнале посещений броузера. Теперь, когда пользователь щелкнет на кнопке броузера Назад или Вперед, плавающий фрейм загрузит предыдущую или следующую страницу. После этого сценарий на JavaScript определит факт загрузки новой страницы и приведет страницу к соответствующему состоянию.

Программный код JavaScript, необходимый для реализации такого решения, далеко не тривиален. Более того, большинство версий Internet Explorer издают звук щелчка при загрузке новой страницы, что может раздражать некоторых пользователей. Однако для большинства Ajax-приложений полученный результат стоит затраченных усилий.

Исправление недостатков в поддержке механизма закладок и кнопок Назад/Вперед с помощью элемента управления UpdateHistory

Платформа ASP.NET AJAX предоставляет удивительные возможности JavaScript, не обрекая вас при этом на разработку больших объемов программного кода. Таким образом, как вы уже догадались, исправление недостатков в поддержке механизма закладок и кнопок Назад/Вперед может быть реализовано за счет использования внешнего модуля и не потребует создания сценариев на JavaScript.

Сотрудник компании Microsoft Нихил Котхари (Nikhil Kothari) (архитектор команды разработчиков ASP.NET AJAX, отец языка программирования Script#, известный своим весьма информативным блогом *http://nikhilk.net/*) написал элемент управления, который будет здесь как нельзя кстати. Элемент управления `UpdateHistory` ничего не делает автоматически, но он действительно принимает на себя самые обременительные аспекты, включая весь программный код на JavaScript. Кроме того, существует еще один компонент в пакете Futures Release, который будет рассматриваться в разделе «Исправление недостатков в поддержке механизма закладок и кнопок Назад/Вперед с помощью элементов управления из пакета ASP.NET AJAX Futures» ниже в этой главе.

К моменту написания этих строк последней версией элемента `UpdateHistory` Нихила Котхари была версия 1.1. Страницу, посвященную этой версии, вы можете найти в блоге по адресу: *http://www.nikhilk.net/UpdateControls11.aspx*. С выходом новой версии запись в блоге будет указывать на новую версию модуля.

Когда вы загрузите код модуля Котхари, отыщите файл *nStuff.UpdateControls.dll* в каталоге *bin*. Скопируйте его в каталог *bin* своего сайта, построенного на базе ASP.NET AJAX. Как вариант, вы можете скопировать этот файл в глобальный кэш сборок (Global Assembly Cache – GAC), так как этот элемент управления даже подписан.

Затем создайте новый файл *.aspx*. Добавьте в него следующую директиву, которая загружает сборку и определяет префикс для тегов:

```
<%@ Register Assembly="nStuff.UpdateControls" TagPrefix="nk"
Namespace="nStuff.UpdateControls" %>
```

Помимо всего прочего, эта сборка включает в себя элемент управления `UpdateHistory`. Это невизуальный элемент управления, который способен обрабатывать записи в журнале посещений и предоставляет возможность различать варианты загрузки страницы, отличая загрузку впервые (в результате ввода URL в адресную строку броузера или щелчка на ссылке) от загрузки при переходе по кнопкам Назад или Вперед. Следующая разметка добавляет элемент управления в страницу и подготавливает механизм обработки событий к загрузке страницы:

```
<nk:UpdateHistory ID="UpdateHistory1" runat="server"
                  OnNavigate="historyNavigate" />
```

Элемент управления UpdateHistory предоставляет возможность обработки событий исключительно на стороне сервера. Чтобы обратиться к серверному программному коду без полной перезагрузки страницы, следует использовать элемент управления UpdatePanel.

С целью демонстрации воспользуемся элементом управления ASP.NET 2.0 Wizard. При размещении его внутри UpdatePanel кнопки Назад и Вперед не работают, и потому невозможно выполнить переход на шаг назад или вперед с помощью панели инструментов броузера.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
  <ContentTemplate>
    <asp:Wizard ID="Wizard1" runat="server" OnActiveStepChanged="stepChanged">
      <WizardSteps>
        <asp:WizardStep runat="server" Title="Step 1">
          <h1>Ready ...</h1>
        </asp:WizardStep>
        <asp:WizardStep runat="server" Title="Step 2">
          <h1>Set ...</h1>
        </asp:WizardStep>
        <asp:WizardStep runat="server" Title="Step 3">
          <h1>Go!</h1>
        </asp:WizardStep>
      </WizardSteps>
    </asp:Wizard>
  </ContentTemplate>
</asp:UpdatePanel>
```

В этой разметке следует отметить два важных момента. Во-первых, атрибуту UpdateMode элемента UpdatePanel присвоено значение "Conditional"; позже, при обнаружении нажатия кнопки броузера Назад или Вперед, нам потребуется обращение к программному коду на стороне сервера для обновления панели. Во-вторых, всякий раз, когда пользователь будет производить переход к следующему шагу мастера (Wizard) (свойство OnActiveStepChanged), будет вызываться метод с именем stepChanged(). Этот метод будет записывать номер текущего шага в элемент управления UpdateHistory.

Для этого можно воспользоваться методом AddEntry(). Он ожидает получить строковый аргумент – остальные типы данных должны быть предварительно преобразованы в строку. Ниже приводится сам код:

```
protected void stepChanged(object sender, EventArgs e)
{
  UpdateHistory1.AddEntry(((Wizard)sender).ActiveStepIndex.ToString());
}
```

Метод historyNavigate() вызывается при загрузке страницы или ее содержимого. Элемент управления UpdateHistory автоматически предо-

ставит соответствующую запись из журнала посещений. После этого все, что останется сделать, – это прочитать запись и на ее основе определить шаг мастера. Не забывайте перепроверять информацию, на случай если злоумышленник попытается добавить в строку URL неверные данные:

```
protected void historyNavigate(object sender, HistoryEventArgs e)
{
  int step = 0;
  if (e.EntryName != null)
  {
    int.TryParse(e.EntryName.ToString(), out step);
  }
  if (step >= 0 && step < Wizard1.WizardSteps.Count)
  {
    Wizard1.ActiveStepIndex = step;
  }
  UpdatePanel1.Update();
}
```

Пример 20.1 содержит полный код этого примера. Данный пример загружает сборку UpdateControls, добавляет в страницу элемент управления UpdateHistory и содержит программный код, необходимый для обеспечения работоспособности механизма закладок и кнопок Назад/Вперед.

Пример 20.1. Исправление недостатков в поддержке механизма закладок и кнопок Назад/Вперед

```
Bookmarks.aspx

<%@ Page Language="C#" %>

<%@ Register Assembly="nStuff.UpdateControls" TagPrefix="nk"
Namespace="nStuff.UpdateControls" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

  protected void stepChanged(object sender, EventArgs e)
  {
    UpdateHistory1.AddEntry(((Wizard)sender).ActiveStepIndex.ToString());
  }

  protected void historyNavigate(object sender, HistoryEventArgs e)
  {
    int step = 0;
    if (e.EntryName != null)
    {
      int.TryParse(e.EntryName.ToString(), out step);
    }
    if (step >= 0 && step < Wizard1.WizardSteps.Count)
```

```
    {
      Wizard1.ActiveStepIndex = step;
    }
    UpdatePanel1.Update();
  }

</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <nk:UpdateHistory ID="UpdateHistory1" runat="server"
                      OnNavigate="historyNavigate" />
    <asp:UpdatePanel ID="UpdatePanel1" runat="server"
                     UpdateMode="Conditional">
      <ContentTemplate>
        <asp:Wizard ID="Wizard1" runat="server"
                    OnActiveStepChanged="stepChanged">
          <WizardSteps>
            <asp:WizardStep runat="server" Title="Step 1">
              <h1>Ready ...</h1>
            </asp:WizardStep>
            <asp:WizardStep runat="server" Title="Step 2">
              <h1>Set ...</h1>
            </asp:WizardStep>
            <asp:WizardStep runat="server" Title="Step 3">
              <h1>Go!</h1>
            </asp:WizardStep>
          </WizardSteps>
        </asp:Wizard>
      </ContentTemplate>
    </asp:UpdatePanel>
  </form>
</body>
</html>
```

На рис. 20.1 показано, как выглядит страница сразу после загрузки. Обратите внимание, как изменилась строка URL после перехода ко второму шагу мастера (рис. 20.2)[1]. Щелчок на кнопке броузера Назад должен выполнить возврат к первому шагу мастера. Если скопировать строку URL из рис. 20.2 в другой броузер, этот броузер тоже должен отобразить страницу на втором шаге мастера (рис. 20.3), что доказывает наличие поддержки механизма закладок.

[1] Отметим также, что автор вновь использует порт 1234 отладочного сервера, как это было в первых главах. – *Примеч. науч. ред.*

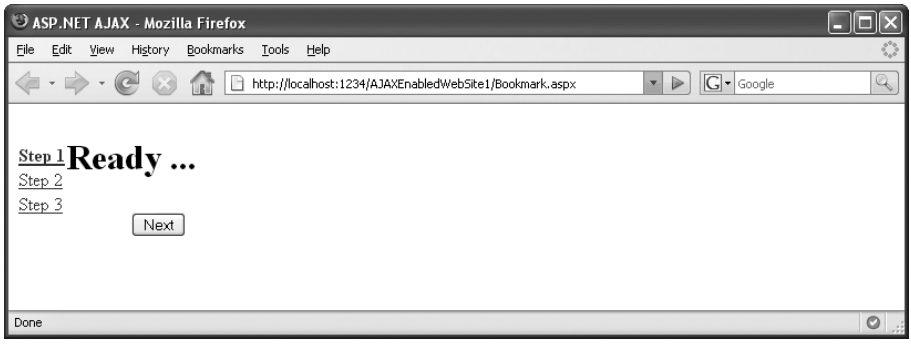


Рис. 20.1. *Начальное состояние страницы*

Рис. 20.2. *Был выполнен переход ко второму шагу мастера*

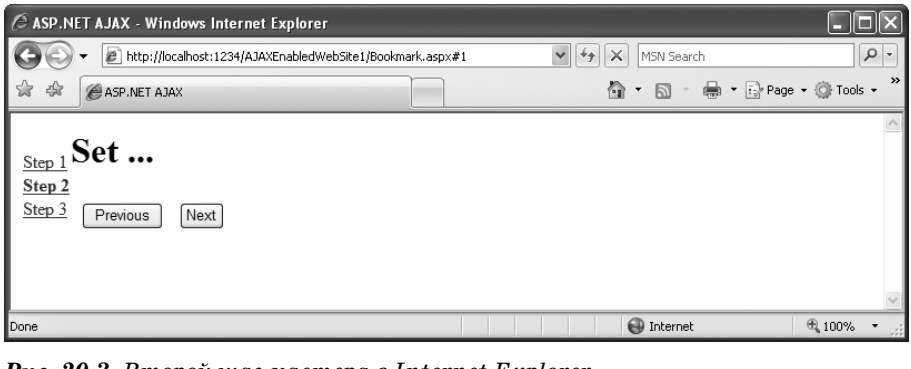


Рис. 20.3. *Второй шаг мастера в Internet Explorer*

Исправление недостатков в поддержке механизма закладок и кнопок Назад/Вперед с помощью элементов управления из пакета ASP.NET AJAX Futures

Начиная с мая 2007 года в состав пакета ASP.NET AJAX Futures CTP входит элемент управления, позволяющий устранить недостатки в поддержке механизма закладок и кнопок Назад/Вперед. Этот элемент управления очень похож на тот, что рассматривался в этой главе ранее, но его прикладной интерфейс несколько отличается. Поскольку содержимое пакета Futures постоянно изменяется, в этой главе рассматриваются оба элемента управления.

Между элементом управления `UpdateHistory` и соответствующим ему элементом управления из пакета Futures существует несколько отличий:

- Имя элемента управления изменилось на `History`.
- Запись информации о текущем состоянии страницы в объект History производится с помощью метода `AddHistoryPoint()`. Этот метод принимает два аргумента – ключ и значение, так как каждая запись в журнале – это объект `Dictionary`.
- Получить доступ к информации о текущем состоянии можно с помощью свойства `State` объекта `HistoryEventArgs`. Как только что упоминалось – это объект `Dictionary`.

В примере 20.2 приводится измененный код страницы, которая на этот раз вместо элемента управления Котхари использует элемент управления из пакета Futures. Обратите внимание: здесь в качестве ключа к информации о состоянии используется значение `"myHistory"`. Данное название было выбрано совершенно произвольно – требуется лишь использовать одно и то же значение при чтении и записи данных. На рис. 20.4 показан результат работы нового примера – теперь строка URL содержит больше информации, поскольку вместо статической строки используется объект `Dictionary`.

Пример 20.2. Исправление недостатков в поддержке механизма закладок и кнопок Назад/Вперед средствами пакета Futures

```
BookmarksUpdatePanel.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

  protected void stepChanged(object sender, EventArgs e)
  {
    History1.AddHistoryPoint("myHistory",
      ((Wizard)sender).ActiveStepIndex.ToString());
```

```
  }

  protected void historyNavigate(object sender, HistoryEventArgs e)
  {
    int step = 0;
    if (e.State.ContainsKey("myHistory"))
    {
      int.TryParse(e.State["myHistory"].ToString(), out step);
    }
    if (step >= 0 && step < Wizard1.WizardSteps.Count)
    {
      Wizard1.ActiveStepIndex = step;
    }
    UpdatePanel1.Update();
  }

</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>ASP.NET AJAX</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <asp:History ID="History1" runat="server" OnNavigate="historyNavigate" />
    <asp:UpdatePanel ID="UpdatePanel1" runat="server"
                     UpdateMode="Conditional">
      <ContentTemplate>
        <asp:Wizard ID="Wizard1" runat="server"
                    OnActiveStepChanged="stepChanged">
          <WizardSteps>
            <asp:WizardStep runat="server" Title="Step 1">
              <h1>Ready ...</h1>
            </asp:WizardStep>
```

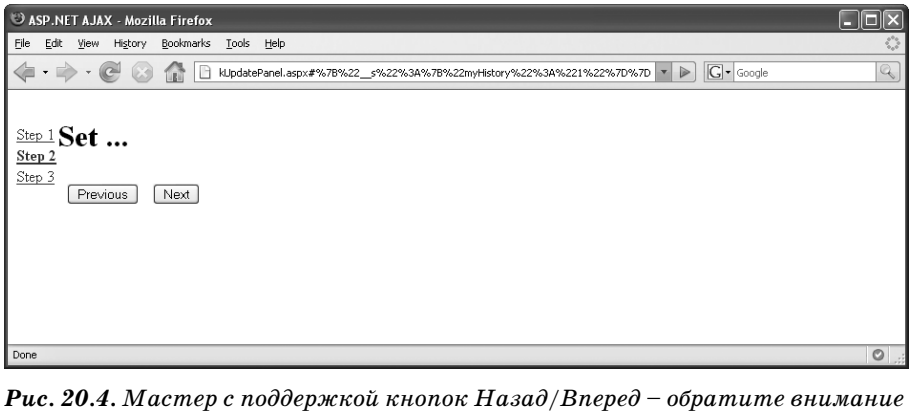


Рис. 20.4. *Мастер с поддержкой кнопок Назад/Вперед – обратите внимание на более длинную строку URL*

```
            <asp:WizardStep runat="server" Title="Step 2">
              <h1>Set ...</h1>
            </asp:WizardStep>
            <asp:WizardStep runat="server" Title="Step 3">
              <h1>Go!</h1>
            </asp:WizardStep>
          </WizardSteps>
        </asp:Wizard>
      </ContentTemplate>
    </asp:UpdatePanel>
  </form>
</body>
</html>
```

Одно из преимуществ элемента управления из пакета Futures состоит в том, что он обладает прикладным интерфейсом на уровне JavaScript. Благодаря этому вам даже не потребуется добавлять элемент управления `UpdatePanel`. Если управление информацией о состоянии производится с помощью JavaScript, можно будет использовать JavaScript-интерфейс элемента управления `History` для записи информации о состоянии (которая, в свою очередь, будет добавлена в хеш URL). Кроме того, данный элемент управления автоматически вызывает функцию `pageNavigate()`. С ее помощью можно привести состояние страницы в соответствие с данными, полученными из элемента управления `History`.

Для следующего примера мы попробуем самостоятельно реализовать клиентскую версию (пусть и ограниченную) элемента управления ASP.NET 2.0 `Wizard`. Три шага мастера моделируются тремя строками HTML-таблицы, которые изначально невидимы:

```
<table cellspacing="0" cellpadding="0" border="0"
       style="height: 100%; width: 100%; border-collapse: collapse;">
  <tr id="Wizard1_Step1" style="height: 100%; display: none;">
    <td>
      <h1>
        Ready ...</h1>
    </td>
  </tr>
  <tr id="Wizard1_Step2" style="height: 100%; display: none;">
    <td>
      <h1>
        Set ...</h1>
    </td>
  </tr>
  <tr id="Wizard1_Step3" style="height: 100%; display: none;">
    <td>
      <h1>
        Go!</h1>
    </td>
  </tr>
</table>
```

Переход между шагами мастера будет производиться с помощью трех ссылок в левой части страницы. На этот раз ссылки будут иметь реализацию на JavaScript:

```
<table id="Wizard1_SideBarContainer_SideBarList" cellspacing="0"
       border="0" style="border-collapse: collapse;">
  <tr>
    <td id="Wizard1_SideBar1">
      <a id="Wizard1_SideBarContainer_SideBarList_ctl00_SideBarButton"
         href="javascript:gotoStepClick(1)">
        Step 1</a></td>
  </tr>
  <tr>
    <td id="Wizard1_SideBar2">
      <a id="Wizard1_SideBarContainer_SideBarList_ctl01_SideBarButton"
         href="javascript:gotoStepClick(2)">
        Step 2</a></td>
  </tr>
  <tr>
    <td id="Wizard1_SideBar3">
      <a id="Wizard1_SideBarContainer_SideBarList_ctl02_SideBarButton"
         href="javascript:gotoStepClick(3)">
        Step 3</a></td>
  </tr>
</table>
```

Функция `gotoStepClick()` сначала добавляет новую запись в журнал посещений. Метод на JavaScript, соответствующий методу `addHistoryPoint()` серверного элемента управления `History`, выглядит следующим образом:

```
function gotoStepClick(nr) {
  Sys.Application.get_history().addHistoryPoint({myHistory: nr});
```

Затем `gotoStepClick()` вызывает еще одну вспомогательную функцию `gotoStep()`:

```
  gotoStep(nr);
}
```

Функция `gotoStep()` отвечает за корректное отображение текущего шага мастера (и сокрытие остальных). Кроме того, текст ссылки, соответствующей текущему шагу, выводится жирным шрифтом:

```
function gotoStep(nr) {
  if (nr >= 1 && nr <= 3) {
    $get("Wizard1_Step1").style.display = "none";
    $get("Wizard1_Step2").style.display = "none";
    $get("Wizard1_Step3").style.display = "none";
    $get("Wizard1_SideBar1").style.fontWeight = "normal";
    $get("Wizard1_SideBar2").style.fontWeight = "normal";
    $get("Wizard1_SideBar3").style.fontWeight = "normal";
    $get("Wizard1_Step" + nr).style.display = "";
```

```
      $get("Wizard1_SideBar" + nr).style.fontWeight = "bold";
    }
  }
```

Заключительный фрагмент этого примера выполняет восстановление состояния страницы, когда пользователь щелкает на кнопке броузера Вперед или Назад. Как уже упоминалось ранее, сделать это можно с помощью JavaScript-функции `pageNavigate()`. Второй аргумент, который передается функции автоматически, содержит все переменные с информацией о состоянии. Значения этих переменных извлекаются с помощью метода `get_state()`. Поскольку ранее, при добавлении записи в журнал посещений, использовался ключ `myHistory`, для извлечения конкретной информации о состоянии, сохраненной ранее, необходимо использовать конструкцию `get_state().myHistory`:

```
function pageNavigate(sender, e) {
  var step = 1;
  if (e.get_state().myHistory != null) {
    step = e.get_state().myHistory;
  }
  gotoStep(step);
}
```

Вот и все! Полный код страницы этого примера приводится в примере 20.3, а на рис. 20.5 показано, как выглядит наш мастер в окне броузера.

Пример 20.3. Исправление недостатков в поддержке механизма закладок и кнопок Назад/Вперед средствами JavaScript

```
BookmarksJavaScript.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
```

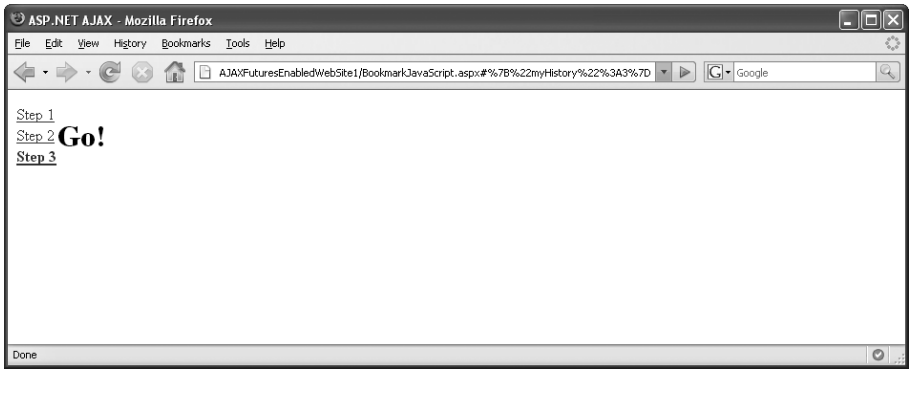


Рис. 20.5. *Имитация элемента управления Wizard*

```
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
  <script type="text/javascript">
    function pageLoad() {
      gotoStep(1);
    }

    function gotoStepClick(nr) {
      Sys.Application.get_history().addHistoryPoint({myHistory: nr});
      gotoStep(nr);
    }

    function gotoStep(nr) {
      if (nr >= 1 && nr <= 3) {
        $get("Wizard1_Step1").style.display = "none";
        $get("Wizard1_Step2").style.display = "none";
        $get("Wizard1_Step3").style.display = "none";
        $get("Wizard1_SideBar1").style.fontWeight = "normal";
        $get("Wizard1_SideBar2").style.fontWeight = "normal";
        $get("Wizard1_SideBar3").style.fontWeight = "normal";
        $get("Wizard1_Step" + nr).style.display = "";
        $get("Wizard1_SideBar" + nr).style.fontWeight = "bold";
      }
    }

    function pageNavigate(sender, e) {
      var step = 1;
      if (e.get_state().myHistory != null) {
        step = e.get_state().myHistory;
      }
      gotoStep(step);
    }

  </script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <asp:History ID="History1" runat="server" />
    <table cellspacing="0" cellpadding="0" border="0" id="Wizard1"
           style="bordercollapse: collapse;">
      <tr>
        <td style="height:100%;"><table
id="Wizard1_SideBarContainer_SideBarList" cellspacing="0" border="0" style="
bordercollapse:collapse;">
          <tr>
            <td id="Wizard1_SideBar1"><a
id="Wizard1_SideBarContainer_SideBarList_ctl00_SideBarButton"
href="javascript:gotoStepClick(1)">Step 1</a></td>

          </tr><tr>
            <td id="Wizard1_SideBar2"><a
id="Wizard1_SideBarContainer_SideBarList_ctl01_SideBarButton"
```

```
href="javascript:gotoStepClick(2)">Step 2</a></td>
          </tr><tr>
            <td id="Wizard1_SideBar3"><a
id="Wizard1_SideBarContainer_SideBarList_ctl02_SideBarButton"
href="javascript:gotoStepClick(3)">Step 3</a></td>
          </tr>
        </table></td><td style="height:100%;"><table cellspacing="0"
cellpadding="0" border="0" style="height:100%;width:100%;border-
collapse: collapse;">
          <tr id="Wizard1_Step1" style="height:100%;display:none;">
            <td>
              <h1>Ready ...</h1>
            </td>
          </tr>
          <tr id="Wizard1_Step2" style="height:100%;display:none;">
            <td>
              <h1>Set ...</h1>
            </td>
```

Создание постоянных ссылок

Под *постоянными ссылками* понимаются ссылки, которые хранят в себе информацию о состоянии веб-страницы. Вероятно, вы заметили в примерах приложений из этой главы, что адреса URL изменяются всякий раз, когда изменяется состояние страницы (и в примерах имеется программный код, который обрабатывает эти изменения). Таким образом, всякий раз, когда устанавливается закладка на текущую страницу, строка URL будет содержать всю информацию, необходимую для восстановления состояния страницы.

Кроме того, элемент управления `History` предоставляет вспомогательный метод, позволяющий извлекать информацию из строки URL. Разумеется, можно было бы использовать свойство JavaScript `location.href`. А также прибегнуть к помощи следующих методов:

`getStateString()`

Серверный метод, позволяющий извлечь информацию о состоянии из URL

`Sys.Application.get_history().get_stateString()`

Клиентский метод, позволяющий извлечь информацию о состоянии из URL

Некоторые веб-страницы включают в себя перманентную гиперссылку (*permalink hyperlink*). Всякий раз, когда изменяется состояние страницы, соответствующим образом изменяется и URL этой гиперссылки.

```
            </tr>
            <tr id="Wizard1_Step3" style="height:100%;display:none;">
              <td>
                <h1>Go!</h1>
              </td>
            </tr>
          </table></td>
        </tr>
      </table>
    </form>
  </body>
  </html>
```

Подведение итогов

В этой главе вы познакомились с двумя известными проблемами, присущими многим Ajax-приложениям: недостатки в поддержке механизма закладок и неработающие кнопки броузера Назад и Вперед. В этой главе также были даны варианты решения этих проблем. Это может потребовать дополнительных усилий, но удобство использования страниц с поддержкой Ajax при этом существенно повысится.

Для дополнительного чтения

http://quickstarts.asp.net/futures/ajax/doc/history.html

Документация с описанием элемента управления `History`

Wenz, Christian. Ajax, Software & Support (Entwickler Press)

Знакомит с программным кодом, решающим проблему недостатков в поддержке механизма закладок и кнопок Назад/Вперед

21

Веб-части

Применение технологий Ajax позволяет создавать веб-приложения, которые по своему поведению напоминают обычные настольные приложения. И чем больше они становятся похожими друг на друга, тем больше появляется разработчиков, стремящихся создавать и многократно использовать компоненты, обладающие широкими функциональными возможностями.

Платформа ASP.NET AJAX предлагает несколько способов многократного использования компонентов с целью расширить функциональные возможности веб-клиентов. Основным примером могут служить расширители элементов управления из пакета ASP.NET AJAX Control Toolkit. В этой главе будет рассматриваться еще одна разновидность особенностей ASP.NET (появившаяся в ASP.NET 2.0) – веб-части (web parts), которые используют дополнительные возможности, предоставляемые пакетом ASP.NET AJAX Futures.

ASP.NET AJAX и ASP.NET Web Parts

В этом разделе будет продемонстрировано, как можно использовать платформу ASP.NET AJAX в комплексе с пакетом ASP.NET Web Parts, чтобы дать пользователю еще больше возможностей по управлению содержимым и его размещением на странице ASP.NET AJAX. ASP.NET Web Parts – это множество элементов управления, позволяющих пользователю добавлять, удалять и изменять элементы страницы в процессе исполнения. Веб-части предоставляют возможность создавать в ASP.NET такие страницы, как индивидуализированная домашняя страница Google (*http://www.google.com/ig*).

Веб-части позволяют в клиентских сценариях поддерживать механизм «перетащить-и-отпустить», возможность сворачивать и разворачивать элементы интерфейса и обеспечение других подобных функций. Одна-

ко веб-части, распространяемые в составе ASP.NET 2.0, имеют ограниченную область применения, поскольку их функциональность доступна только в Internet Explorer. Таким образом, компоненты ASP.NET Web Parts в основном используются в локальных сетях, в которых можно гарантировать работу с Internet Explorer.

Разумеется, Internet Explorer – не единственный броузер, и многие используют Firefox или другие броузеры. Поэтому, несмотря на всю привлекательность, веб-части плохо подходят для общедоступных веб-сайтов, обладающих широким кругом пользователей.

Однако это ограничение может быть компенсировано с помощью платформы ASP.NET AJAX. Благодаря пакету ASP.NET AJAX Futures теперь возможно использовать веб-части, одинаково функциональные в Internet Explorer и в Firefox. Если вы занимаетесь разработкой общедоступного сайта и вам необходимо обеспечить поддержку разных типов броузеров, то использование компонентов ASP.NET AJAX Web Parts будет лучшим выбором.

В этом разделе вы узнаете, как можно реализовать веб-части с использованием ASP.NET AJAX. Я не буду здесь углубляться в описание принципов, лежащих в основе веб-частей (это слишком обширная тема), поэтому, если вам необходимы дополнительные сведения об основах веб-частей, обращайтесь к соответствующей документации. Лучше всего начинать изучение с домашней страницы проекта ASP.NET AJAX Web Parts (*http://msdn2.microsoft.com/en-US/library/e0s9t4ck.aspx*).

В примере, который следует ниже, веб-части ASP.NET AJAX будут использоваться для упаковки элементов управления `Calendar` и `Wizard`, затем к обоим будет добавлена поддержка механизма «перетащить-и-отпустить», чтобы пользователь имел возможность расположить эти компоненты на странице по своему усмотрению. Пример предусматривает сохранение изменений, произведенных пользователем. Таким образом, если в броузере будет включена поддержка cookies, то при повторном посещении страницы эти два элемента управления окажутся в том же положении, в каком они были оставлены при предыдущем посещении.

Существует два способа работы с веб-частями ASP.NET AJAX. Первый заключается в отображении тегов ASP.NET Web Parts в эквивалентные им теги ASP.NET AJAX – в этом случае для `<asp:WebPartZone>` используется версия веб-частей платформы ASP.NET AJAX. Такой подход можно применить, когда требуется адаптировать существующие веб-страницы, использующие веб-части, под использование элементов управления ASP.NET AJAX, не перестраивая сайт с самого начала.

Для отображения тегов следует добавить в файл *Web.config* элемент `<tagMapping>`. Этот элемент все ссылки на теги определенного типа отображает в ссылки на теги другого типа.

Следующий фрагмент файла *Web.config* демонстрирует, как реализовать отображение двух тегов веб-частей ASP.NET (которые определены

в пространстве имен System.Web.UI.WebControls.WebParts.WebPartManager) в эквивалентные им теги веб-частей ASP.NET AJAX (которые определены в пространстве имен Microsoft.Web.Preview.UI.Controls.WebParts.WebPartManager).

```
<pages>
<!-- Прочие настройки страницы -->
  <tagMapping>
    <add tagType="System.Web.UI.WebControls.WebParts.WebPartManager"
mappedTagType="Microsoft.Web.Preview.UI.Controls.WebParts.WebPartManager"/>
    <add tagType="System.Web.UI.WebControls.WebParts.WebPartZone"
mappedTagType="Microsoft.Web.Preview.UI.Controls.WebParts.WebPartZone"/>
  </tagMapping>
</pages>
```

Данная разметка отобразит типы ASP.NET WebPartManager и WebPartZone в эквивалентные им типы ASP.NET AJAX. (Вообще тип, указанный в атрибуте tagType отображается в тип, указанный в атрибуте mappedTagType.) Такой способ отобразит *все* теги веб-частей в приложении.

Второй способ применения веб-частей версии ASP.NET AJAX заключается в непосредственном использовании элементов управления веб-частей. Это позволяет использовать веб-части в отдельных страницах, не затрагивая приложение в целом.

Для этого достаточно зарегистрировать пространство имен Microsoft.Web.Preview.UI.Controls.WebParts. Вставьте следующий фрагмент в элемент <system.web> в файле *Web.config*:

```
<pages>
<!-- Прочие настройки страницы -->
  <controls>
    <!-- Прочие пространства имен элементов управления -->
    <add
      namespace="Microsoft.Web.Preview.UI.Controls.WebParts"
      assembly="Microsoft.Web.Preview"
      tagPrefix="ajax" />
  </controls>
</pages>
```

Теперь вы можете обращаться к элементам ASP.NET AJAX Web Parts, используя префикс ajax, например: <ajax:WebPartManager>.

В этой главе будет использоваться первый способ (основанный на отображении тегов), поэтому нам не потребуется вводить в употребление дополнительный префикс. Не забудьте добавить элемент <tagMapping> в файл *Web.config*, прежде чем приступить к опробованию примера из этой главы. Помните также, что он отобразит *все* элементы управления Web Parts на веб-сайте.

Теперь можно приступать к созданию страницы ASP.NET с поддержкой веб-частей ASP.NET AJAX. Как обычно, нам потребуется элемент управления ScriptManager. Кроме того, чтобы активировать поддержку веб-частей, необходимо добавить элемент управления WebPartManager:

```
<asp:WebPartManager ID="WebPartManager1" runat="server" />
```

Зоны веб-частей – это области на странице, где могут появляться веб-части – в сущности, они являются контейнерами для веб-частей. Вы можете перемещать веб-части между зонами, а также скрывать или показывать зоны, чтобы скрывать или показывать веб-части, расположенные в них. Создаются зоны с помощью элемента управления WebPartZone и его дочернего элемента <ZoneTemplate>, в котором размещается содержимое веб-части. Ниже приводится объявление двух зон веб-частей, которые содержат элементы управления – Calendar и Wizard:

```
<asp:WebPartZone ID="WebPartZone1" HeaderText="Zone 1" runat="server">
  <ZoneTemplate>
    <asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
  </ZoneTemplate>
</asp:WebPartZone>
  <asp:WebPartZone ID="WebPartZone2" HeaderText="Zone 2" runat="server">
    <ZoneTemplate>
      <asp:Wizard ID="Wizard1" runat="server">
        <WizardSteps>
          <asp:WizardStep ID="Step1" runat="server" Title="Step 1" />
          <asp:WizardStep ID="Step2" runat="server" Title="Step 2" />
        </WizardSteps>
      </asp:Wizard>
    </ZoneTemplate>
</asp:WebPartZone>
```

Чтобы получить возможность перетаскивать веб-части мышью, необходимо в свойство DisplayMode элемента управления WebPartManager записать значение DesignDisplayMode. Режим отображения нельзя установить декларативно, но нам поможет следующий серверный программный код на C#:

```
void Page_Init ()
{
  WebPartManager1.DisplayMode =
Microsoft.Web.Preview.UI.Controls.WebParts.WebPartManager.DesignDisplayMode;
}
```

Поскольку теперь у нас определены два пространства имен WebParts (одно в ASP.NET 2.0, а второе в ASP.NET AJAX), появляется неоднозначность ссылки на WebPartManager.DesignDisplayMode. Таким образом, при любой ссылке на режим отображения необходимо указывать полное имя.

В примере 21.1 содержится полный код этого примера. На рис. 21.1 можно увидеть результаты работы страницы в броузере Firefox – возможность перемещения веб-частей с помощью мыши теперь поддерживается. Самый первый запуск примера будет происходить с заметной задержкой, так как в это время ASP.NET выполняет настройку базы данных, где будет сохраняться информация о веб-частях. При пере-

мещении веб-части или при закрытии зоны информация о состоянии веб-частей сохраняется между сеансами броузера.

Не забудьте, что вам нужно настроить отображение тегов в файле *Web.config*.

Пример 21.1. Веб-части в ASP.NET AJAX

```
WebParts.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  void Page_Init()
  {
    WebPartManager1.DisplayMode =
Microsoft.Web.Preview.UI.Controls.WebParts.WebPartManager.DesignDisplayMode;
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>
</head>
```

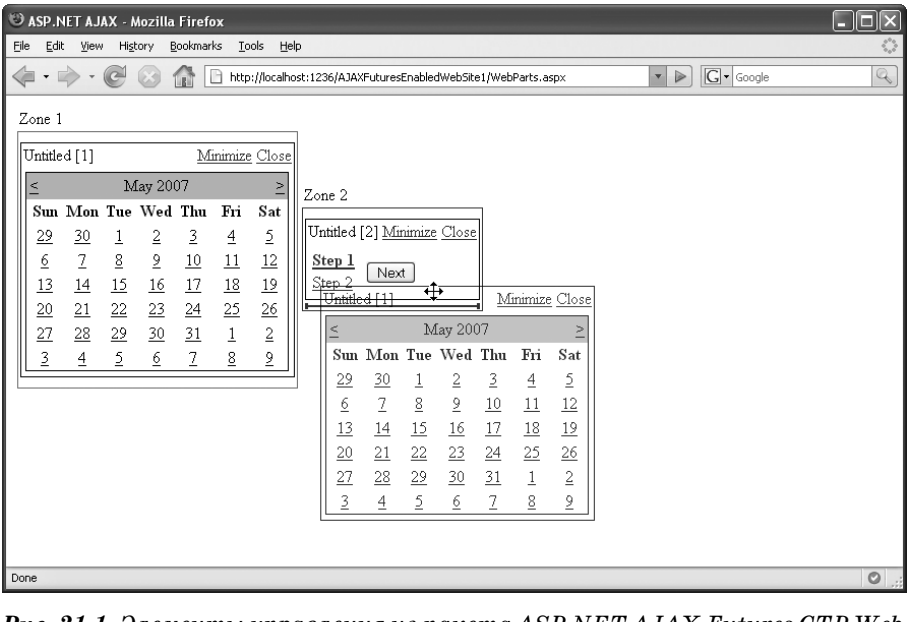


Рис. 21.1. *Элементы управления из пакета ASP.NET AJAX Futures CTP Web Parts поддерживают возможность перемещения мышью в броузерах Mozilla*

```
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div>
      <asp:WebPartManager ID="WebPartManager1" runat="server" />
      <table>
        <tr>
          <td>
            <asp:WebPartZone ID="WebPartZone1" HeaderText="Zone 1"
                             runat="server">
              <ZoneTemplate>
                <asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
              </ZoneTemplate>
            </asp:WebPartZone>
          </td>
          <td>
            <asp:WebPartZone ID="WebPartZone2" HeaderText="Zone 2"
                             runat="server">
              <ZoneTemplate>
                <asp:Wizard ID="Wizard1" runat="server">
                  <WizardSteps>
                    <asp:WizardStep ID="Step1" runat="server"
                                    Title="Step 1" />
                    <asp:WizardStep ID="Step2" runat="server"
                                    Title="Step 2" />
                  </WizardSteps>
                </asp:Wizard>
              </ZoneTemplate>
            </asp:WebPartZone>
          </td>
        </tr>
      </table>
    </div>
  </form>
</body>
</html>
```

Перемещение элементов Web Parts всегда вызывает обратную передачу данных с сервера, что приводит к обновлению окна броузера. Если поместить на страницу элемент управления ASP.NET AJAX `UpdatePanel` и включить в него `WebPartManager` и два элемента управления `WebPartZone`, тогда перемещение элементов управления Web Parts больше не будет вызывать полного обновления страницы. Однако в этом случае вы сможете всего один раз переместить элемент, после этого данная возможность перестает работать. (Если обновить страницу в броузере, вы сможете переместить элемент еще раз – одно перемещение на каждое обновление страницы.) Ожидается, что в следующих версиях ASP.NET AJAX Futures этот недостаток будет исправлен, что позволит вам вставлять элемент управления `UpdatePanel` в каждую зону `WebPartZone`.

В блоге Майка Хардера (Mike Harder) (ссылка приводится в разделе «Для дополнительного чтения» в конце этой главы) описываются три ситуации, когда веб-части работают или не работают:

- Веб-части работают в броузерах разных типов
- В настоящее время веб-части не работают внутри `UpdatePanel`, но, возможно, будут работать с выходом следующей версии Visual Studio (Visual Studio 2008)
- Внутри веб-частей элементы `UpdatePanel` работают

Подведение итогов

В этой главе был исследован один из уникальных подходов к использованию (и многократному использованию) компонентов ASP.NET AJAX. В настоящее время веб-части используются довольно редко, но начало уже положено. В отличие от своих аналогов в ASP.NET 2.0, веб-части из ASP.NET AJAX Futures CTP работают в броузерах любого типа.

Для дополнительного чтения

http://msdn2.microsoft.com/en-US/library/e0s9t4ck.aspx

Раздел в библиотеке MSDN, посвященный ASP.NET 2.0 Web Parts

http://blogs.msdn.com/mharder/archive/2007/01/23/webparts-and-asp-net-ajax-1-0.aspx

Статья в блоге Майка Хардера (Mike Harder), посвященная ASP.NET AJAX и веб-частям

V

Microsoft AJAX Library

Глава 22. ASP.NET AJAX в комбинации с другими серверными технологиями

22

ASP.NET AJAX в комбинации с другими серверными технологиями

Как уже обсуждалось в начале книги, в главе 1, платформа ASP.NET AJAX включает в себя как клиентские, так и серверные компоненты. Серверные компоненты ASP.NET AJAX очень тесно связаны с элементами управления ASP.NET 2.0, а клиентские компоненты распространяются в виде библиотек JavaScript. Хотя клиентские библиотеки встраиваются в страницы в виде тега `<script>`, ссылающегося на *WebResource.axd*, сами библиотеки в действительности являются самостоятельными файлами *.js*, которые поставляются в виде встроенных ресурсов библиотеки ASP.NET AJAX.

В первой главе также упоминалась библиотека Microsoft AJAX Library. Она представляет собой подмножество библиотеки ASP.NET AJAX. Но поскольку данная библиотека не зависит от ASP.NET, она также может использоваться совместно с любым другим языком сценариев, исполняемых на стороне сервера.

С помощью этой библиотеки вы сможете пользоваться некоторыми преимуществами ASP.NET AJAX и при использовании других серверных технологий (не имеющих отношения к ASP.NET). Ваши возможности не ограничиваются клиентскими особенностями ASP.NET AJAX; вы можете также использовать дополнительные возможности сервера. Однако, чтобы реализовать функциональность ASP.NET AJAX на платформах, отличных от ASP.NET 2.0 и IIS, некоторые из особенностей ASP.NET AJAX и функциональность серверных элементов управления (таких как `ScriptManager`) необходимо будет имитировать с помощью других технологий.

В этой главе демонстрируется, как использовать поддержку веб-служб ASP.NET AJAX с помощью PHP. Представленный здесь программный код проделал долгий путь. Сначала Шанку Ниеги (Shanku Niyogi) напи-

сал демонстрационную версию и опубликовал ее в своем блоге (ссылка приводится в разделе «Для дополнительного чтения» в конце этой главы). Мы обменялись несколькими электронными письмами, и спустя некоторое время он выложил обновленную версию. В начале 2007 года Стив Маркс (Steve Marx) (Microsoft, «Developer Evangelist» в области ASP.NET AJAX) создал проект на сайте Codeplex с целью продемонстрировать возможность использования PHP совместно с библиотекой Microsoft AJAX Library. Программный код этого проекта, по сути, представлял собой обновленную версию кода, написанного Ниеги.

В следующих разделах демонстрируются возможности библиотеки и говорится о том, что необходимо сделать, чтобы можно было использовать библиотеку вместе с PHP. Кроме того, мы познакомимся с фрагментами программного кода. Здесь вы узнаете, как написать на языке PHP серверный модуль, который имитирует работу веб-службы. Мы будем обращаться к этой псевдослужбе из обычной HTML-страницы (не из страницы *.aspx*). Мы напишем программный код на JavaScript, который будет обращаться к модулю PHP с использованием функциональных возможностей ASP.NET AJAX для вызова веб-служб.

Использование ASP.NET AJAX вместе с PHP

Чтобы использовать платформу ASP.NET AJAX вместе с PHP, мы создадим HTML-страницу. Эта демонстрационная страница содержит текстовое поле и будет отправлять текст из этого поля веб-службе. Чтобы добавить функциональные возможности ASP.NET AJAX вызова веб-служб, необходимо загрузить клиентские библиотеки ASP.NET AJAX в HTML-страницу.

Подготовьте библиотеку ASP.NET AJAX, создав на веб-сайте новый каталог с именем *AjaxLibrary*. (Например, если веб-сайт размещается в каталоге *C:\Document and Settings\JaneDoe\AJAXEnabledWebSite1*, то необходимо создать каталог *C:\Document and Settings\JaneDoe\AJAXEnabledWebSite1\AjaxLibrary*.) Загрузите библиотеку Microsoft AJAX Library со страницы *http://ajax.asp.net/downloads/default.aspx?tabid=47* и скопируйте каталог *System.Web.Extensions* в только что созданный каталог *AjaxLibrary*.

Затем загрузите библиотеку Microsoft AJAX Library для PHP со страницы *http://codeplex.com/phpmsajax* (как и в случае с ASP.NET AJAX, достаточно будет загрузить пакет без исходных текстов («NoSource»)). Распакуйте библиотеку и скопируйте файлы *MSAjaxService.php* и *MSAjaxProxyGenerator.php* в корневой каталог веб-сайта.

В этой главе предполагается, что PHP на вашем компьютере уже установлен и настроен. Загрузить библиотеки PHP можно со страницы *http://www.php.net/downloads.php*. Вам потребуется PHP с версией не ниже 5.2.0 – более ранние версии не поддерживают некоторые из особенностей (такие как сериализация

в формат JSON), которые потребуются для реализации веб-службы. Внимательно следуйте инструкциям в процессе установки!

Чтобы иметь возможность запускать PHP-файлы под управлением IIS, вы должны использовать IIS (ASP.NET Development Server не способен запускать файлы *.php*). Инструкция по установке PHP содержит указания по установке PHP в IIS и в других веб-серверах, таких как Apache. Не устанавливайте файлы PHP в каталоги, имена которых содержат пробелы, например: *C:\Program Files*. Пробелы в представлении ISAPI могут привести к тому, что IIS будет неспособен отыскать соответствующие файлы *.dll*.

Если вы используете инсталлятор *.msi* PHP, он настроит IIS так, что все запросы к файлам *.php* будут передаваться обработчику PHP ISAPI. В противном случае вам нужно будет вручную описать реакцию веб-сервера на файлы с расширением *.php*.

В части примера на языке PHP необходимо создать веб-службу, которая может вызываться из ASP.NET AJAX. Программный код, который мы будем писать, в действительности не является «настоящей» веб-службой с интерфейсом SOAP, использующей описание WSDL, – но этот код совместим со способом, каким ASP.NET AJAX взаимодействует с веб-службами.

Создайте текстовый файл с именем *PHPHelloWorldService.php* в корневом каталоге веб-сайта. В этом файле сначала нужно загрузить программный код PHP, являющийся частью библиотеки Microsoft AJAX Library, который поможет создать «совместимую» веб-службу.

```
<?php
  require_once 'MSAjaxService.php';
```

Код в файле *MSAjaxService.php* создает расширяемый класс, пригодный для создания служб, которые могут вызываться из ASP.NET AJAX. Мы создадим простой класс PHP с очень простым методом «Hello World», аналогичным тому, что использовался в главе 1:

```
class PHPHelloWorldService extends MSAjaxService {
  function sayHello($name) {
    return "Hello $name, says the server!";
  }
}
```

В заключение сценарий PHP должен создать экземпляр класса `PHPHelloWorldService` и вызвать его метод `ProcessRequest()`, чтобы обработать входящий запрос от ASP.NET AJAX:

```
  $ps = new PHPHelloWorldService();
  $ps->ProcessRequest();
?>
```

В примере 22.1 содержится полный код нашей веб-службы на PHP.

Пример 22.1. Веб-служба на PHP, совместимая с ASP.NET AJAX

```
PHPHelloWorldService.php

<?php
  require_once 'MSAjaxService.php';

  class PHPHelloWorldService extends MSAjaxService {
    function sayHello($name) {
      return "Hello $name, says the server!";
    }
  }

  $ps = new PHPHelloWorldService();
  $ps->ProcessRequest();
?>
```

Главное назначение программного кода PHP состоит в том, чтобы снова создать объект-посредник JavaScript. На рис. 22.1 показан программный код на JavaScript, который создается при обращении по URL *PHPHelloWorldService.php/js*.

Теперь нам нужно написать JavaScript-код, который будет вызывать эту службу. В клиентском сценарии необходимо загрузить библиотеку Microsoft AJAX Library, которая была распакована в подкаталог *Ajax-Library*. Здесь мы используем относительный путь – проверьте, существует ли указанный путь на вашем веб-сайте, и измените его в случае необходимости:

```
<script type="text/javascript"
  src="AjaxLibrary/System.Web.Extensions/1.0.61025.0/MicrosoftAjax.js">
</script>
```

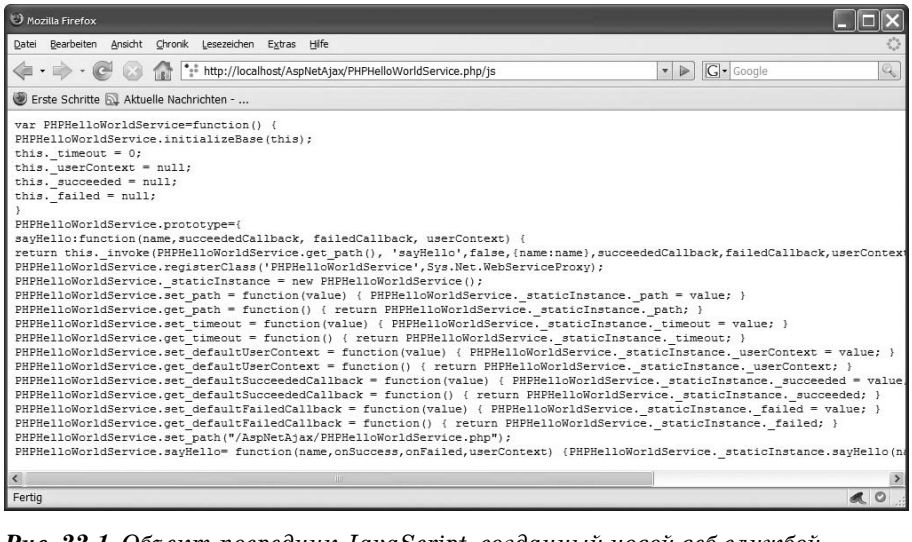


Рис. 22.1. *Объект-посредник JavaScript, созданный новой веб-службой*

Кроме того, необходимо загрузить объект-посредник JavaScript. Для этого нужно лишь добавить */js* в конец URL файла веб-службы:

```
<script type="text/javascript" src="PHPHelloWorldService.php/js"></script>
```

Последний элемент `<script>` генерирует программный код на JavaScript, который позволит вызвать метод веб-службы с использованием синтаксиса `<classname>.<methodname>`, то есть в нашем случае – метод `PHPHelloWorldService.sayHello()`. Пример 22.2 содержит полный код HTML-страницы, которая вызывает службу PHP с помощью библиотеки Microsoft AJAX Library. На рис. 22.2 показан результат работы этого примера.

Пример 22.2. Вызов веб-службы на PHP, совместимой с ASP.NET AJAX

```
PHPHelloWorldService.html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"

"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>ASP.NET AJAX</title>

  <script type="text/javascript"
    src="AjaxLibrary/System.Web.Extensions/1.0.61025.0/MicrosoftAjax.js">
  </script>
  <script type="text/javascript" src="PHPHelloWorldService.php/js"></script>

  <script language="Javascript" type="text/javascript">
  function callService(f) {
    PHPHelloWorldService.sayHello(
      f.elements["name"].value,
      callComplete,
      callError);
  }
```

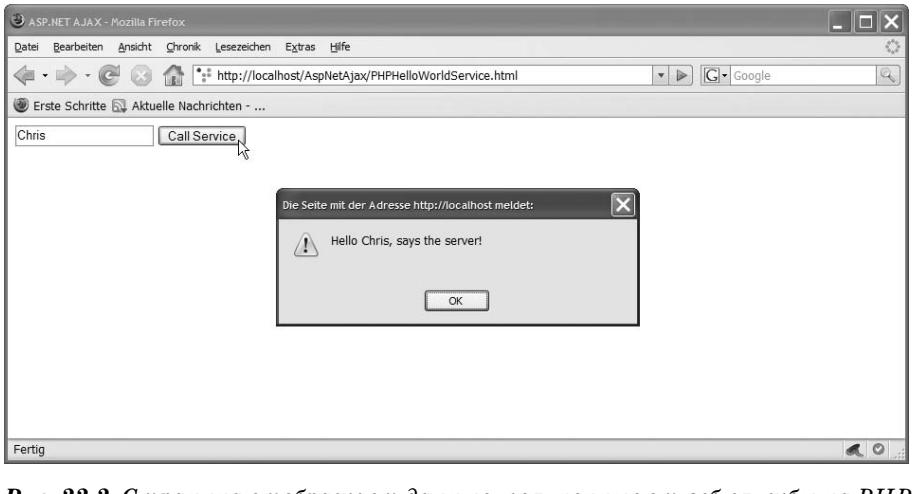


Рис. 22.2. *Страница отображает данные, полученные от веб-службы на PHP*

```
    function callComplete(result) {
      window.alert(result);
    }
    function callError(result) {
      window.alert("Error! " + result);
    }
    </script>
  </head>
  <body>
    <form id="form1">
      <div>
        <input type="text" id="name" name="name" />
        <input type="button" value="Call Service"
               onclick="callService(this.form);" />
      </div>
    </form>
  </body>
  </html>
```

Подведение итогов

В этой главе было показано, как работать с ASP.NET AJAX из PHP, используя библиотеку Microsoft AJAX Library из другой серверной технологии (и при желании – из другой операционной системы). Клиентские компоненты ASP.NET AJAX легко могут использоваться с другими языками программирования, так как сам JavaScript является платформонезависимым; работу серверных компонентов, с другой стороны, необходимо имитировать. Эта имитация, разумеется, не связана с определенным языком, так, скажем, пример в этой главе можно было бы написать на JSP, Perl, ColdFusion или даже на классическом ASP.

Проект PHP for Microsoft AJAX облегчает использование ASP.NET AJAX для PHP, поскольку большая часть хитроумного программного кода (такого как создание объекта-посредника на JavaScript) уже реализована.

Для дополнительного чтения

http://www.shankun.com/AtlasPhp.aspx

Оригинальная статья в блоге, демонстрирующая порядок использования платформы Atlas с PHP

http://www.shankun.com/Atlas_Php_2.aspx

Самая последняя версия программного кода Шанку Ниеги с некоторыми исправлениями

http://codeplex.com/phpmsajax

Сценарии PHP для Microsoft AJAX Library

VI

Приложения

Приложение A. Отладка приложений ASP.NET AJAX

Приложение B. Справочник по XMLHttpRequest

Приложение C. Справочник DOM

Приложение D. Справочник по ASP.NET AJAX

Приложение E. Справочник по ScriptManager, UpdatePanel, UpdateProgress и Timer

A

Отладка приложений ASP.NET AJAX

В совершенном мире каждая строка программного кода, написанная нашей рукой, была бы безупречна. Наверное, есть разработчики, которые никогда не совершают ошибок, но я не отношусь к их числу. Отладка приложений ASP.NET AJAX и приложений на JavaScript является делом более сложным, чем отладка «обычных» программ. Ajax-приложения исполняются и на стороне клиента, и на стороне сервера. Это влечет за собой определенные трудности в отладке: некоторые ошибки на стороне сервера не обнаруживаются на стороне клиента. Кроме того, нередко ошибки бывают обусловлены данными, курсирующими между клиентом и сервером, которые также необходимо контролировать.

В этом приложении будет представлено несколько полезных инструментальных средств, которые помогут повысить эффективность процесса отладки приложений ASP.NET AJAX, работающих в броузере (как и любой другой программный код JavaScript). Кроме того, будут рассмотрены возможности отладки, предоставляемые самой платформой ASP.NET AJAX.

Инструменты отладки

На вопрос: «Какой способ отладки JavaScript в процессе разработки самый лучший?», многие разработчики отвечают: «Отказаться от Internet Explorer». Несмотря на то, что самые последние версии всех основных броузеров действительно отличаются высоким качеством, степень подробности сообщений об ошибках у них разная. Internet Explorer дает самые неинформативные сообщения. Взгляните, какие сообщения об одной и той же ошибке генерируют различные броузеры. На рис. A.1 показано сообщение в Firefox, которое включает в себя номер строки, имя файла, а также (после щелчка на имени файла) сам программный код, вызвавший ошибку. Не так много, но с этим, по

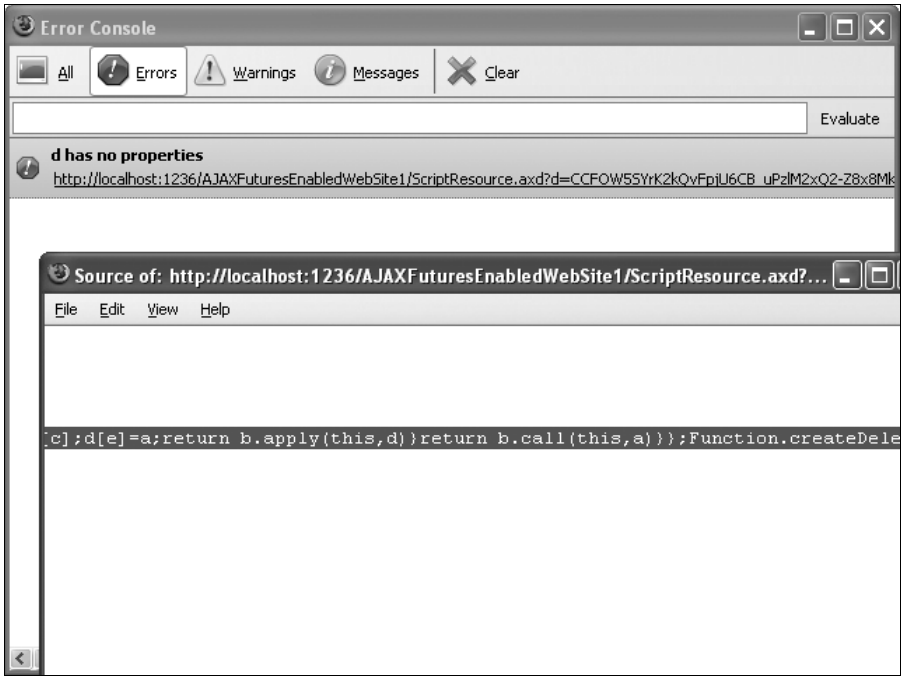


Рис. А.1. *Ошибка, порожденная сценарием на JavaScript в Firefox*

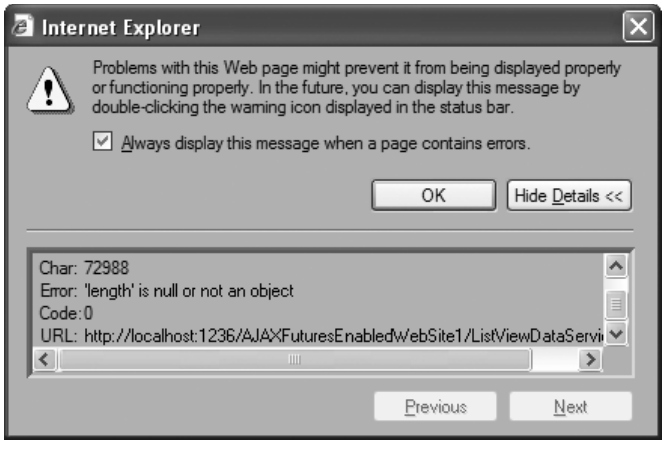


Рис. А.2. *Ошибка, порожденная сценарием на JavaScript в Internet Explorer*

крайней мере, уже можно работать. А на рис. А.2 представлено сообщение об ошибке в Internet Explorer – не настолько описательное, с другим номером строки и даже с другим именем файла.

Не поймите меня неправильно – я согласен с тем, что есть достаточно веские причины, почему Internet Explorer выводит малоинформатив-

ные сообщения. Но во время разработки содержимое таких сообщений может оказаться совершенно не соответствующим действительности и вводящим в заблуждение. Поэтому у разработчика должны быть дополнительные инструментальные средства. В действительности, и в Internet Explorer, *и* в Firefox полезно использовать дополнительные инструментальные средства отладки.

В этом приложении рассматриваются наиболее популярные в настоящее время броузеры: Internet Explorer и Firefox. Другие броузеры, такие как Opera и Safari, пока не обеспечивают широких возможностей подключения расширений, однако и для этих программ имеются очень неплохие решения.

Расширение Firebug для Firefox

Одним из самых лучших расширений броузера Firefox, предназначенных для веб-разработки, является Firebug. Он тесно интегрируется с броузером и обеспечивает богатые возможности для отладки. После установки нажмите в Firefox клавишу F12, чтобы запустить Firebug. После запуска расширение Firebug соберет и отобразит всю соответствующую информацию о текущей странице, начиная со структуры DOM (включая возможность доступа к элементам дерева для записи!) и заканчивая подробными сообщениями об ошибках в JavaScript. Firebug даже обладает возможностью выполнять переход к строке программы, вызвавшей ошибку, после щелчка на сообщении об ошибке. Кроме того, он позволяет получить подробный список HTTP-запросов, отправленных страницей (бесценная возможность для отладки обращений к `XMLHttpRequest`).

Чтобы установить расширение Firebug, перейдите на страницу *http://www.getfirebug.com/* и щелкните на кнопке Download. При этом вам может потребоваться разрешить установку расширений для Firefox из домена *getfirebug.com*. На рис. A.3 показано расширение Firebug в действии.

Web Development Helper (для Internet Explorer)

В течение долгого времени отладка JavaScript-приложений и особенно приложений с поддержкой Ajax в Internet Explorer была весьма сложным делом. Для Internet Explorer существовало несколько расширений, но ни одно из них не обладало всем необходимым для отладки современных веб-сайтов. Наконец, Нихил Котхари (Nikhil Kothari), сотрудник компании Microsoft (архитектор команды разработчиков ASP.NET AJAX, создатель элемента управления `UpdateHistory` [представленного в главе 10], имеющий также ряд других достижений), написал собственный инструмент под названием Web Development Helper. Загрузить его можно с *http://www.nikhilk.net/projects/WebDevHelperDebuggingTools.aspx*. Это расширение интегрируется в 6 и 7 версии Internet Explorer. Для запуска расширения необходимо выбрать в меню Вид (View)→ Панели обозревателя (Explorer Bar)→Web Develop-

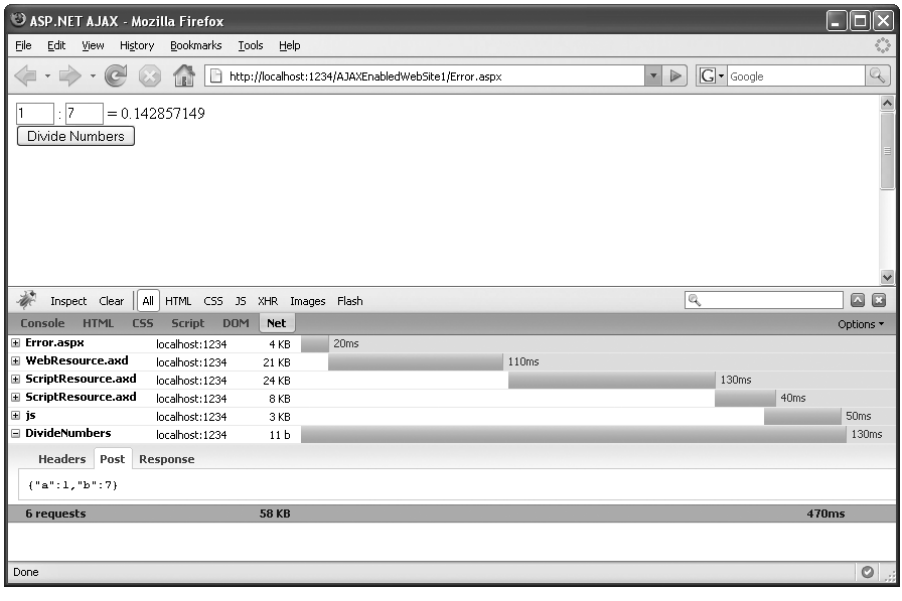


Рис. А.3. *Расширение Firebug для Firefox*

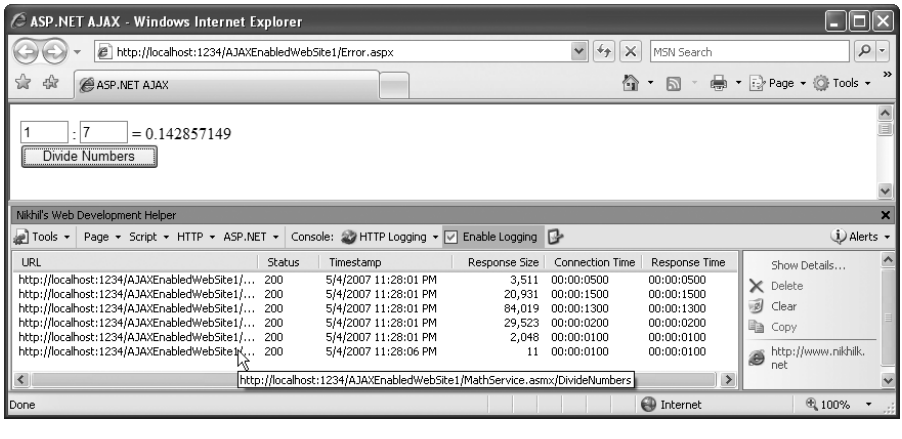


Рис. А.4. *Расширение Web Development Helper для Internet Explorer*

ment Helper. После этого Internet Explorer отобразит окно этой надстройки в верхней части страницы. Расширение Web Development Helper не только обеспечивает доступ к информации в странице, но и обладает некоторыми функциональными возможностями платформы ASP.NET, такими как ViewState и вывод трассировочной информации (особенности, которые в настоящее время отсутствуют в Firebug). С его помощью можно также активизировать возможность журналирования трафика HTTP и даже создавать снимки экрана. Интерфейс расширения Web Development Helper показан на рис. А.4.

Отладка в Visual Studio

В противоположность некоторым мнениям, Visual Studio 2005 обладает возможностями отладки программного кода на JavaScript. Однако здесь есть некоторые проблемы и ловушки, которых вам следует избегать – и вам еще придется отыскать (хорошо спрятанную) возможность отладки сценариев в этой среде разработки компании Microsoft.

Прежде чем приступить к отладке, вам нужно будет проверить настройки броузера. Далее предполагается, что Visual Studio запускает именно Internet Explorer – если вы пользуетесь другим броузером, измените параметр «Browser With» (броузер) так, чтобы для отладки использовался броузер Internet Explorer. В Internet Explorer выберите пункт меню Сервис→Свойства обозревателя (Tools→Internet Options), в открывшемся диалоге перейдите на вкладку Дополнительно (Advanced). В Internet Explorer 6 и 7 необходимо отыскать следующие два параметра:

- Отключить отладку сценариев (Internet Explorer) (Disable Script Debugging (Internet Explorer))
- Отключить отладку сценариев (другие) (Disable Script Debugging (Other))

В более ранних версиях имеется параметр

- Отключить отладку сценариев (Disable Script Debugging)

Независимо от того, один или два параметра отображает ваш броузер, снимите флажки в них, в противном случае отладка в Visual Studio 2005 не будет работать. На рис. А.5 показаны рекомендуемые настройки.

Теперь загрузите в Visual Studio файл *.aspx*, который требуется отладить. Существует три способа заставить Internet Explorer вызвать отладчик в процессе исполнения файла:

- Добавить точку останова из меню Debug или нажав клавишу F9. Однако этот метод не работает с программным кодом JavaScript внутри блоков `<script>` – он годится только для отладки внешних файлов *.js*. (С точками останова связано еще несколько выявленных проблем, о которых будет говориться чуть ниже в этом разделе.)
- Добавить в сценарий JavaScript строку со следующим содержимым: `debugger;` (этот метод работает в Firefox и в Internet Explorer, и не работает в Opera и Safari).
- Использовать методы `Sys.Debug.fail()` и `Sys.Debug.assert()`, которые предоставляются платформой ASP.NET AJAX (раздел «Возможности отладки в ASP.NET AJAX» ниже в этом приложении).

После того как будут выполнены необходимые приготовления, запустите страницу ASP.NET в Visual Studio в режиме отладки (нажатием клавиши F5). Если перед этим вы не запускали отладчик для этого веб-сайта, вам будет предложено добавить в файл *Web.config* дополнительные настройки отладки – примите эти изменения. Дождитесь, по-

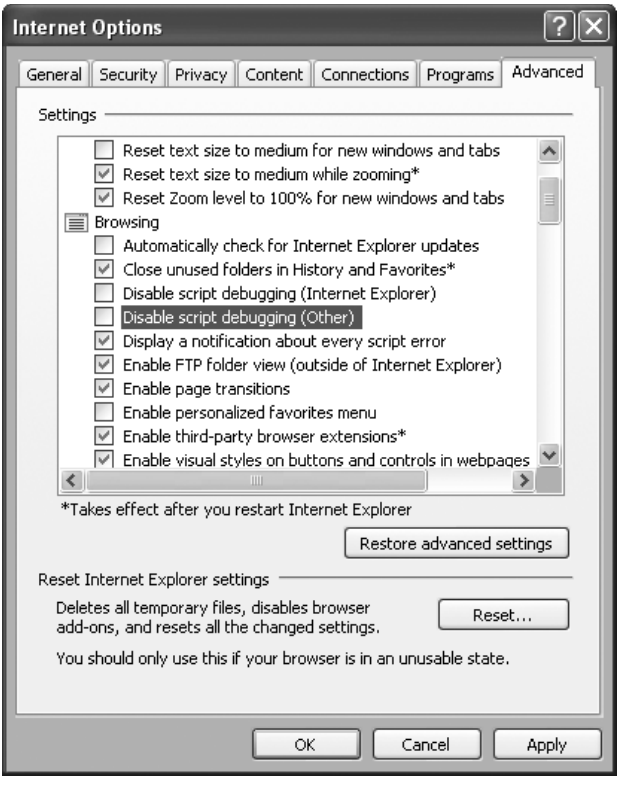


Рис. А.5. *Чтобы использовать возможность отладки, необходимо настроить Internet Explorer*

ка будет достигнута точка останова, в результате – если все идет, как планировалось, – произойдет возврат в Visual Studio либо появится диалог, как показано на рис. А.6.

После этого будет активизирована Visual Studio в режиме отладки. Однако в большинстве случаев вы не увидите текущую строку сценария, вызвавшую выход из броузера в отладчик. Вместо этого вы увидите сообщение: «There is no source code available for the current location» (исходный код для текущего местоположения недоступен) (рис. А.7) или «No debug symbols have been loaded» (отладочная информация не была загружена). У такого поведения может быть несколько причин, включая тот факт, что страница ASP.NET AJAX динамически загружает различные библиотеки JavaScript (то есть файлы сценариев, которые динамически добавляются в дерево DOM текущей страницы и которые не связаны напрямую со страницей при помощи элементов `<script>`).

Если подобная проблема возникла как результат установки точки останова, то ее можно обойти с помощью малоизвестной особенности Visual Studio 2005: *Script Explorer*. Этот инструмент доступен только

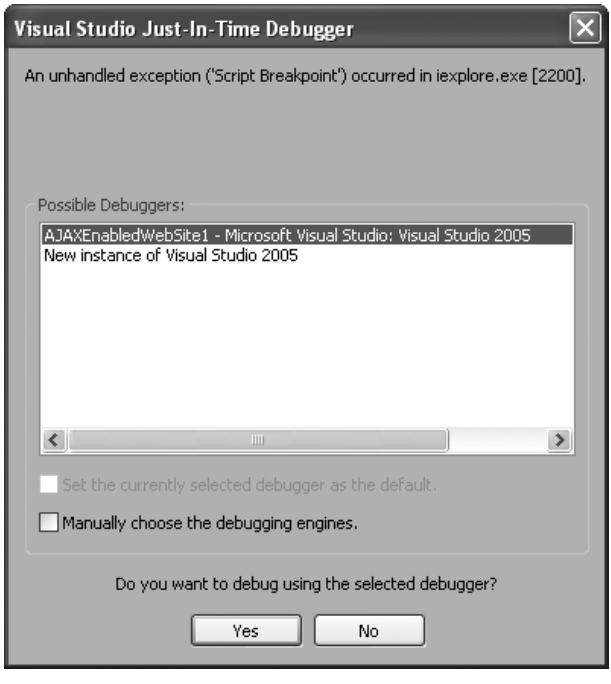


Рис. А.6. *Internet Explorer предлагает перейти в отладчик и предоставляет возможность выбора*

Рис. А.7. *Visual Studio не смогла найти исходный код*

в режиме отладки (это объясняет, почему так мало людей знают о его существовании) через пункт меню Debug→Windows→Script Explorer. Также поддерживается возможность настроить Visual Studio так, чтобы окно Script Explorer отображалось по умолчанию.

Откройте Script Explorer. После этого вы сможете открыть любой файл JavaScript, загруженный текущей страницей, и выполнять следующие действия:

- Произвести пошаговое выполнение кода
- Просмотреть содержимое текущих переменных
- Выполнять команды JavaScript и многое другое

На рис. А.8 показан сеанс отладки с использованием Script Explorer.

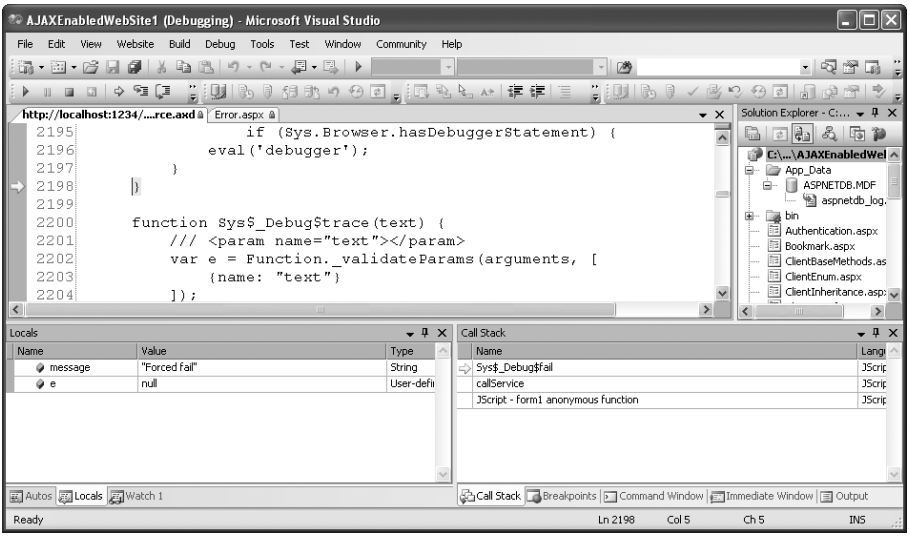


Рис. А.8. *Инструмент Script Explorer из Visual Studio способен работать с программным кодом JavaScript*

Возможности Visual Studio для работы с программным кодом JavaScript будут улучшены в следующей версии (Visual Studio 2008). К моменту написания этих строк уже вышла вторая бета-версия Visual Studio 2008, и вы можете ознакомиться с тем, что вас ожидает в следующей версии.

Возможности отладки в ASP.NET AJAX

Платформа ASP.NET AJAX включает в себя дополнительные возможности отладки, которые прекрасно работают с приемами, уже описанными в этом приложении. Однако, чтобы получить доступ к этим возможностям ASP.NET AJAX, вы должны использовать отладочные версии программного кода JavaScript, используемого вашим приложением.

Сделать это можно двумя способами. Первый: запускать веб-страницы в режиме отладки – либо добавив в файл *Web.config* элемент `<compilation debug="true">`, либо нажав клавишу F5 в Visual Studio. Второй: записать значение `"Debug"` в свойство `ScriptMode` элемента управления `ScriptManager`. В обоих случаях это приведет к тому, что платформа ASP.NET AJAX автоматически перейдет на использование отладочных версий библиотек, которые поддерживаются инструментальными средствами отладки. Разумеется, такой подход годится только для систем, в которых ведется разработка, – в промышленных системах следует использовать (меньшие по объему и потому более производительные) рабочие версии ASP.NET AJAX.

В отладочных версиях библиотек имеется класс `Sys.Debug`, который предоставляет пять методов, удобных для отладки. Большая часть этих методов взаимодействует с консолью JavaScript. Существуют два различных способа, которыми ASP.NET AJAX взаимодействует с этой консолью:

- Отладочные сообщения появляются в консоли JavaScript либо броузера, либо специализированного расширения для броузера (такого как Firefox). Очевидно, что это зависит от используемого броузера и наличия специального расширения.
- Отладочные сообщения появляются внутри HTML-элемента `<textarea>` со значением атрибута ID, равным `TraceConsole`, если таковой элемент имеется.

Класс `Sys.Debug` предоставляет следующие методы:

`Sys.Debug.assert(condition, message, displayCaller)`

Если условие `condition` не соблюдается, в консоль выводится сообщение `message` (включая имя вызывающей функции, если параметр `displayCaller` имеет значение `true`) и запускается отладчик.

Отладка в Firefox

После установки расширения Firebug вы получаете все, что необходимо для отладки, поскольку Firebug включает в себя эти инструменты. Когда сценарий запускает отладчик – например, выполняет инструкцию отладчика – в действие вступает Firebug и отображает текущую исполняемую строку сценария, как показано на рис. А.9.

Для Firefox существует еще один очень интересный отладчик JavaScript, который называется Venkman (его можно загрузить со страницы *https://addons.mozilla.org/firefox/addon/216*). Он предоставляет все, что требуется от отладчика, за исключением возможностей интегрированной среды разработки, которые может предложить Visual Studio. С другой стороны, отладчик Venkman очень удобно интегрируется в броузер. Для запуска отладчика нужно выбрать пункт меню Инструменты→JavaScript Debugger (Tools→JavaScript Debugger).

Обратите внимание: в некоторых версиях Firefox проявляется досадная ошибка при использовании отладчика Venkman – вы можете открыть окно отладчика только один раз в течение сеанса работы с броузером. В этом случае, чтобы повторно запустить отладчик, вам потребуется перезапустить броузер. Поэтому в процессе отладки приложения лучше держать окно отладчика открытым постоянно. На рис. А.10 показано, как выглядит отладчик Venkman.

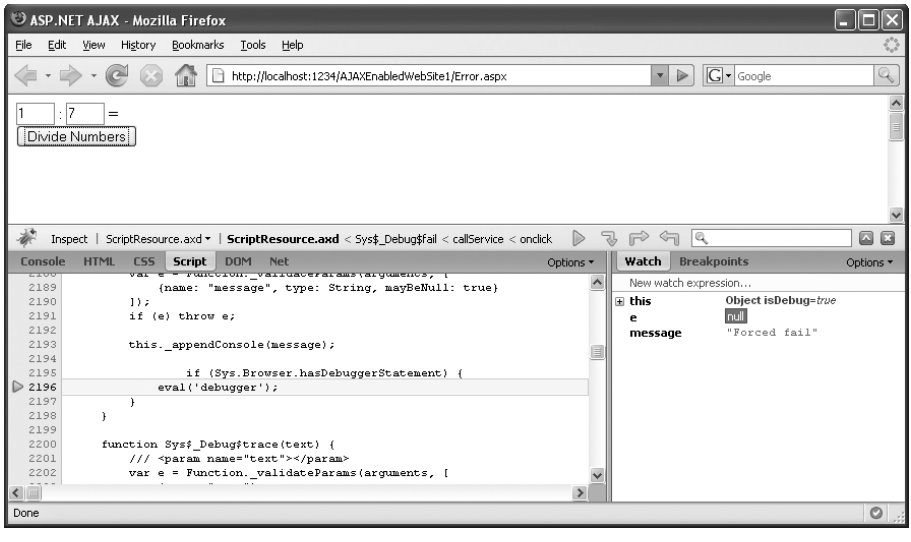


Рис. А.9. *Для Firefox также имеется отладчик JavaScript*

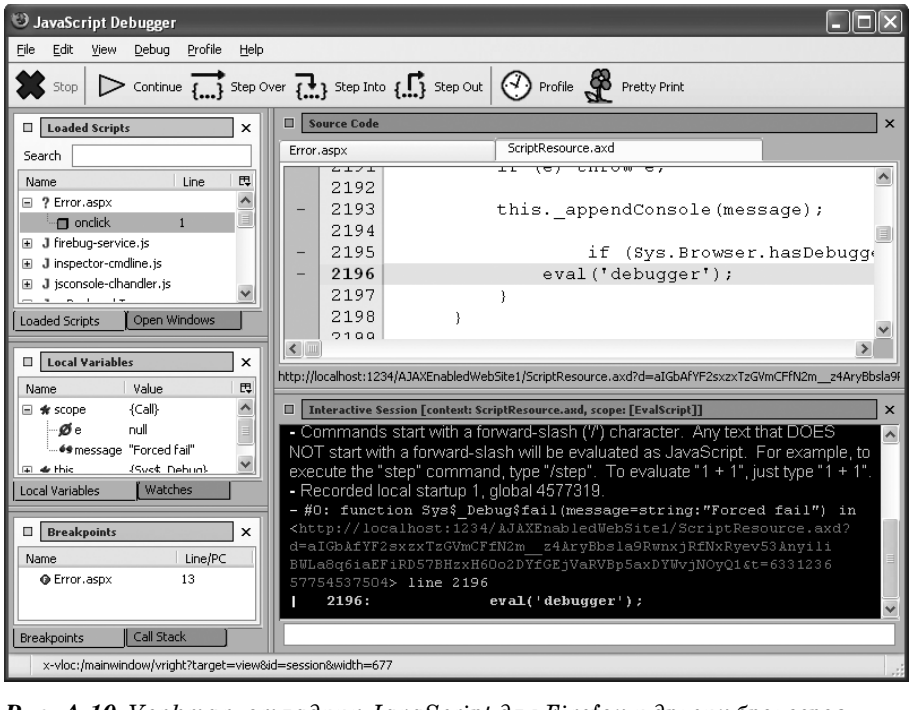


Рис. А.10. *Venkman, отладчик JavaScript для Firefox и других броузеров семейства Mozilla*

`Sys.Debug.clearTrace()`

Выполняется очистка консоли.

`Sys.Debug.fail(message)`

В консоль выводится сообщение `message` и запускается отладчик.

`Sys.Debug.trace(text)`

В консоль выводится содержимое аргумента `text`.

`Sys.Debug.traceDump(object, name)`

В консоль выводится содержимое объекта `object` (включая содержимое всех его членов и необязательного аргумента `name`!).

В примере А.1 демонстрируется использование некоторых из этих методов на базе последнего примера из главы 5, в котором производится вызов веб-службы. Для работы этого примера необходимо присутствие на сервере файла *MathService.asmx* из главы 5 (версия, которая поддерживает метод `ExtendedDivideNumbers()`).

Прежде чем будет вызвана веб-служба, в консоль JavaScript выводится информация, включающая сведения об объекте-посреднике JavaScript `MathService`:

```
Sys.Debug.trace("Calling MathService");
Sys.Debug.traceDump(MathService, "MathService: ");
```

Данные, полученные от службы, также выводятся в консоль:

```
Sys.Debug.trace("Received result!");
Sys.Debug.traceDump(result, "result: ");
```

В случае появления ошибки текст сообщения отображается в консоли JavaScript (в зависимости от типа броузера и его настроек). Затем сценарий выходит в отладчик JavaScript (если таковой имеется):

```
function callError(result) {
  Sys.Debug.fail(
    result.get_exceptionType() +
    ": " +
    result.get_message() +
    "\n" +
    result.get_stackTrace());
}
```

Полный программный код приводится в примере А.1, а на рис. А.11 показано, как могут выглядеть полученные результаты в броузере. Вся прелесть поддержки отладки в ASP.NET AJAX заключается в том, что этот программный код оказывает влияние на страницу только в режиме отладки. Если приложение настроено или страница запускается в рабочем режиме, в консоль JavaScript не передается никаких данных. Благодаря этому отпадает необходимость удалять из приложения отладочный код, исключение составляет элемент `<textarea>`, используемый в качестве консоли (если таковой присутствует). С другой стороны, если

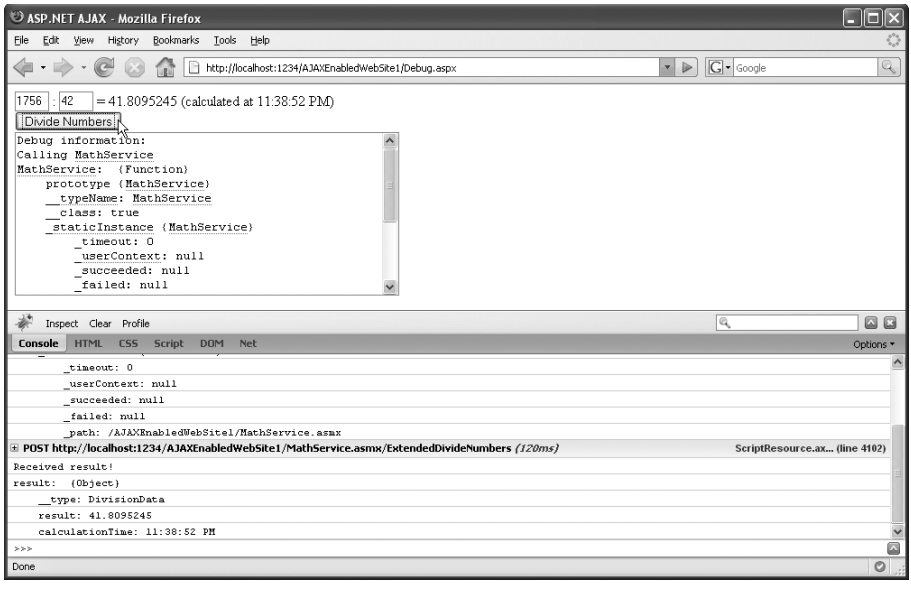


Рис. А.11. *При выводе в консоль JavaScript информация отображается и в текстовом поле, и в окне расширения Firebug*

удалить все вызовы `Sys.Debug.*`, сценарий станет немного меньше, что положительно скажется на производительности страницы.

Пример А.1. Использование отладочных возможностей ASP.NET AJAX

```
Debug.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>ASP.NET AJAX</title>

  <script language="Javascript" type="text/javascript">
    function callService(f) {
      document.getElementById("c").innerHTML = "";
      Sys.Debug.trace("Calling MathService");
      Sys.Debug.traceDump(MathService, "MathService: ");
      MathService.ExtendedDivideNumbers(
        parseInt(f.elements["a"].value),
        parseInt(f.elements["b"].value),
        callComplete,
        callError);
    }
```

```
    function callComplete(result) {
      Sys.Debug.trace("Received result!");
      Sys.Debug.traceDump(result, "result: ");
      document.getElementById("c").innerHTML =
        result.result +
        " (calculated at " +
        result.calculationTime +
        ")";
    }

    function callError(result) {
      Sys.Debug.fail(
        result.get_exceptionType() +
        ": " +
        result.get_message() +
        "\n" +
        result.get_stackTrace());
    }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server"
                       ScriptMode="Debug">
      <Services>
        <asp:ServiceReference Path="MathService.asmx" />
      </Services>
    </asp:ScriptManager>
    <div>
      <nobr>
        <input type="text" id="a" name="a" size="2" />
        :
        <input type="text" id="b" name="b" size="2" />
        =
        <span id="c" style="width: 50px;"></span>
      </nobr>
      <br />
      <input type="button" value="Divide Numbers"
             onclick="callService(this.form);" />
      <br />
      <textarea id="TraceConsole" rows="10" cols="50">Debug information:
      </textarea>
    </div>
  </form>
</body>
</html>
```

Подведение итогов

Это приложение должно было развеять ложное мнение о том, что Ajax-приложения невозможно отлаживать. В действительности существует несколько возможностей и инструментальных средств. Хотя отладка Windows-приложений остается значительно более удобно организованной, однако JavaScript постепенно приближается к этому уровню.

Для дополнительного чтения

http://ajax.asp.net/docs/overview/ASPNETAJAXDebuggingAndTracingOverview.aspx

Документация компании Microsoft, где описываются возможности отладки веб-приложений ASP.NET AJAX. Здесь приводится информация об известных проблемах при отладке в Internet Explorer

http://chrispederick.com/work/webdeveloper/

Еще одно интересное расширение для броузера Firefox, включающее отладчик программного кода JavaScript

B

Справочник по XMLHttpRequest

В этом приложении приводится сводная информация обо всех методах и свойствах объекта XMLHttpRequest. Квадратными скобками [] обозначаются массивы, круглыми () – методы.

Чтобы создать объект XMLHttpRequest для Internet Explorer, необходимо использовать элемент ActiveX:

```
XMLHTTP = new ActiveXObject("Microsoft.XMLHTTP");
```

В остальных броузерах объект XMLHttpRequest является встроенным типом и его экземпляр может быть создан обычным способом, как показано в следующем фрагменте:

```
XMLHTTP = new XMLHttpRequest();
```

После того как объект XMLHttpRequest создан, он поддерживает следующие свойства и методы (для всех типов броузеров):

Методы

Метод	Описание
abort()	Завершает запрос и отбрасывает любой ответ
getAllResponseHeaders()	Возвращает все заголовки из HTTP-ответа
getResponseHeader(header)	Возвращает значение заданного заголовка из HTTP-ответа
open(method, url, async, username, password)	Создает HTTP-запрос и подготавливает его к отправке заданным методом (GET или POST) по заданному URL. Остальные параметры являются необязательными: использовать или не использовать асинхронный (по умолчанию) режим отправки запроса и данные для HTTP-аутентификации. (Обратите внимание: данные для аутентификации

Метод	Описание
	отправляются в простом текстовом виде – если вас беспокоит вопрос безопасности, используйте для работы со страницей протокол HTTPS.)
send(content)	Отправляет HTTP-запрос – в необязательном аргументе можно указать дополнительные данные, которые будут отправлены вместе с запросом (POST-информация)
setRequestHeader(name, value)	Добавляет к HTTP-запросу заголовок с заданным именем и значением

Свойства

Свойство	Описание
readyState	Состояние HTTP-запроса (0=uninitialized, 1=loading, 2=loaded, 3=waiting, 4=complete)
responseText	Данные, полученные в ответ на запрос, в виде текста
responseXML	Данные, полученные в ответ на запрос, в виде объекта XML DOM
status	Код HTTP-статуса HTTP-ответа
statusText	Сообщение HTTP-статуса HTTP-ответа

C

Справочник DOM

В данном приложении приводится сводная информация обо всех свойствах и методах DOM, которые доступны с помощью JavaScript.

Используемая объектная модель документа – это версия W3C DOM, поддерживаемая современными версиями Internet Explorer, Mozilla, Safari/Konqueror и Opera. Методы и свойства, не поддерживаемые двумя «главными» броузерами – Internet Explorer и Mozilla, – здесь не упоминаются.

Квадратными скобками [] обозначаются массивы, круглыми () – методы.

Общие методы и свойства

Методы и свойства, описываемые в этом разделе, имеются у всех элементов DOM.

Методы

Метод	Описание
`appendChild(node)`	Добавляет дочерний узел в элемент
`appendData(data)`	Добавляет в узел данные (код разметки HTML или текст) – существующие данные при этом не затираются
`blur()`	Выносит фокус ввода из элемента
`click()`	Имитирует щелчок мышью на элементе
`cloneNode(deep)`	Создает копию узла (если аргумент `deep` имеет значение `true`, выполняется копирование всех вложенных узлов)
`deleteData(start, length)`	Удаляет заданное число символов из данных узла
`focus()`	Переносит фокус ввода в элемент

Метод	Описание
getAttribute(attribute)	Возвращает значение заданного атрибута
getAttributeNode(attribute)	Возвращает узел, содержащий заданный атрибут
getElementsByTagName(name)	Возвращает массив из всех элементов с заданным именем тега
hasChildNodes()	Возвращает значение, которое указывает, имеет ли данный элемент вложенные узлы
insertBefore(node)	Вставляет узел перед элементом
insertData(position, data)	Вставляет данные (код разметки HTML или текст) в указанную позицию
removeAttribute(attribute)	Удаляет из элемента указанный атрибут и его значение
removeChild(node)	Удаляет из элемента указанный вложенный узел
replaceChild(newnode, oldnode)	Замещает указанный старый вложенный узел (oldnode) новым (newnode)
replaceData(start, length, newdata)	Замещает данные (начиная с указанной позиции, заданной длины) на новые
setAttribute(name, value)	Присваивает заданное значение указанному атрибуту
setAttributeNode(node)	Добавляет новый атрибут в виде узла, замещая существующий

Свойства

Свойство	Описание
attributes[]	Массив атрибутов элемента и их значений (для чтения/записи)
childNodes[]	Список вложенных узлов элемента (для чтения/записи)
className	Имя CSS-класса элемента (для чтения/записи)
data	Символьные данные в текстовом узле (для чтения/записи)
dir	Направление чтения для элемента (для чтения/записи)
firstChild	Первый вложенный узел элемента (для чтения)
id	Идентификатор элемента (для чтения/записи)
innerHTML	Разметка HTML, содержащаяся в элементе (не соответствует стандартам W3C, но реализовано в большинстве броузеров) (для чтения/записи)
lang	Язык элемента (атрибут lang) (для чтения/записи)
lastChild	Последний вложенный узел элемента (для чтения)
length	Длина элемента (для чтения/записи)
localeName	Код региональных настроек элемента (без пространства имен, если таковое указано) (для чтения/записи)

Свойство	Описание
namespaceURI	Идентификатор URI пространства имен элемента (для чтения/записи)
nextSibling	Элемент, расположенный следующим после текущего в дереве DOM (для чтения/записи)
nodeName	Имя тега элемента узла (для чтения/записи)
nodeType	Тип узла или элемента узла (для чтения/записи)
nodeValue	Значение в узле элемента (для чтения/записи)
ownerDocument	Документ, которому принадлежит данный элемент (для чтения)
parentNode	Родительский узел элемента (для чтения)
prefix	Префикс пространства имен, используемого узлом (для чтения/записи)
previousSibling	Элемент, расположенный перед текущим в дереве DOM (для чтения/записи)
style	Информация о стилях элемента (для чтения/записи)
tabIndex	Порядковый номер элемента для передачи фокуса ввода клавишей табуляции (для чтения/записи)
tagName	Имя тега элемента (для чтения/записи)
title	Заголовок элемента (для чтения/записи)

Методы и свойства документа

В этом разделе приводятся методы и свойства, реализованные в объекте document. Методы и свойства, описанные в предыдущем разделе, опущены.

Методы

Метод	Описание
clear()	Очищает содержимое документа
close()	Закрывает доступ на запись (открытый методом open()) в документ
createAttribute(attribute)	Создает атрибут с указанным именем
createDocumentFragment()	Создает фрагмент документа
createElement(name)	Создает элемент с заданным именем тега
createTextNode(text)	Создает текстовый узел с заданным текстом
getElementById(id)	Возвращает элемент с указанным идентификатором
getElementsByTagName(name)	Возвращает массив всех элементов с указанным именем тега
open(mime, replace)	Открывает документ для записи, устанавливает MIME-тип и, если необязательный параметр replace

Метод	Описание
	имеет значение true, замещает прежнее содержимое документа (в противном случае новое содержимое добавляется в конец документа)
write(text)	Записывает данные в документ
writeln(text)	Записывает данные и комбинацию символов перевода строки (\r\n) в документ

Свойства

Свойство	Описание
alinkColor	Цвет активной ссылки (для чтения/записи)
anchors[]	Массив всех якорных элементов в документе (для чтения)
applets[]	Массив всех Java-апплетов в документе (для чтения)
bgColor	Цвет фона документа (для чтения/записи)
body	Тело документа (для чтения/записи)
compatMode	Значение, которое указывает, должен ли механизм отображения использовать режим совместимости с прежними версиями Internet Explorer (для чтения/записи)
cookie	Cookies, которые доступны из документа (для чтения/записи)
documentElement	DOM-узел документа (для чтения/записи)
domain	Домен документа (для чтения)
embeds[]	Массив всех внедренных объектов (для чтения/записи)
fgColor	Цвет переднего плана (текста) документа (для чтения/записи)
forms[]	Массив всех форм в документе. В страницах ASP.NET может присутствовать только одна форма, которая всегда соответствует элементу массива forms[0] (для чтения/записи)
images[]	Массив всех изображений в документе (для чтения)
lastModified	Дата и время последнего изменения документа на сервере (для чтения)
linkColor	Цвет ссылок (для чтения/записи)
links[]	Массив всех ссылок в документе (для чтения)
location	Информация о строке URL документа (для чтения)
referrer	URL документа, из которого пользователь перешел к текущему документу (для чтения)
styleSheets[]	Массив всех каскадных таблиц стилей, ссылки на которые имеются в документе (для чтения)
URL	URL документа (для чтения)
vlinkColor	Цвет для посещавшихся ссылок (для чтения/записи)

D

Справочник по ASP.NET AJAX

Платформа ASP.NET AJAX предоставляет несколько вспомогательных прикладных интерфейсов на JavaScript и расширяет функциональность существующих объектов JavaScript. В этом приложении приводится выборочный список наиболее важных функций и методов.

Вспомогательные функции

Платформа ASP.NET AJAX имеет несколько удобных вспомогательных функций. Наиболее важные из них имеют сокращенные имена, начинающиеся с символа доллара ($).

Свойство	Описание
`$addHandler(element, eventName, handler);`	Добавляет в элемент DOM обработчик указанного события
`$addHandlers(element, events, handlerOwner)`	Добавляет в элемент DOM несколько обработчиков событий
`$clearHandlers(element)`	Удаляет все обработчики событий, присоединенные к элементу DOM
`$create(type, properties, events, references, element)`	Создает компонент и инициализирует его
`$find(id, parent)`	Отыскивает компонент с указанным идентификатором
`$get(id, element)`	Отыскивает элемент DOM с указанным идентификатором
`$removeHandler(element, eventName, handler)`	Удаляет указанный обработчик события из элемента DOM

Расширения объектов

Платформа ASP.NET AJAX расширяет некоторые стандартные объекты JavaScript, такие как строки или логические значения, дополнительными методами. Расширения JavaScript-объекта `Error` используются преимущественно внутренней реализацией платформы, однако расширения других объектов также могут использоваться программистами при разработке своего кода.

Расширения класса Array

Метод	Описание
`add(array, item)`	Добавляет элемент в массив
`addRange(array, items)`	Добавляет несколько элементов в массив
`clear(array)`	Очищает содержимое массива
`clone(array)`	Создает копию массива
`contains(array, item)`	Проверяет, содержит ли массив указанный элемент
`dequeue(array)`	Удаляет (и возвращает) первый элемент массива
`enqueue(array, item)`	Добавляет элемент в массив (этот метод не должен использоваться непосредственно, вместо него следует вызывать метод `add()`)
`forEach(array, method, contxt)`	Выполняет обход всех элементов и для каждого из них вызывает указанный метод
`indexOf(array, item, start)`	Возвращает индекс указанного элемента в массиве (индексация начинается с 0) или -1, если такой элемент отсутствует в массиве
`insert(array, index, item)`	Добавляет новый элемент в массив в указанную позицию
`parse(value)`	Преобразует строку формата JSON в массив
`remove(array, item)`	Удаляет элемент из массива
`removeAt(array, index)`	Удаляет из массива элемент с указанным индексом

Расширения класса Boolean

Имеющийся метод класса Boolean должен вызываться статически (Boolean.parse()).

Метод	Описание
`parse(value)`	Преобразует указанное значение в логическое значение

Расширения класса Date

Все методы класса `Date` должны вызываться в контексте конкретного экземпляра класса, если не указано иное (например: `new Date().format()`).

Метод	Описание
`format(format)`	Форматирует дату в соответствии с заданной строкой формата
`localeFormat(format)`	Форматирует дату в соответствии с заданной строкой формата и текущими региональными настройками; может также вызываться статически
`parseLocale(value, formats)`	Выполняет разбор строки с датой в соответствии с текущими региональными настройками; может также вызываться статически
`parseInvariant(value, formats)`	Выполняет разбор строки с датой в соответствии с инвариантной культурой; может также вызываться статически

Расширения класса Number

Все методы класса `Number` должны вызываться в контексте конкретного экземпляра класса, если не указано иное.

Метод	Описание
`format(format)`	Форматирует число в соответствии с инвариантной культурой
`localeFormat(format)`	Форматирует число в соответствии с текущими региональными настройками
`parseInvariant(value)`	Выполняет разбор строки с числом в соответствии с инвариантной культурой; может вызываться статически
`parseLocale(value)`	Выполняет разбор строки с числом в соответствии с текущими региональными настройками; может вызываться статически

Расширения класса Object

Все методы класса `Object` должны вызываться статически (например: `Object.getType()`).

Метод	Описание
`getType(instance)`	Возвращает тип экземпляра класса
`getTypeName(instance)`	Возвращает имя типа экземпляра класса

Расширения класса String

Все методы класса `String` должны вызываться в контексте конкретного экземпляра класса, если не указано иное.

Метод	Описание
`endsWith(suffix)`	Определяет, заканчивается ли строка указанным суффиксом
`format(format, args)`	Замещает символы-заполнители `{0}`, `{1}`... в строке формата значениями, переданными в метод в виде дополнительных аргументов, с учетом текущих региональных настроек для дат и чисел; должен вызываться статически
`startsWith(prefix)`	Определяет, начинается ли строка указанным префиксом
`trim()`	Удаляет ведущие и завершающие пробельные символы в строке
`trimLeft()`	Удаляет ведущие пробельные символы в строке
`trimRight()`	Удаляет завершающие пробельные символы в строке

E

Справочник по ScriptManager, UpdatePanel, UpdateProgress и Timer

В этом приложении описываются свойства четырех наиболее важных серверных элементов управления платформы ASP.NET AJAX: `ScriptManager`, `UpdatePanel`, `UpdateProgress` и `Timer`. Здесь описаны все имеющиеся свойства (которые используются при декларативном подходе), за исключением свойств `ID` и `runat="server"`. Кроме того, здесь были опущены свойства, унаследованные от элемента управления `Control`.

ScriptManager

Элемент управления ScriptManager является самым важным для сайтов, построенных на базе платформы ASP.NET AJAX, поскольку он отвечает за загрузку клиентских библиотек, а также занимается созданием объектов-посредников.

Свойства

Свойство	Описание
`AllowCustomErrorsRedirect`	Разрешает (*true*, по умолчанию) или запрещает (*false*) перенаправление ошибок (определяемое в файле *Web.config*), возникающих на сервере в процессе обработки асинхронных запросов
`AsyncPostBackErrorMessage`	Текст сообщения об ошибке, возникшей в процессе выполнения асинхронного запроса
`AsyncPostBackSourceElementID`	Идентификатор элемента, инициировавшего асинхронный запрос
`AuthenticationService-Path`	Путь к службе аутентификации

Свойство	Описание
EnablePageMethods	Разрешает (*true*) или запрещает (*false*, по умолчанию) использование статических встроенных методов (методов страницы)
EnablePartialRendering	Разрешает (*true*, по умолчанию) или запрещает (*false*) возможность частичного отображения, реализованную в UpdatePanel
EnableScriptGlobalization	Разрешает (*true*) или запрещает (*false*, по умолчанию) использование региональных настроек из ASP.NET AJAX
EnableScriptLocalization	Разрешает (*true*) или запрещает (*false*, по умолчанию) использование локализованного содержимого из ASP.NET AJAX
LoadScriptsBeforeUI	Определяет, когда следует загружать библиотеки JavaScript – до (*true*, по умолчанию) или после (*false*) разметки страницы
ProfileService-LoadProperties	Свойства профиля, доступные клиентскому сценарию
ProfileService-Path	Путь к веб-службе управления профилями
ScriptMode	Определяет, какую версию библиотек следует использовать – отладочную или рабочую
ScriptPath	Корневой каталог для пользовательских библиотек JavaScript и библиотек ASP.NET AJAX

AuthenticationService

Элемент <Authentication>, дочерний по отношению к элементу управления ScriptManager используется для организации поддержки службы аутентификации. Обладает следующим свойством:

Свойство	Описание
Path	Путь и имя файла веб-службы аутентификации

ProfileService

Элемент <ProfileService>, дочерний по отношению к элементу управления ScriptManager используется для организации поддержки службы управления профилями. Обладает следующим свойством:

Свойство	Описание
LoadProperties	Содержит список свойств профиля, которые должны быть доступны клиентскому сценарию

Scripts

Элемент <Scripts>, дочерний по отношению к элементу управления ScriptManager содержит все клиентские сценарии, которые будут загружены с помощью элемента управления <asp:ScriptReference>. Обладает следующими свойствами.:

Свойство	Описание
Assembly	Сборка, в которой содержится сценарий
IgnoreScriptPath	Определяет, следует игнорировать (*true*) свойство ScriptPath элемента управления ScriptManager или нет (*false*, по умолчанию)
Name	Имя используемого встроенного ресурса сценария
NotifyScriptLoaded	Определяет, следует (*true*, по умолчанию) или нет (*false*) автоматически добавлять в конец сценария программный код, который будет извещать элемент управления ScriptManager о том, что сценарий был загружен
Path	Путь и имя файла загружаемого сценария
ResourceUICultures	Список используемых культур пользовательского интерфейса
ScriptMode	Определяет, какую версию сценария следует использовать – отладочную (Debug) или рабочую (Release)
Path	Путь и имя файла веб-службыуправления профилями

UpdatePanel

Благодаря элементу управления UpdatePanel можно выполнять обновление отдельного раздела страницы с поддержкой ASP.NET AJAX независимо от остальной части страницы; содержимое должно размещаться в элементе <ContentTemplate>, вложенном в <asp:UpdatePanel>.

Свойства

Свойство	Описание
ChildrenAsTriggers	Определяет, должно ли (*true*, по умолчанию) обновляться содержимое элемента UpdatePanel при получении данных дочерними элементами или нет (*false*)
RenderMode	Определяет, как должно отображаться содержимое элемента управления UpdatePanel – в виде элемента (Block, по умолчанию) или (Inline)
UpdateMode	Определяет, когда выполнять обновление, всегда (Always, то есть при любом получении данных) или по условию (Conditional, то есть только принудительно)

Triggers

Элемент <Triggers>, дочерний по отношению к элементу управления UpdatePanel, содержит объекты-триггеры, которые могут вызвать обновление содержимого элемента управления UpdatePanel. Имеются два триггера — <asp:AsyncPostBackTrigger> и <asp:PostBackTrigger>, со следующими свойствами:

Свойство	Описание
ControlID	Идентификатор элемента управления, который может вызвать обработчик
EventName	Событие, которое возникает при срабатывании триггера (только для <asp:AsyncPostBackTrigger>)

UpdateProgress

Элемент управления UpdateProgress отображает экранную заставку (*waiting screen*), пока идет обновление элемента UpdatePanel. Фактическое содержимое заставки размещается в элементе <ProgressTemplate>, вложенном в элемент <asp:UpdateProgress>.

Свойства

Свойство	Описание
AssociatedUpdatePanelID	Идентификатор элемента управления UpdatePanel, ассоциированного с данным элементом
DisplayAfter	Длительность интервала времени в миллисекундах, через которое будет показана экранная заставка (по умолчанию 500)
DynamicLayout	Определяет, следует ли резервировать пространство на странице под заставку (false) или место должно выделяться динамически, только после того, как в этом появится необходимость (true, по умолчанию)

Timer

Элемент управления Timer запускает события и вызывает передачу данных через заданные интервалы времени.

Свойства

Свойство	Описание
Enabled	Разрешает (*true*, по умолчанию) или запрещает (*false*) срабатывания таймера
Interval	Интервал времени в миллисекундах между срабатываниями таймера (по умолчанию 60000)

Алфавитный указатель

A

Accordion, элемент управления, 234
addCssClass(), метод, 298
$addHandler(), функция, 94, 263
$addHandlers(), функция, 94
AddHistoryPoint(), метод, 440
AdventureWorks, база данных, 148
  установка, 32
Ajax
  JSON, 86
  архитектура, 23
  и ASP.NET AJAX, 23
  объект XMLDocument, 81
  объект XMLHttpRequest, 69
AjaxControlExtender, папка, 212
Always, режим, 155
AlwaysVisibleControlExtender, элемент управления, 236
Amazon, веб-служба, 393
AnimationChildren, свойство, 227
AnimationExtender, элемент управления, 221, 226
<anonymousIdentification>, элемент, 165
API (Application Programming Interface), служба аутентификации, 180
applyPageState(), функция, 433
Array, класс
  расширения, 486
.asmx файл, 121
ASP.NET, 25
  Configuration, команда, 177
  Web Application Administration Tool, 177
  веб-части, 448
  и ASP.NET AJAX, 25
ASP.NET AJAX
  ASP.NET Web Parts, 448
  Control Toolkit
    Accordion, элемент управления, 234
    использование, 216
    пакет элементов управления, 211
    содействие проекту, 268
    создание собственных компонентов, 258
    установка, 212
  get_text(), метод, 287
  ScriptManager, элемент управления, 39
  set_text(), метод, 287
  анимация
    группировка, 427
    перемещение элемента, 422, 425
  архитектура, 34
  аспекты поведения
    Sys.Preview.UI.ClickBehavior, 342
    Sys.Preview.UI.FloatingBehavior, 346
    использование, 341
  аутентификация
    подготовка приложения, 177
  базовые методы, 297
  гиперссылки, 290
  данные на стороне сервера, элемент управления ListView, 359
  доступ к сгруппированным данным в профиле, 171
  загрузка библиотек, 41
  использование с PHP, 458
  клиентские элементы управления, 281
    использование, 282
    обработка событий, 300
  клиенты, 111

компоненты, 355
объектно-ориентированные возможности JavaScript, 97
отладка, 472
перетащить-и-отпустить, аспект поведения, 348
привязка данных, 305
примеры, 35
профили
доступ к данным, 166
подготовка веб-сайта, 165
псевдонимы, 93
расширения к существующим объектам JavaScript, 96
сопутствующие ресурсы, 192
справочник, 485
установка, 28
элементы управления
Select, 294
кнопки, 292
списки выбора, 293
текстовые поля, 296
флажки, 292
AssemblyInfo.cs, файл, 197
asyncInvoke(), метод, 139
AsyncPostBackTrigger, свойство, 153
Atlas, 23
Authentication.aspx, файл, 181
<authenticationService>, элемент, 179
AutoCompleteExtender, элемент управления, 238
AWSECommerceServiceWrapper, класс, 394
AWSECommerceServiceWrapper.cs, файл, 394

B

BackgroundCssClass, свойство, 250
baseArguments, параметр, 106
<binding>, элемент, 315
Boolean, класс расширения, 486

C

CalendarExtender, элемент управления, 246
callComplete(), функция, 413
callService(), метод, 136
$clearHandlers(), функция, 94
CollapseControlID, свойство, 247
CollapsiblePanelExtender, элемент управления, 247
Conditional, режим, 155
ConfirmButton, элемент управления, 216, 217
<ContentTemplate>, элемент, 254
Control Toolkit
Accordion, элемент управления, 234
использование, 216
пакет ASP.NET AJAX, 27
пакет элементов управления, 211
платформа создания анимационных эффектов, 220
содействие проекту, 268
создание собственных компонентов, 258
установка, 212
ControlID, свойство, 153
cookie, сеансовый, 129
Cross-Site Scripting (XSS), 433
CSS (Cascading Style Sheets, 24
Accordion, элемент управления, 234
методы классов, 96
позиционирование HTML-элементов, 236
CTP (Community Technology Preview), пакет ASP.NET AJAX, 27
customValidator, валидатор, 323

D

dataPath, свойство, 370
DataRowColection, объект, 361
DataTable, класс, 378
Date, класс расширения, 487
dateformat, переменная, 189
daynames, переменная, 189
Development Branch, для загрузки, 271
, элемент, 119
DivideNumbers(), метод, 118, 129
doClick(), функция, 181
document, объект, 61
document.getElementById(), метод, 93
DOM (Document Object Model), 24
справочник, 481
элементы, 96
DragHandleID, свойство, 229
DragOverlayExtender, 348
DragPanelExtender, элемент управления, 229
_duration, свойство, 418

E

Enabled, свойство, 349
EnableScriptLocalization, атрибут, 201
encode(), метод, 139
eval(), функция, 88
EventName, свойство, 153
ExpandControlID, свойство, 247
Extensions, пакет ASP.NET AJAX, 27

F

<FadeOut>, элемент, 221
failedCallback, параметр, 167
Fiddler, 139
Firebug, расширение для броузера Firefox, 467
Fireburg, расширение броузера Firefox, 173
Firefox, броузер
    отладка, 473
<floatingBehavior>, элемент, 346
for, цикл, 52
for...in, цикл, 53
_fps, 418
Futures
    клиентские элементы управления, 281
Futures CTP, пакет, установка, 33
Futures Release, пакет ASP.NET AJAX, 27

G

GAC (Global Assembly Cache), 31
get_accessKey(), метод, 298
get_alternateText(), метод, 288
get_argument(), метод, 292
get_command(), метод, 292
getCurrentPageState(), функция, 433
get_exceptionType(), метод, 119
get_height(), метод, 288
get_imageURL(), метод, 289
get_message(), метод, 119
GetPropertyValue(), метод, 261
get_selectedValue(), метод, 296
get_stackTrace(), метод, 119
get_statusCode(), метод, 119
get_text(), метод, 287
get_timeOut(), метод, 119
get_width(), метод, 288
getXMLHTTP(), функция, 72
Global Assembly Cache (GAC), 31
Google Suggest, веб-приложение, 238
Google Web API Developer's Kit (набор инструментальных средств разработчика Google Web API), 387
Google веб-службы, 385
    файл XSL-преобразования, 400
GoogleSearch.wsdl, файл, 387
gotoStepClick(), функция, 443
GridView, элемент управления, 147

H

HeaderText, свойство, 254
Hello User, пример, 35
HistoryEventArgs, объект, 440
historyNavigate(), метод, 436
HorizontalSide, свойство, 236
.htc, файлы HTML control (элемент управления HTML), 136
HTML (Hypertext Markup Language)
    привязка данных, 305
    привязка данных к HTML-списку, 371
    привязка данных к HTML-таблице, 373
    списки выбора, 373
    элементы, 314
        доступ, 286
        позиционирование, 236
HTTP (Hypertext Transfer Protocol)
    заголовок Accept-Language, 203
    объект XMLHttpRequest, 69
    создание запроса, 73
HttpContext.Current.Session, 125

I

id, атрибут, 320
<iframe>, элемент, 434
Image, элемент управления, 288
initialize(), метод, 263
inputBox(), метод, 283
instance, параметр, 106
IntelliSense, 27
Internet Explorer
    Visual Studio 2005, отладка, 469
    Web Development Helper, 467
    и веб-службы, 135
    объект XMLHttpRequest, 70
<invokeMethodAction>, тег, 316
ItemSearchRequest, объект, 394

<itemTemplate>, элемент, 360
itemTemplateParentElementId, атрибут, 369

J

JavaScript
ASP.NET AJAX
клиенты, 111
объектно-ориентированные возможности JavaScript, 97
псевдонимы, 93
расширения к существующим объектам, 96
анонимные функции, 76
аутентификация, 183
встраивание, 46
встроенные объекты, 54
доступ к веб-службам, 134
доступ к элементам страницы, 61
доступ к элементам форм, 61
массивы, 49
методы, 47
доступ, 283
модальные диалоги, 283
наследование, 59
обзор, 46
переменные, 48
программирование анимационных эффектов, 227
расширители, 264
управляющие конструкции, 49
JSON (JavaScript Object Notation)
сериализация и десериализация данных, 130
формат представления данных, 86

L

Label, элемент управления, 148, 287
добавление, 148
<layoutTemplate>, элемент, 360
LengthAnimation, перемещение элементов, 425
ListView, элемент управления, 359
LiveScript, 44
Load Vendors, кнопка, 372
load(), метод, 168, 173
loadCompletedCallback, параметр, 167
LocDateControl, класс, 199
login(), метод, 180
loginComplete(), функция, 182
loginError(), функция, 182
logout(), метод, 180

M

mashups, гибридные веб-приложения, 23, 384
MathService.asmx, файл, 131
messageBox(), метод, 283
methodName, атрибут, 320
methodName, параметр, 106
Microsoft Ajax Library, библиотека, 26, 27
Microsoft SQL Server Management Studio Express (SSMSE), 32
Microsoft.Web.Preview.UI.Controls.WebParts.WebPartManager, пространство имен, 450
Mocha, 44
ModalPopupExtender, элемент управления, 250
Mozilla
и веб-службы, 137
объект XMLHttpRequest, 71
отладка, 473

N

.NET
клиентские версии классов, 111
управление состоянием сеанса, 125
NoBot, элемент управления, 252
NoSource, ZIP-архив, 212
Number, класс
расширения, 487

O

Object, класс
расширения, 487
OKControlID, свойство, 250
onreadystatechange, свойство, 74
open(), метод, 73
OWA (Outlook Web Access), 69

P

pageLoad(), метод, 368
pageLoad(), функция, 95, 181
Page_Load, обработчик события, 76
PageMethods, класс, 122
pageNavigate(), функция, 442

PageRequestManager, класс, 159
<Panes> элемент, 234
PHP, использование с ASP.NET AJAX, 458
PHPHelloWorldService, класс, 459
PHPHelloWorldService.php, файл, 459
play(), метод, 418
PopupControlID, свойство, 250
POST, команда, 78
PostBackTrigger, свойство, 153
preventDefault(), метод, 264
PreviewDragDrop.js, файл, 346
PreviewGlitz.js, файл, 417
PreviewScript.js, файл, 346
ProcessRequest(), метод, 459
profileError(), функция, 168, 174
profileLoaded(), функция, 168
ProfileService, класс, 166
<ProgressTemplate>, шаблон, 157
property, атрибут, 317
property, свойство, 370
propertyKey, атрибут, 317
propertyNames, параметр, 166
<proxy>, элемент, 389

R

rangeValidator, валидатор, 323
registerClass(), метод, 99
removeCssClass(), метод, 298
$removeHandler(), функция, 94
Repeater, элемент управления, 361
requiredFieldValidator, валидатор, 323
responseText, свойство, 75
responseXML, свойство, 75
REST, 407

S

SampleWebSite, папка, 212
saveProfile(), функция, 169
SaveTime(), метод, 126
Script Explorer, средство отладки, 470
<script>, тег, 46
ScriptManager, серверный элемент управления, 34
ScriptManager, элемент управления, 39, 489
ScriptMethod, атрибут, 122
Search(), метод, 390
SearchXslt(), метод, 401
Select, элемент управления, 294
send(), метод, 74
set_accessKey(), метод, 298
set_alternateText(), метод, 288
set_argument(), метод, 292
set_command(), метод, 292
set_direction(), метод, 310
set_height(), метод, 288
set_imageURL(), метод, 289
set_OnClick(), метод, 228
<setPropertyAction>, тег, 316
SetPropertyValue(), метод, 261
set_text(), метод, 287
set_transformerArgument(), метод, 307
set_width(), метод, 289
SmartTag, поддержка, 148
SOAPCall, класс, 138
, элемент, 119
SQL (Structured Query Language), предотвращение инъекции, 242
SQL Server Express, установка, 32
SSMSE (Microsoft SQL Server Management Studio Express), 32
stateChanged(), функция, 75
String, класс
  расширения, 488
StringBuilder, класс, 111
Sys.Debug, класс, 473
Sys.Preview.Binding, класс, 306
Sys.Preview.BindingBase.Transformers.Adds, преобразователь, 307
Sys.Preview.BindingBase.Transformers.Compare, преобразователь, 307
Sys.Preview.BindingBase.Transformers.CompareInverted, преобразователь, 307
Sys.Preview.BindingBase.Transformers.Invert, преобразователь, 307
Sys.Preview.BindingBase.Transformers.Multiply, преобразователь, 307
Sys.Preview.BindingBase.Transformers.ToString, преобразователь, 307
Sys.Preview.BindingDirection.In, направление привязки, 310
Sys.Preview.BindingDirection.InOut, направление привязки, 310
Sys.Preview.BindingDirection.Out, направление привязки, 310
Sys.Preview.UI, пространство имен, 281
Sys.Preview.UI.ClickBehavior, аспект поведения, 342

Sys.Preview.UI.Effects.ColorAnimation, объект, 418
Sys.Preview.UI.Effects.CompositeAnimation, объект, 418
Sys.Preview.UI.Effects.DiscreteAnimation, объект, 418
Sys.Preview.UI.Effects.FadeAnimation, объект, 418
Sys.Preview.UI.Effects.InterpolatedAnimation, объект, 418
Sys.Preview.UI.Effects.LengthAnimation, объект, 418
Sys.Preview.UI.Effects.NumberAnimation, класс, 422
Sys.Preview.UI.Effects.NumberAnimation, объект, 418
Sys.Preview.UI.Effects.PropertyAnimation, объект, 418
Sys.Preview.UI.FloatingBehavior, аспект поведения, 342, 346
Sys.Preview.UI.LayoutBehavior, аспект поведения, 342
Sys.Preview.UI.OpacityBehavior, аспект поведения, 342
Sys.Preview.UI.Window, класс, 283
<system.web>, элемент, 165
<system.web.extensions>, элемент, 166
System.Web.UI.WebControls.WebParts.WebPartManager, пространство имен, 450

T

TabContainer, элемент управления, 254
TabPanel, элемент управления, 254
<tagMapping>, элемент, 449
target, атрибут, 317
_target, свойство, 419
TargetControlID, свойство, 221, 229, 247, 250, 349
<template>, элемент, 370
text/xml-script, тип, 314
TextBox, элемент управления, функция автодополнения, 237
TextBoxMaskBehavior.js, файл, 260, 262
TextBoxMaskDesigner, класс, 260
TextBoxMaskDesigner.cs, файл, 260
TextBoxMaskExtender, класс, 261
TextBoxMaskExtender.cs, файл, 260
Tick, событие, 153
Timer, элемент управления, свойства, 492
toggleCssClass(), метод, 298
toString(), метод, 60
  добавление, 105
typeof, оператор, 71
typeValidator, валидатор, 323

U

Update(), метод, 154
UpdateHistory, элемент управления, 435
UpdatePanel, элемент управления, 146, 436
  область страницы, 146
  свойства, 491
UpdateProgress, элемент управления, свойства, 492
URL (Uniform Resource Locator), JavaScript, 46
url, атрибут, 320
useGet, атрибут, 320
userContext, параметр, 167
UserData, свойство, 165
userName, свойство, 165
useService(), метод, 136

V

value, атрибут, 317
VerticalSide, свойство, 236
Virtual Earth, веб-приложение с поддержкой Ajax, 24
visibilityMode, свойство, 330
Visual Studio 2005, 28
  отладка, 469
  установка, 28
Visual Studio Integration, пакет интеграции с Visual Studio, 212
Visual Web Developer Express Edition, установка, 28

W

WCF (Windows Communication Foundation), 117
Web Development Helper, расширение для Internet Explorer, 467
Web.config, файл, 165
WebPartManager, элемент управления, 450
WebPartZone, элемент управления, 450

while, цикл, 53
window.setInterval(), метод, 153
window.setTimeout(), метод, 153
WPF (Windows Presentation Foundation), 306
WSDL (Web Services Description Language), 135

X

XML (Extensible Markup Language)
  дерево, 81
XMLDocument, объект, 81
XMLHttpRequest, объект, 69, 479
  справочник, 479
xml-script, 25, 313, 315
  вызов методов, 321
  изменение значений свойств, 317
  эффект плавного исчезновения, 421
XPath, 407
XSS (Cross-Site Scripting), 433

Y

Yahoo!, веб-служба, 407

А

автодополнения функция в элементе управления TextBox, 237
активация профилей, 165
анимация
  использование, 417
  перемещение элемента, 422, 425
  перетащить-и-отпустить, 229
  платформа, 220
  события, 222
  типы, 223
анонимные функции JavaScript, 76
аргументы, создание собственных функций, 55
архитектура
  ASP.NET, 25
  ASP.NET AJAX, 34
аспекты поведения
  Sys.Preview.UI.ClickBehavior, 342
  Sys.Preview.UI.FloatingBehavior, 346
  использование, 341
атаки, отказ в обслуживании, 241
атрибуты
  EnableScriptLocalization, 201
  id, 320
  itemTemplateParentElementId, 369
  methodName, 320
  property, 317
  propertyKey, 317
  ScriptMethod, 122
  target, 317
  url, 320
  useGet, 320
  value, 317
аутентификация
  в приложениях
    вход и выход, 180
  подготовка приложения, 177
  служба поддержки, 490

Б

базовые методы, 297
базовые типы, 96
базовый тип, 99
базы данных
  AdventureWorks, 148
  установка испытательной базы данных, 32
безопасность
  JSON, 88
  закладки, 433
  объект XMLHttpRequest, 80
белый список, методика, 241
библиотека базовых классов, 25
библиотеки
  JavaScript, загрузка, 41
  Microsoft Ajax Library, 26, 27
  анимационных эффектов, 417
  базовых классов, 25
блоги, спам, 252
Брендан Эйч (Brendan Eich), 44
броузеры
  изменение языковых предпочтений, 204
  объект XMLHttpRequest, 69
  поддержка JavaScript, 47
  прием cookie, 129
  совместимость, 25

В

валидаторы, 323
  customValidator, 323
  rangeValidator, 323
  regexValidator, 323

requiredFieldValidator, 323
typeValidator, 323
собственные, 330
веб-сайты
Control Toolkit, пакет элементов управления
использование, 216
глобализация, 203
интернационализация, 203
локализация, 188
сопутствующие ресурсы, 192
сценариев, 188
профили
подготовка, 165
веб-службы
PHP, 459
внешние, 384
Amazon, 393
Google, 385
Yahoo!, 407
и Internet Explorer, 135
и Mozilla, 137, 141
методы страницы, 121
обмен данными со сложной структурой, 130
обработка ошибок, 116
управление состоянием сеанса, 125
веб-страницы
ASP.NET AJAX
клиенты, 111
объектно-ориентированные возможности JavaScript, 97
псевдонимы, 93
расширения к существующим объектам JavaScript, 96
JavaScript
встроенные объекты, 54
доступ к элементам, 61
доступ к элементам форм, 61
массивы, 49
методы, 47
обзор, 46
ООП, 57
переменные, 48
управляющие конструкции, 49
анимация
платформа, 220
архитектура Ajax, 23
жизненный цикл, 35
перетащить-и-отпустить, 229
программирование анимационных эффектов, 226
веб-части (ASP.NET), 448
внешние веб-службы, 384
Amazon, 393
Google, 385
Yahoo!, 407
возбуждение исключений, 117
всплывающие окна, отображение, 248
вспомогательные функции, 93
ASP.NET AJAX, 485
встраивание JavaScript, 46
встроенные анимационные эффекты
группировка, 427
использование, 417
перемещение элемента, 422, 425
встроенные объекты, JavaScript, 54
встроенные преобразователи, 306
вход, 180
вызов
веб-служб
в броузерах Mozilla, 141
веб-служб из Internet Explorer, 135
веб-службы Amazon, 396
веб-службы Google, 391
веб-службы на PHP, совместимой с ASP.NET AJAX, 461
удаленных веб-служб, 134
файла моста к веб-службе, 402
вызов методов, 319
выход, 180

Г

гибридные веб-приложения (mashups), 384
гиперссылки, 290
изображения, 290
глобализация, 203
группировка
анимационных эффектов, 427
группы
доступ к данным в профиле, 171
проверка, 335

Д

данные со сложной структурой, обмен, 130
даты
глобализация, 204
действия, 224

дерево, XML-документа, 81
Джесси Джеймс Гарретт (Jesse James Garrett), 68
динамическая свертка единственной панели, 247
добавление
анимации
перетащить-и-отпустить, 229
программно, 226
анимационных эффектов
платформа, 220
метода toString(), 105
обработчиков событий, 94, 95
файлов ресурсов, 194
функции автодополнения к элементу управления TextBox, 237
элементов управления из пакета Control Toolkit, 215
документация, Control Toolkit, 212
документы, объект XMLDocument, 81
доступ
OWA, 69
к атрибутам элементов, 64
к данным в профиле, 166
к методам JavaScript, 283
к методам базового класса, 105
к произвольным элементам, 64
к сгруппированным данным в профиле, 171
к элементам HTML, 286
к элементам страницы, 61
к элементам форм, 61
доступ к веб-службам из JavaScript, 134

Ж

жизненный цикл веб-страницы, 35

З

загрузка файлов JavaScript, 41
закладки
безопасность, 433
исправление недостатков, 432
Назад/Вперед кнопки, 435
пакет Futures, 440
запись
данных профиля, 167
сгруппированных данных в профиле, 171
запросы
HTTP, создание, 73
UpdatePanel, элемент управления, 146
область страницы, 146
инициация, 152
отмена, 160
значения, свойства readyState, 74
зоны веб-частей, 451

И

изображения, 288
гиперссылки, 290
импорт, 201
импорт компонента, 201
имя класса, параметр, 99
индивидуализация поведения перетащить-и-отпустить, 350
инкапсуляция, 57
инструменты
ASP.NET Web Application Administration Tool, 177
Control Toolkit, 27
отладки, 465
ASP.NET AJAX, 472
Firebug, 467
Visual Studio 2005, 469
Web Development Helper, 467
интегрированная среда разработки, 28
интернационализация, 203
интерпретация, данных в формате JSON, 87
интерфейс
тип параметра, 99
интерфейсы, 108
с вкладками, 254
инъекция SQL-кода, предотвращение, 242
исключения
возбуждение исключений, 117
обработка ошибок, веб-службы, 116
использование
ASP.NET AJAX вместе с PHP, 458
анимационных эффектов, 417
группировка, 427
аспектов поведения
Sys.Preview.UI.ClickBehavior, 342
перетащить-и-отпустить, 346
веб-службы Amazon, 393
веб-службы Yahoo!, 407
внешних веб-служб, 384

Google, 385
компонентов, 355
пакета Control Toolkit, 216
элементов управления, 282
использование аспектов поведения, 341
исправление недостатков
Назад/Вперед кнопки, 435
испытательная база данных, установка, 32
история
JavaScript, 44
появления объекта XMLHttpRequest, 69
источники данных, нестандартные, 377

К

календарь, присоединение к текстовому полю, 246
класс-обертка
для доступа к веб-службе Google, 388
для доступа к веб-службе Yahoo!, 410
для обращения к веб-службе Amazon, 394
классы, 199
AWSECommerceServiceWrapper, 394
LocDateControl, 199
PageMethods, 122
PageRequestManager, 159
PHPHelloWorldService, 459
ProfileService, 166
SOAPCall, 138
StringBuilder, 111
Sys.Debug, 473
Sys.Preview.Binding, 306
Sys.Preview.UI.Effects.NumberAnimation, 422
Sys.Preview.UI.Window, 283
TextBoxMaskDesigner, 260
TextBoxMaskExtender, 261
библиотеки, 199
наследование, 102
производные, 102
клиентские элементы управления, 281
использование, 282
обработка событий, 300
проверка данных, 323
клиенты, 34, 111
ASP.NET AJAX, программирование, 24
классы .NET, 111
программирование анимационных эффектов, 227
кнопки, 292
Load Vendors, 372
Назад/Вперед, 435
события, 300
команды
ASP.NET Configuration, 177
POST, 78
композитные анимационные эффекты, 427
компоненты, 34
ASP.NET AJAX, 355
контекст данных, 306
кэши
GAC, 31
на стороне клиента, 89

Л

логические операторы, 50
локализация, 188
сопутствующие ресурсы, 192
сценариев, 188
локализация текста, 199

М

маркированный список, 56
массивы
JavaScript, 49
метки
эффект плавного исчезновения, 419
методы
addCssClass(), 298
AddHistoryPoint(), 440
ASP.NET AJAX, 485
asyncInvoke(), 139
callService(), 136
DivideNumbers(), 118, 129
document.getElementById(), 93
DOM, 481
encode(), 139
get_accessKey(), 298
get_alternateText(), 288
get_argument(), 292
get_command(), 292
get_exceptionType(), 119
get_height(), 288
get_imageURL(), 289
get_message(), 119
GetPropertyValue(), 261

get_selectedValue(), 296
get_stackTrace(), 119
get_statusCode(), 119
get_text(), 287
get_timeOut(), 119
get_width(), 288
historyNavigate(), 436
initialize(), 263
inputBox(), 283
JavaScript, 47
    доступ, 283
load(), 168, 173
login(), 180
logout(), 180
messageBox(), 283
open(), 73
pageLoad(), 368
play(), 418
preventDefault(), 264
ProcessRequest(), 459
registerClass(), 99
removeCssClass(), 298
SaveTime(), 126
Search(), 390
SearchXslt(), 401
send(), 74
set_accessKey(), 298
set_alternateText(), 288
set_argument(), 292
set_command(), 292
set_direction(), 310
set_height(), 288
set_imageURL(), 289
set_OnClick(), 228
SetPropertyValue(), 261
set_text(), 287
set_transformerArgument(), 307
set_width(), 289
toggleCssClass(), 298
toString(), 60
    добавление, 105
Update(), 154
useService(), 136
window.setInterval(), 153
window.setTimeout(), 153
базового класса
    доступ, 105
базовые, 297
вызов, 319
объекта XMLHttpRequest, 479
страницы, 121
    элементов DOM, 96
методы базового класса
    доступ, 105
модальные диалоги JavaScript, 283
мост к веб-службе, файлы, 389, 395

Н

Назад/Вперед кнопки, 435
направление, привязки, 306, 310
наследование
    JavaScript, 59
    классов, 102
нестандартные типы данных, возврат, 377
нотация
    JSON, сериализация и десериализация данных, 130

О

обертка
    для доступа к веб-службе Google, 388
    для доступа к веб-службе Yahoo!, 410
    для обращения к веб-службе Amazon, 394
области страицы
    управление асинхронными запросами, 159
области страницы
    отображение экранной заставки в период ожидания, 157
область страницы, UpdatePanel, элемент управления, 146
обмен данными со сложной структурой, 130
обновление
    области страницы
        отображение экранной заставки, 157
    части страницы, 147
        через определенные интервалы времени, 152
        через определенные интервалы времени, программное, 154
обработка ошибок, веб-службы, 116
обработка событий, 316
    в клиентских элементах управления, 300
    с помощью сценариев, 46
обработчики событий
    добавление, 94, 95

удаление, 94
объектно-ориентированное программирование, 57
объекты
ASP.NET AJAX
клиенты, 111
объектно-ориентированные возможности JavaScript, 97
расширения к существующим объектам JavaScript, 96
DataRowColection, 361
document, 61
HistoryEventArgs, 440
ItemSearchRequest, 394
JSON, 86
XMLDocument, 81
XMLHttpRequest, 69, 479
встроенные, JavaScript, 54
перечисления, 113
расширения, 486
создание, 86
ООП (объектно-ориентированное программирование), 24, 57
в JavaScript, 57, 97
операторы
typeof, 71
логические, 50
сравнения, 51
отказ в обслуживании, атака, 241
отмена запроса, 160
отображение
всплывающего окна, 248
данных из нестандартного источника данных, 381
тегов, 449
экранной заставки в период ожидания, 157
отображение тегов веб-частей, 449

П

пакеты инструментальных средств
Control Toolkit, 211
установка, 212
пакеты, ASP.NET AJAX, 27
панели
Accordion, элемент управления, 234
свертка, 247
панели, перетащить-и-отпустить, 229
папки
AjaxControlExtender, 212
SampleWebSite, 212
параметры
baseArguments, 106
failedCallback, 167
instance, 106
loadCompletedCallback, 167
methodName, 106
propertyNames, 166
userContext, 167
базового типа, 99
имя класса, 99
интерфейс, 99
переменные
dateformat, 189
daynames, 189
JavaScript, 48
создание собственных функций, 55
перемещение
веб-частей, 451
элементов, 422, 425
перетащить-и-отпустить
анимация, 229
аспект поведения, 346
перечисления, 113
платформа создания анимационных эффектов, 220
поддержка
SmartTag, 148
объекта XMLHttpRequest, 72
профилей
подготовка веб-сайта, 165
службы аутентификации, 490
службы службы управления профилями, 490
сценарии, 491
позиционирование HTML-элементов, 236
поиск в каталоге Amazon, 407
поля
текстовые, 296
присоединение календаря, 246
форм, свойства, 63
постоянные ссылки, 446
предотвращение
инъекции SQL-кода, 242
возможности отправки формы, 338
кэширования на стороне клиента, 89
преобразователи, 306
встроенные, 306
собственные, 311

привязка данных, 305
к HTML-списку, 371
к HTML-таблице, 373
к элементу управления ListView, 360
направление, 306, 310
програмная, 306
разметка, 313
прием cookie, 129
приложения
Ajax, обзор, 24
ASP.NET, 25
JSON, 86
UpdatePanel, элемент управления, 146
область страницы, 146
аутентификация
вход, 180
выход, 180
подготовка, 177
доступ к данным в профиле, 166
доступ к сгруппированным данным в профиле, 171
локализация, 187
сопутствующие ресурсы, 192
сценариев, 188
методы страницы, 121
отладка
ASP.NET AJAX, 472
Firebug, 467
Visual Studio, 469
Web Development Helper, 467
инструменты, 465
примеры, ASP.NET AJAX, 35
присоединение календаря к текстовому полю, 246
проверка
групповая, 335
данных, 323
на соответствие регулярному выражению, 326
полей, обязательных к заполнению, 323, 324
принадлежности диапазону, 328
программная, 332
типа данных, 327
программирование
анимационных эффектов, 226
объекта XMLHttpRequest, 70
ООП, 24
JavaScript, 57
программная привязка данных, 306
программная проверка, 332
программное обновление части страницы, 154
программный код
интерфейсы, использование для структурирования, 108
производные классы, 102
просмотр
данных из нестандартного источника данных, 381
пространства имен, 98
Sys.Preview.UI, 281
профили
доступ к данным, 166
доступ к сгруппированным данным, 171
подготовка веб-сайта, 165
служба поддержки, 490
псевдонимы, ASP.NET AJAX, 93
путь к данным, 306

Р

разметка
для привязки данных, 313
расширения
Mozilla, 137, 139
класса Array, 486
класса Boolean, 486
класса Date, 487
класса Number, 487
класса Object, 487
класса String, 488
объектов, 486
расширители
JavaScript, 264
перетащить-и-отпустить, 348
регистрация
пакета компонентов, 216
регулярные выражения
проверка на соответствие, 326
режим
обработки события, 153
проектирования, UpdatePanel, 148
результаты веб-службы, преобразование с помощью XSLT, 398

С

свертка панели, 247
свойства, 306
AnimationChildren, 227

свойства
ASP.NET AJAX, 485
AsyncPostBackTrigger, 153
BackgroundCssClass, 250
CollapseControlID, 247
ControlID, 153
dataPath, 370
DOM, 482
DragHandleID, 229
_duration, 418
Enabled, 349
EventName, 153
ExpandControlID, 247
_fps, свойство, 418
HeaderText, 254
HorizontalSide, 236
OKControlID, 250
onreadystatechange, 74
PopupControlID, 250
PostBackTrigger, 153
property, 370
responseText, 75
responseXML, 75
ScriptManager, элемент управления, 489
_target, 419
TargetControlID, 221, 229, 247, 250, 349
Timer, элемент управления, 492
Triggers, 492
UpdatePanel, элемент управления, 491
UpdateProgress, элемент управления, 492
UserData, 165
userName, 165
VerticalSide, 236
visibilityMode, 330
изменение значений через xml-script, 317
объекта XMLHttpRequest, 480
полей формы, 63
профиля, объявление, 165
элемента <binding>, 315
серверы, 25, 34
ListView, элемент управления, 359
программирование анимационных эффектов, 226
элементы управления, 26
служба аутентификации
вход и выход, 180
службы
ASP.NET, 26
основные, 25
служба аутентификации, 490
служба управления профилями, 490
собственные валидаторы, 330
собственные преобразователи, 311
собственные функции, создание, 54
события
Tick, 153
для запуска анимации, 222
для кнопок, 300
для списков, 302
добавление обработчиков, 95
нажатия клавиш, 263
обработка, 316
в клиентских элементах управления, 300
с помощью сценариев, 46
совместимость, 25
создание
интерфейса с вкладками, 254
маркированного списка, 56
постоянных ссылок, 446
собственных функций, 54
сообщения об ошибках, 119
сопутствующие ресурсы, локализация, 192
состояние сеанса, управление, 125
спам, блоги, 252
списки
HTML
выбора, 373
привязка данных, 371
выбора, 293
маркированные, 56
события, 302
сравнения операторы, 51
ссылки
WSDL (Web Services Description Language), 135
элементы управления, гиперссылки, 290
стандарты, объект XMLHttpRequest, 70
страницы методы, 121
сценарии
компоненты и элементы управления, 25
локализация, 188
поддержка, 491
события, обработка, 46

Т

таблицы, HTML, 373
таймеры, 355
теги
  HTML, 314
  <invokeMethodAction>, 316
  <script>, 46
  <setPropertyAction>, 316
текстовые
  гиперссылки, 290
  поля, 296
типы
  анимационных эффектов, 223
  базовые, 96, 99
типы данных
  проверка, 327
точки останова, 469

У

удаление обработчиков событий, 94
удаленные веб-службы
  доступ, 134
управление
  асинхронными запросами, 159
  состоянием сеанса, 125
управляющие конструкции JavaScript, 49
установка
  ASP.NET AJAX, 28
  Control Toolkit, пакет элементов управления, 212
  Firebug, 467
  Futures CTP, 33
  SQL Server Express, 32
  интегрированной среды разработки, 28
  испытательной базы данных, 32
  культуры, 195
утилита для работы с исправлениями, 268

Ф

файл моста к веб-службе
  измененная версия, 403
  кэширование запросов, 415
файлы
  .asmx, 121
  AssemblyInfo.cs, 197
  Authentication.aspx, 181
  AWSECommerceServiceWrapper.cs, 394
  GoogleSearch.wsdl, 387
  .htc, HTML control (элемент управления HTML), 136
  MathService.asmx, 131
  PHPHelloWorldService.php, 459
  PreviewDragDrop.js, 346
  PreviewGlitz.js, 417
  PreviewScript.js, 346
  TextBoxMaskBehavior.js, 260, 262
  TextBoxMaskDesigner.cs, 260
  TextBoxMaskExtender.cs, 260
  Web.config, 165
  моста к веб-службе, 389, 395
    вызов, 402
  ресурсов, добавление, 194
флажки, 292
формы
  ввода, 252
  доступ к элементам, 61
  предотвращение отправки, 338
  свойства полей, 63
  спам, 252
функции
  $addHandler(), 94, 263
  $addHandlers(), 94
  applyPageState(), 433
  callComplete(), 413
  $clearHandlers(), 94
  doClick(), 181
  eval(), 88
  getCurrentPageState(), 433
  getXMLHTTP(), 72
  gotoStepClick(), 443
  JavaScript, анонимные, 76
  loginComplete(), 182
  loginError(), 182
  pageLoad(), 95, 181
  pageNavigate(), 442
  profileError(), 168, 174
  profileLoaded(), 168
  $removeHandler(), 94, 263
  saveProfile(), 169
  stateChanged(), 75
  вспомогательные, 93
    ASP.NET AJAX, 485
  собственные, создание, 54

Ц

циклы
  for, 52
  for…in, 53
  while, 53

Ч

части страницы, 147
черный список, методика, 241
чтение
  данных профиля, 166
  сгруппированных данных в профиле, 171

Ш

шаблоны
  Control Toolkit, 212
  ListView, элемент управления, 360
  <ProgressTemplate>, 157
  собственных элементов управления, 259
  содержимого, 147

Э

экземпляры
  объектов от одного и того же базового класса, 105
элементы
  <anonymousIdentification>, 165
  <authenticationService>, 179
  <binding>, 315
  <ContentTemplate>, 254
  , 119
  DOM, 96
  <FadeOut>, 221
  <floatingBehavior>, 346
  HTML, 314
    доступ, 286
    позиционирование, 236
  <iframe>, 434
  <itemTemplate>, 360
  <layoutTemplate>, 360
  <Panes>, 234
  <proxy>, 389
  , 119
  <system.web>, 165
  <system.web.extensions>, 166
  <tagMapping>, 449
  <template>, 370
  анимационный эффект перемещения, 422, 425
  произвольный доступ, 64
  страницы, доступ, 61
элементы упарвления
  Accordion, 234
  AlwaysVisibleControlExtender, 236
  AnimationExtender, 221, 226
  ASP.NET AJAX, 26
  AutoCompleteExtender, 238
  CalendarExtender, 246
  CollapsiblePanelExtender, 247
  ConfirmButton, 216, 217
  Control Toolkit, 216
  DragPanelExtender, 229
  GridView, 147
  Image, 288
  Label, 148, 287
  ListView, 359
  ModalPopupExtender, 250
  NoBot, 252
  Repeater, 361
  ScriptManager, 39, 489
  Select, 294
  TabContainer, 254
  TabPanel, 254
  Timer, свойства, 492
  UpdateHistory, 435
  UpdatePanel, 146, 436
    область страницы, 146
  UpdateProgress, 492
  WebPartManager, 450
  WebPartZone, 450
  базовые методы, 297
  гиперссылки, 290
  клиентские, 281
    использование, 282
    обработка событий, 300
    проверка данных, 323
  кнопки, 292
  создание собственных элементов управления, 258
  списки выбора, 293
  текстовые поля, 296
  флажки, 292
эффект плавного исчезновения, 419

Я

языки
    локализация сценариев, 188
языки программирования
    JavaScript
        встроенные объекты, 54
        доступ к элементам страницы, 61
        доступ к элементам форм, 61
        массивы, 49
        методы, 47
        обзор, 46
        ООП, 57
        переменные, 48
        управляющие конструкции, 49

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-122-3, название «Программирование в ASP.NET AJAX» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.