ВИКТОР ПЕТИН

bhv®

API Яндекс, Google и других популярных веб-сервисов

Готовые решения для вашего сайта

API Яндекс.Карт

ISPmanager API

API Google-сервисов

API Twitter

API Wikipedia

Готовые к размещению в сети сайты

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+ Материалы на www.bhv.ru

Виктор Петин

API Яндекс, Google и других популярных веб-сервисов

Готовые решения для вашего сайта

Санкт-Петербург
«БХВ-Петербург»
2012

УДК 681.3.06
ББК 32.973.26-018.2
П29

Петин В. А.

П29 API Яндекс, Google и других популярных веб-сервисов. Готовые решения для вашего сайта. — СПб.: БХВ-Петербург, 2012. — 480 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0743-1

Рассмотрены возможности, предоставляемые API Яндекс, Google, Twitter, ISPmanager, Wikipedia. Показано, как повысить функциональность и привлекательность веб-проектов, интегрировав в них возможности, предоставляемые API этих популярных веб-сервисов. Описано создание 4-х больших готовых к размещению в сети проектов (личного кабинета для сайта хостинговой компании, каталога предприятий, сайта учета заказов для фирмы такси, интерактивной карты местности региона), а также ряда небольших практических решений. Во всех случаях использованы современные технологии создания сайтов без перезагрузки страницы, в том числе подробно рассмотренные в книге фреймворки xajax и jQuery. Исходные коды описанных в книге и готовых к размещению в сети проектов можно скачать по ссылке: ftp://85.249.45.166/9785977507431.zip.

Для веб-разработчиков

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.08.11.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 38,7.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0743-1

Оглавление

Введение

Для кого и о чем эта книга

Предлагаемая книга ориентирована на читателей, владеющих языком разметки HTML, языком JavaScript и имеющих навыки программирования сайтов на языке PHP.

У владельцев сайтов всегда есть желание добавить функционала своим проектам или просто украсить дополнительными красивыми "фишками". Для этого далеко не обязательно программисту писать весь код, тем более, иногда это и невозможно — попробуйте с нуля написать поисковую систему или сервис, подобный Яндекс.Картам. Многие сервисы специально предоставляют для доступа к возможностям своих сервисов API (Application Programming Interface, интерфейс прикладного программирования), что позволяет с помощью небольшого кода использовать функционал этих сервисов на своих сайтах.

Вместе с вами мы создадим несколько больших проектов, широко использующих API популярных веб-сервисов. Это действующие проекты, которые уже существуют в сети. Это сайт хостинговой компании, каталог предприятий, сайт учета заказов для фирмы такси, сайт — интерактивная карта региона с отображением различных объектов. Вы можете их использовать целиком или на их основе создать собственные проекты. Кроме этого, в книге представлено большое количество небольших полезных примеров, которые вы можете встроить в свои проекты.

Все готовые проекты, а также отдельные примеры находятся на прилагаемом к книге компакт-диске.

Структура книги

Книга содержит введение, семь глав, заключение и приложение.

В *главе 1* рассмотрены возможности использования функционала популярных веб-сервисов на сторонних сайтах с помощью предоставляемых сервисами API, а также популярные библиотеки xAjax и jQuery, применяемые для связи с API популярных веб-сервисов.

Глава 2 посвящена рассмотрению основных веб-сервисов главной российской интернет-компании — "Яндекс" и API данных сервисов. В главе представлены примеры внедрения возможностей рассмотренных сервисов с помощью предоставляемых этими сервисами API.

В *главе 3* рассматриваются возможности, предоставляемые самым популярным и востребованным сервисом Яндекса — Яндекс.Картами. Для лучшего понимания возможностей сервиса представлены примеры внедрения возможностей данного сервиса на сторонние сайты.

В *главе 4* закрепляем материал предыдущей главы на практических примерах создания больших проектов, использующих API Яндекс.Карт — каталога предприятий, сайта учета заказов для компании такси, создание интерактивной карты местности с несколькими слоями пользовательских карт для отображения с помощью меток различных объектов и информации о них (адреса, телефоны, фото, видео), построение полноценной администраторской панели для проекта создания интерактивной карты местности.

В *главе 5* рассматривается API ISPmanager — полнофункциональной панели управления хостингом, самой популярной среди российских провайдеров. Для демонстрации возможностей API создадим сайт-тренажер демонстрации возможностей API.

В *главе 6* создадим личный кабинет для сайта хостинговой компании. В проекте будем использовать API ISPmanager.

В *главе 7* рассмотрим API некоторых веб-сервисов компании Google, а также интернет-энциклопедии Wikipedia, самого популярного в мире сервиса микроблогов Twitter и API Loginza, предоставляющего возможность внедрения OpenID-авторизации на своем сайте. Рассмотрение API производится на примерах, которые вы сможете использовать в своих проектах.

Благодарности

Хочу поблагодарить родных и близких, которые с пониманием относились к потраченному на книгу (за счет общения с ними) времени.

Большая благодарность издательству "БХВ-Петербург", где поверили в необходимость данной книги для читателя, и всем сотрудникам, которые помогали мне в создании книги.

Хочу поблагодарить и всех читателей, которые купят эту книгу, я делал все, чтобы она была интересной и полезной. Надеюсь, что так оно и есть.

Если возникнут вопросы или пожелания по данной книге, то вы всегда сможете связаться с издательством (**mail@bhv.ru**) или со мной по электронной почте **victoruni@km.ru, kmvnews@bk.ru** или оставить сообщение в блоге **http://goodtovars.ru/blog**, где рассматриваются вопросы создания сайтов без перезагрузки страницы.

ГЛАВА 1

API веб-сервисов и технологии использования

1.1. Использование возможностей общедоступных Web API в асинхронных приложениях

Создание веб-сайтов требует от разработчиков немало опыта и навыков. При этом они должны постоянно следить за новшествами и быть в курсе современных технологий. Развитие технологий программирования привело к тому, что многие популярные интернет-сервисы принесли на просторы сети такую функциональность, о которой еще несколько лет назад никто не мог и мечтать. Сделан громадный шаг от простых страниц для отображения статической информации до сервисов, не уступающих своим десктопным собратьям. Кто мог несколько лет назад предположить, что мы сможем путешествовать по миру с Google Street View или Яндексом? Панорамы, с помощью которых можно бродить по улицам любого города мира, сидя дома в уютном кресле! Или пользоваться графическим редактором прямо на страницах сайта. Как хочется разместить такие же сервисы на своих сайтах для привлечения клиентов. И сейчас это становится реальностью! Очень многие популярные Web-сервисы дают возможность использования своего функционала предоставлением API своих сервисов для публичного пользования.

Что же такое API? API — интерфейс прикладного программирования (или интерфейс программирования приложений), т. е. набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) для использования во внешних программных продуктах. API определяет функциональность, которую предоставляет программа (модуль, библиотека), при этом API позволяет абстрагироваться от того, как именно эта функциональность реализована. Если программу (модуль, библиотеку) рассматривать как черный ящик, то API — это множество "ручек", которые доступны пользователю данного ящика, которые он может вертеть и дергать.

Вообще, API — довольно широкое понятие. Мы же здесь будем рассматривать только API для сайтов — Web API. Web API (или API веб-приложений) — это интерфейс программирования приложений, использующих данные определенных веб-

сервисов. Web API представляют собой инструмент, позволяющий получать данные от определенного стороннего сервиса, обрабатывать их в собственном приложении и при необходимости передавать какие-либо данные или команды обратно в сторонний сервис. Web API — это внешние приложения, которые могут легко встраиваться в любой сайт. Обычно используют HTML, XML, JavaScript и Flash в различных комбинациях.

Количество сервисов, предоставляющих публичный доступ к своему API, очень велико. Это и платежные системы (например, PayPal, WebMoney), торговые площадки (Amazon, "Молоток"), социальные сети (Facebook, Twitter, "ВКонтакте") и др.

Использование API популярных сайтов позволяет многократно увеличить полезность собственного ресурса, создать интересный и неповторимый контент, удовлетворить требования всевозможных бизнес-задач — утроить их не составляет труда. Использование API очень выгодно — они потребляют ресурсы своего сервера, а не вашего, при этом работают на вашем сайте. Это очень удобно: часто для установки API на сайт, как правило, достаточно встроить предложенный производителем код в нужное место на странице сайта, хотя можно создавать и собственные неповторимые проекты.

Использование API весьма рекомендуется владельцам небольших сайтов, а также сайтов с небольшой функциональностью — блок мировых новостей, красивые часы, котировки валют или прогноз погоды (смотря что подойдет вашему сайту) и многое другое может расширить функциональность сайта, украсить его, а значит, сделать более привлекательным для посетителей. Не следует пренебрегать возможностями API и крупным проектам. Например, полезными будут встроенная проверка орфографии, переводчик, поисковик, интерактивная карта и многое другое.

Для веб-программистов изучение Web API поможет не просто повысить свой профессиональный уровень знаний в веб-разработке, но и получить бесценные знания в области интеграционных технологий и стать широко востребованным специалистом.

Применение API популярных сервисов и внедрение их возможностей в своих проектах мы будем рассматривать в связке с современными технологиями сайтостроения, предполагающего применение технологии Ajax. Поэтому далее мы рассмотрим библиотеки xAjax и jQuery, позволяющие создавать современные сайты без перезагрузки страницы.

1.2. Библиотека xAjax

xAjax — это библиотека Open Source классов PHP, которая позволяет легко создавать мощные веб-ориентированные Ajax-приложения, использующие HTML, CSS, JavaScript и PHP. Приложения, разработанные при помощи библиотеки xAjax, могут асинхронно вызывать расположенные на сервере PHP-функции и обновлять содержание без перезагрузки страницы. xAjax предоставляет простую реализацию технологии Ajax, а начиная с версии 0.5, еще и PHP-инструменты для формирования HTML-форм и документов. В отличие от многих других подобных библиотек,

xAjax позволяет разрабатывать Ajax-приложения, не требуя от разработчика знания JavaScript. Библиотека xAjax распространяется по лицензии GNU Lesser General Public License (LGPL) и может быть использована для написания платного программного обеспечения. Сайт проекта — **http://xajaxproject.org**. На момент написания книги доступна версия 0.6.

1.2.1. Как работает xAjax

Библиотека xAjax создает функции JavaScript, являющиеся оболочкой для PHP-функций, которые вы можете вызывать с сервера из вашего приложения. Когда вызывается функция JavaScript, являющаяся оболочкой для функции PHP, то она использует объект `XMLHttpRequest` для асинхронного соединения с xAjax-объектом на сервере, который вызывает соответствующую функцию PHP. После завершения этого действия возвращается xAjax XML-ответ от вызванной PHP-функции. Возвращенный XML-код содержит инструкции и данные, которые будут проанализированы специальными функциями JavaScript-части xAjax и использованы для обновления содержания вашего приложения.

1.2.2. Возможности xAjax

Библиотека xAjax предлагает следующие возможности, которые вместе делают ее уникальным и мощным инструментом.

- xAjax — уникальная библиотека на JavaScript, которая может анализировать возращенный XML-код и автоматически его обрабатывать согласно инструкциям, находящимся в этом ответе. Так как xAjax обрабатывает все это, то вам не нужно писать отдельные функции на JavaScript для того, чтобы обрабатывать возвращенный XML-код.
- xAjax — это объект, ориентированный на создание отношений между программным кодом и данными для хранения xAjax-кода отдельно от другого программного кода. Так как это объектно-ориентированный код, то вы всегда можете добавлять свои функции в класс `xajaxResponse`, используя метод `script()`.
- xAjax работает в браузерах Firefox, Mozilla, Internet Explorer и Safari. Помимо обновления значений элементов (имеется в виду DOM) и innerHTML, xAjax также может быть использован для обновления стилей, CSS-классов, значений флажков и раскрывающихся списков или каких-либо других свойств элемента.
- xAjax может использовать одномерные и многомерные массивы, а также ассоциативные массивы из JavaScript в PHP как параметры ваших xAjax-функций. В дополнение, если вы вводите JavaScript-объект в xAjax-функцию, функция PHP будет получать ассоциативный массив, определяющий свойства этого объекта.
- xAjax предоставляет легкую асинхронную обработку форм. Используя JavaScript-метод `xajax.getFormValues()`, в форме вы можете легко передать массив данных, как параметры для асинхронной xAjax-функции:

```
xajax_processForm(xajax.getFormValues('formId').
```

Используя xAjax, вы можете динамически подгружать дополнительный JavaScript-код для вашего приложения для того, чтобы при его исполнении менялись свойства элемента DOM.

- xAjax автоматически сравнивает данные, возвращенные из PHP-функций, с текущими значениями свойства элемента, который вы хотите изменить. Свойство изменяется только в том случае, если это изменение актуально на текущий момент. Это позволяет устранить мерцание, которое происходит, если элемент обновляется через определенные промежутки времени. Каждая функция регистрируется для того, чтобы быть доступной через xAjax, который имеет различные типы запросов. Все функции по умолчанию используют метод передачи POST, за малым исключением — метод GET. Это сделано для большей безопасности запросов.
- Если не определен запрашиваемый URI, xAjax пытается автоматически определить запрашиваемый URL скрипта. Алгоритм автоопределения xAjax достаточно универсален, так что он будет работать как на безопасном протоколе https://, так и на http:// и на нестандартных портах.
- xAjax перекодирует все свои запросы и ответы в кодировку UTF-8. Таким образом, он поддерживает большой спектр различных знаков и языков.
- xAjax был протестирован на различных языках в кодировке Unicode, включая испанский, русский, арабский. Почти весь JavaScript-код динамически подгружается через JavaScript-расширения.
- xAjax может быть использован в шаблонном движке Smarty. Для создания переменной в Smarty должен быть следующий код:

```
$smarty->assign('xajax_JavaScript', $xajax->getJavascript());
```

При использовании xAjax подставляйте в заголовок тег `{$xajax_JavaScript}`.

1.2.3. Подключение xAjax

xAjax — это PHP-библиотека, которая отличается тем, что позволяет исполнять JavaScript-код на основе PHP-кода. Весь процесс состоит из двух PHP-классов и обработчика XML на JavaScript. В общем, на PHP сначала инициализируется объект и объявляются функции, которые будут отвечать на Ajax-запрос. В этих функциях необходимо использовать объект, который и будет генерировать XML-ответ.

Для скачивания библиотеки xAjax следует обратиться по адресу **http://xajax-project.org/en/download/** и нажать на ссылку **Xajax 0.6 beta1**. Начнется скачивание архива с библиотекой (рис. 1.1).

Далее надо распаковать архив в корневой каталог сайта. Теперь покажем, какой код необходимо внести в файл для подключения xAjax:

```
<html>
<head>
<?php
```

```
  // подключение библиотеки
  require_once ("xajax_core/xajax.inc.php");
  // создать новый xAjax-объект
  $xajax = new xajax();
  // регистрация функций
  $xajax->register(XAJAX_FUNCTION,"Function1");
  // разрешение обрабатывать асинхронные xAjax-запросы
  $xajax->processRequest();
  function Function1()
  {
    $objResponse = new xajaxResponse();
    // код
    return $objResponse;
  }
?>
<?php
  echo $xajax->getJavascript("");
?>
</head>
<body>

</body>
</html>
```

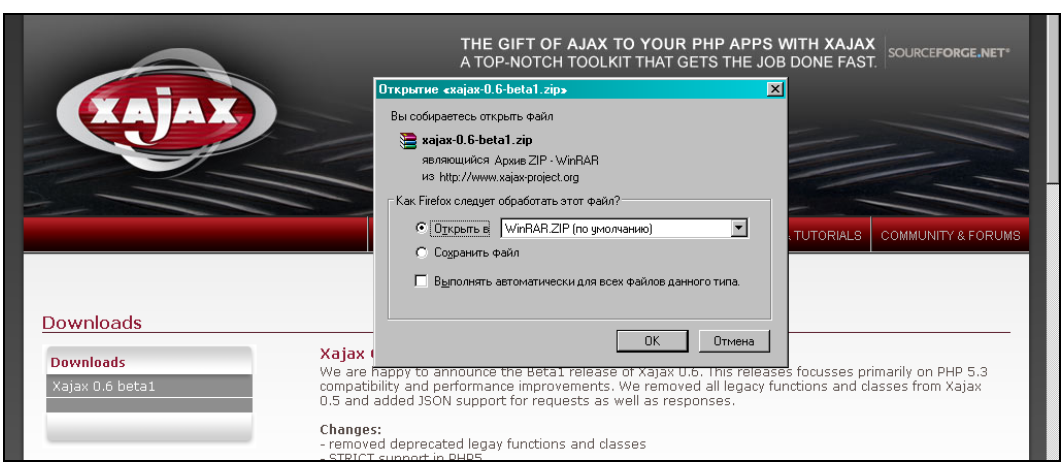


Рис. 1.1. Страница скачивания библиотеки xAjax

Ясно видно, что xAjax-объект — основной, в него регистрируются функции обработки, а `xajaxResponse` — вспомогательный, он генерирует XML-код, который потом распознается на уровне JavaScript и выполняет соответствующие действия. При вызове xAjax-функций из JavaScript-кода добавляется префикс `xajax_`:

```
<a 'href=JavaScript:void();' onclick='xajax_Function1();'></a>
```

1.2.4. Методы объекта *xajaxResponse*

Объект `xajaxResponse` имеет следующие методы:

- `assign();`
- `append();`
- `prepend();`
- `replace();`
- `remove();`
- `create();`
- `insert();`
- `insertafter();`
- `clear();`
- `createInput();`
- `insertInput();`
- `insertInputAfter();`
- `removeHandler();`
- `includeScript();`
- `script();`
- `addEvent();`
- `call();`
- `alert();`
- `redirect()`.

Рассмотрим подробнее эти методы.

> ***Примечание***
>
> На компакт-диске, прилагаемом к данной книге, в каталоге glava_01\1 представлен Web-сайт — тренажер для изучения этих методов с просмотром результатов в визуальном режиме. Подробно работа тренажера рассмотрена в *разд. 1.2.5*.

Метод *assign()*

Изменяет параметры HTML-элементов, будь то `innerHTML`, `style` и т. п.

```
$objResponse->assign($sTarget, $sAttribute, $sData)
```

Заменяет значение `$sAttribute` элемента `$sTarget` на `$sData`.

Примеры:

```
// установить новое содержимое элемента с id=div1 — <b>New</b>
$objResponse->assign("div1","innerHtml","<b>New</b>");
// установить красный цвет текста в элементе с id=div1
$objResponse->assign("div1","style.color","red");
// скрыть элемент с id=div1
$objResponse->assign("div1","style.display","block");
```

Метод *append()*

Изменяет параметры элементов, добавляя данные в конец.

```
$objResponse->append($sTarget, $sAttribute, $sData)
```

Добавляет `$sData` в конец значения атрибута `$sAttribute` элемента `$sTarget`.

Примеры:

```
// добавить в конец содержимого элемента с id=div1 код HTML — <U>element</U>
$objResponse->append("div1","innerHtml","<U>element</U>");
// если HTML-код был <b>New</b>, то теперь будет <b>New</b><U>element</U>
```

Метод *prepend()*

Добавляет данные в начало.

```
$objResponse->prepend($sTarget, $sAttribute, $sData)
```

Добавляет `$sData` в начало значения атрибута `$sAttribute` элемента `$sTarget`.

Примеры:

```
// добавить в начало содержимого элемента с id=div1 код HTML — <U>element</U>
$objResponse->append("div1","innerHtml","<U>element</U>");
// если HTML-код был <b>New</b>, то теперь будет <U>element</U><b>New</b>
```

Метод *replace()*

Заменяет в элементе одни значения другими.

```
$objResponse->replace($sTarget, $sAttribute, $sSearch, $sData)
```

Заменяет найденное по маске `$sSearch` значение атрибута `$sAttribute` элемента `$sTarget` на `$sData`.

Примеры:

```
// заменить в содержимом элемента с id=div1 код HTML — 1 на код 2<br>
$objResponse->replace("div1","innerHtml","1","2<br>");
// если HTML-код был <b>1234512345</b>, то теперь будет
// <b>2<br>23452<br>2345</b>
// напоминает PHP-функцию str_replace
```

Метод *remove()*

Удаляет элемент.

```
$objResponse->remove($sTarget)
```

Уничтожает элемент `$sTarget`.

Примеры:

```
// удалить элемент с id=div1
$objResponse->remove("div1");
```

Метод *create()*

Создает элемент.

```
$objResponse->create($sParent, $sTag, $sId, $sType = "")
```

Создает элемент `$sTag` с идентификатором `$sId`, как потомка от `$sParent` и типом `$sType`.

Примеры:

```
// создать элемент span с id=span1 в элементе с id=div1
$objResponse->create("div1","span","span1");
// если на странице был элемент div с id=div1: <div id=div1></div>,
// то теперь будет <div id=div1><span id=span1></span></div>
```

Метод *insert()*

Вставляет новый элемент.

```
$objResponse->insert($sBefore, $sTag, $sId)
```

Вставляет элемент `$sTag` с идентификатором `$sId` перед элементом `$sBefore`.

Примеры:

```
// создать элемент span с id=span1 перед элементом с id=div1
$objResponse->create("div2","span","span1");
// если на странице был div-элемент с id=div1,
// а внутри него div-элемент с id=div2
// <div id=div1><div id=div2></div></div>, то теперь будет
// <div id=div1><span id=span1></span><div id=div2></div></div>
```

Метод *insertAfter()*

Добавляет элемент после заданного элемента.

```
$objResponse->insertAfter($sAfter, $sTag, $sId)
```

Вставляет элемент `$sTag` с идентификатором `$sId` после элемента `$sAfter`.

Примеры:

```
// создать span-элемент с id=span1 после элемента с id=div1
$objResponse->create("div2","span","span1");
// если на странице был div-элемент с id=div1,
// а внутри него div-элемент с id=div2
// <div id=div1><div id=div2></div></div>, то теперь будет
// <div id=div1><div id=div2><span id=span1></span></div></div>
```

Метод *clear()*

Очищает содержимое элемента.

```
$objResponse->clear($sTarget, $sAttribute)
```

Очищает значение атрибута `$sAttribute` элемента `$sTarget`.

Примеры:

```
// очищает содержимое элемента с id=div1
$objResponse->clear("div2","innerHTML");
// очищает содержимое свойства color элемента с id=div1
$objResponse->clear("div2","color");
```

Метод *createInput()*

Создает элемент формы.

```
$objResponse->createInput($sParent, $sType, $sName, $sId)
```

Создает элемент HTML-формы как дочерний элемент от элемента `$sParent` с типом `$sType`, именем `$sName` и идентификатором, равным `$sId`.

Примеры:

```
// создает в форме с id=form1 элемент input
// с id=input2 и name=input2
$objResponse->createInput("form1","input","input2","input2");
```

Метод *insertInput()*

Создает элемент формы.

```
$objResponse->insertInput($sBefore, $sType, $sName, $sId)
```

Создает элемент HTML-формы перед элементом `$sBefore` с типом `$sType`, именем `$sName` и идентификатором, равным `$sId`.

Примеры:

```
// создает в форме input-элемент с id=input2 и name=input2
// перед элементом с id=input1
$objResponse->insertInput("input1","input","input2","input2");
```

Метод *insertInputAfter()*

Создает элемент формы.

```
$objResponse->insertInputAfter($sAfter, $sType, $sName, $sId)
```

Создает элемент HTML-формы после элемента `$sAfter` с типом `$sType`, именем `$sName` и идентификатором, равным `$sId`. Примеры:

```
// создает в форме input-элемент с id=input2 и name=input2
// после элемента с id=input1
$objResponse->insertInputAfter("input1","input","input2","input2");
```

Метод *removeHandler()*

Удаляет функцию обработки событий.

```
$objResponse->removeHandler($sTarget, $sEvent, $sHandler)
```

Удаляет js-функцию `$sHandler` для обработки события `$sEvent` элемента `$sTarget`.

Примеры:

```
// удаляет функцию fun_over для события onmouseover элемента с id=div1
$objResponse->removeHandler("div1","onmouseover","fun_over");
```

Метод *includeScript()*

Подключает внешний js-файл.

```
$objResponse->includeScript($sPath)
```

Подключает внешний js-файл, путь к которому задан в параметре `$sPath`.

Примеры:

```
// Подключаем внешний js-файл, чтобы первая страница
// загружалась не слишком долго.
// Внешний js-файл загружаем не в начале, а только тогда,
// когда он будет использоваться
$objResponse->includeScript("js/jquery.fancybox-1.3.0.js");
```

Метод *script()*

Добавляет прописанную вручную js-обработку.

```
$objResponse->script($sScript)
```

Выполняет JavaScript-код, содержащийся в `$sScript`.

Примеры:

```
// выполнить js-код — выдать alert-окно с сообщением "Сообщение!!!"
$objResponse->script("alert('Сообщение!!!')");
// элемент с id=d1 в зону видимости страницы
$objResponse->script("document.getElementById('d1').scrollIntoView();");
// установить красный цвет текста в элементе с id=div1
// (аналогично $objResponse->assign("d1","style.color","red");)
$objResponse->script("document.getElementById('d1').style.color='red';");
```

Метод *addEvent()*

Создает новое событие.

```
$objResponse->event($sTarget, $sEvent, $sScript)
```

Создает событие `$sEvent`, привязывая его к элементу `$sTarget` и связывая с ним код `$sScript`.

Примеры:

```
// создаем новые события для элемента с id=div1:
//   onmouseover — при наведении мыши на объект цвет элемента красный;
//   onmouseout — при наведении мыши на объект цвет элемента синий
$objResponse->event("div1","onmouseover","this.style.color='red'");
$objResponse->event("div1","onmouseout","this.style.color='blue'");
```

Метод *call()*

Вызывает заданную js-функцию с указанными параметрами.

```
$objResponse->call($sFunc, $args, ...)
```

Вызывает js-функцию `$sFunc` с заданными параметрами `$args`.

Примеры:

```
// вызывает js-функцию my_function("div1",4,10) с тремя аргументами
$objResponse->call("my_function", "div1",4, 10))
```

Метод *alert()*

Создает окно-оповещение.

```
$objResponse->alert($sMsg)
```

Отображает предупреждение JavaScript с текстом, заданным параметром `$sMsg`.

Примеры:

```
// выдать alert-окно с сообщением "Сообщение!!!"
$objResponse->script("alert('Сообщение!!!');");
```

Метод *redirect()*

Осуществляет перенаправление на другую страницу, возможно, через некоторое время.

```
$objResponse->redirect($sURL)
```

Перенаправляет на страницу `$sURL`.

Примеры:

```
// перенаправляет на другую страницу
$objResponse->redirect("http://www.site.ru/register.php");
```

1.2.5. Сайт — тренировочный стенд для изучения xAjax

Для лучшего усвоения материала я создал небольшое веб-приложение для изучения Response-методов библиотеки xAjax. Сайт позволяет изучать асинхронно вызываемые функции xAjax (Response-методы): в режиме тренировочного стенда можно выбрать параметры и сразу же увидеть результат на странице. Сайт сделан с использованием библиотеки xAjax без перезагрузки страницы. Сайт находится на компакт-диске в каталоге glava_01\1. Данный пример доступен в Интернете по адресу **http://examples-api.bazakatalogov.ru/glava_01/1**. Вид главной страницы представлен на рис. 1.2.

Для удобства восприятия методы `xajaxResponse` объекта разбиты на 3 группы:

1. Работа с элементами:
   - метод `assign()` изменяет параметры HTML-элементов, будь то `innerHTML`, `style` и т. п.;
   - метод `append()` также изменяет параметры элементов, добавляя данные в конец;
   - метод `prepend()` добавляет данные в начало;
   - метод `replace()`, как вы догадались, заменяет в элементе одни значения другими, как функция `str_replace()`;
   - метод `remove()` удаляет элемент;
   - метод `create()` создает элемент;

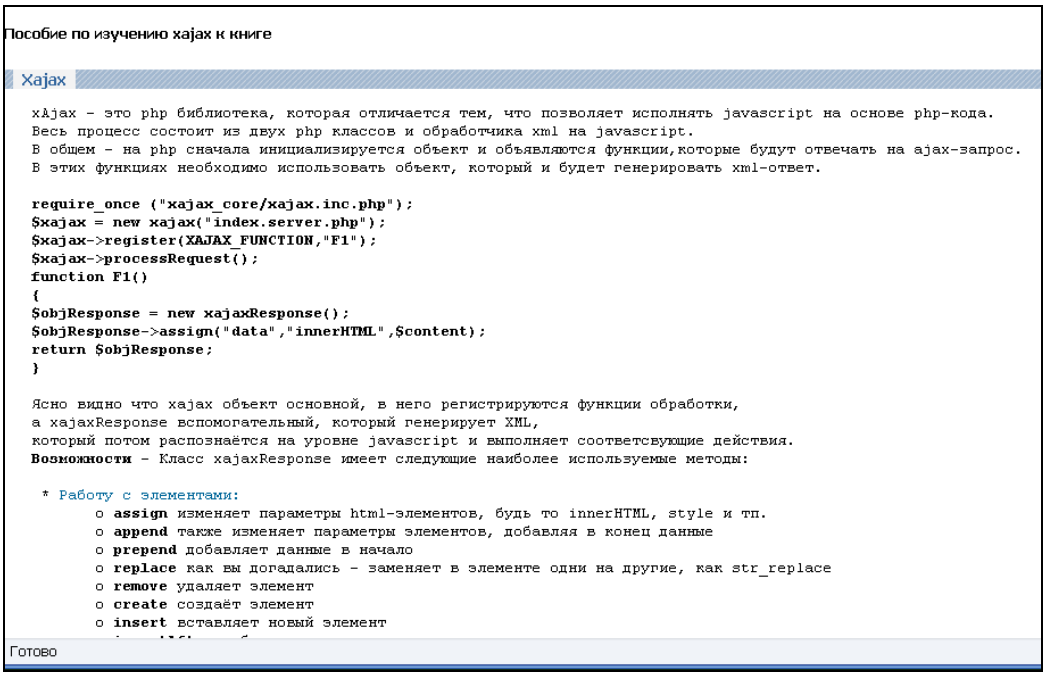


Рис. 1.2. Страница сайта — тренировочного стенда

- метод `insert()` вставляет новый элемент;
- метод `insertafter()` добавляет элемент после заданного элемента;
- метод `clear()` очищает содержимое элемента.

2. Работа с `input`-полями:
   - метод `createInput()`;
   - метод `insertInput()`;
   - метод `insertInputAfter()`.
3. Особые процессы:
   - метод `removeHandler()`;
   - метод `includeScript()` подключает внешний js-файл;
   - метод `script()` добавляет прописанные вручную js-обработки;
   - метод `addEvent()` создает новое событие;
   - метод `call()` вызывает заданную js-функцию с указанными параметрами;
   - метод `alert()` создает сообщение;
   - метод `redirect()` осуществляет перенаправление на другую страницу, возможно, через некоторое время.

Как работать с сайтом? При нажатии на ссылку **Работа с элементами** видим страницу, изображенную на рис. 1.3.

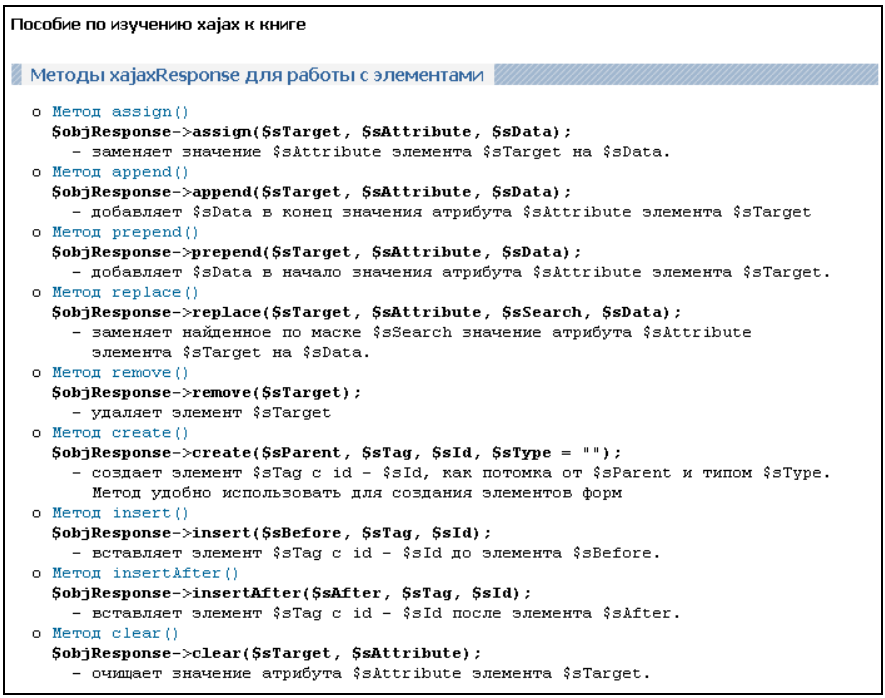


Рис. 1.3. Выбор методов для работы с элементами

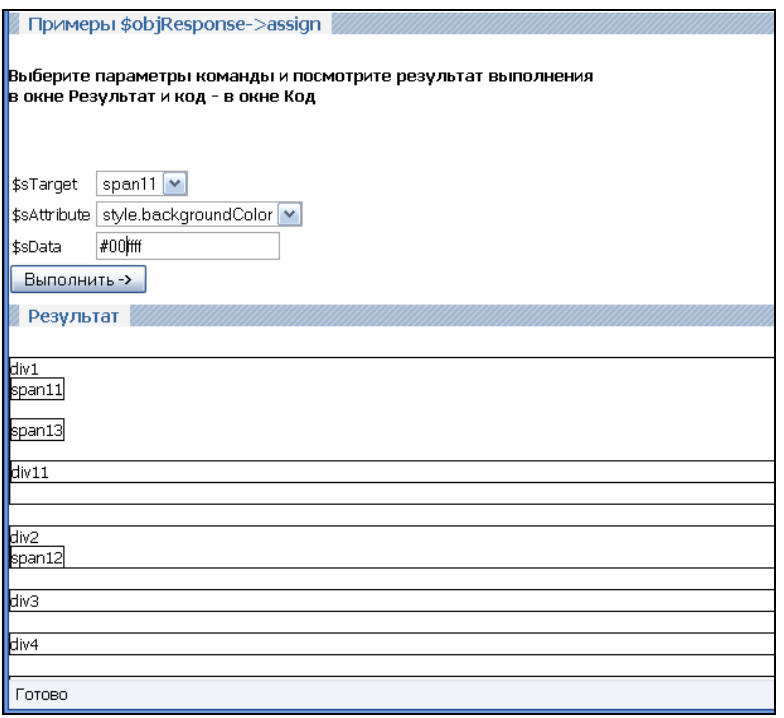


Рис. 1.4. Выбор параметров функции `assign()`

Далее выбираем какой-нибудь метод, например `assign()`, и попадаем на страницу, изображенную на рис. 1.4. Выбираем в форме значения атрибутов и нажимаем кнопку **Выполнить**.

Результат работы функции видим на рис. 1.5. Изменился фон элемента `span13`. В блоке **Код** видим код отправленной команды.

Теперь вернемся по ссылке **Назад** и выберем метод `remove()` (рис. 1.6). В раскрывающемся списке выберем элемент, например `span13`.

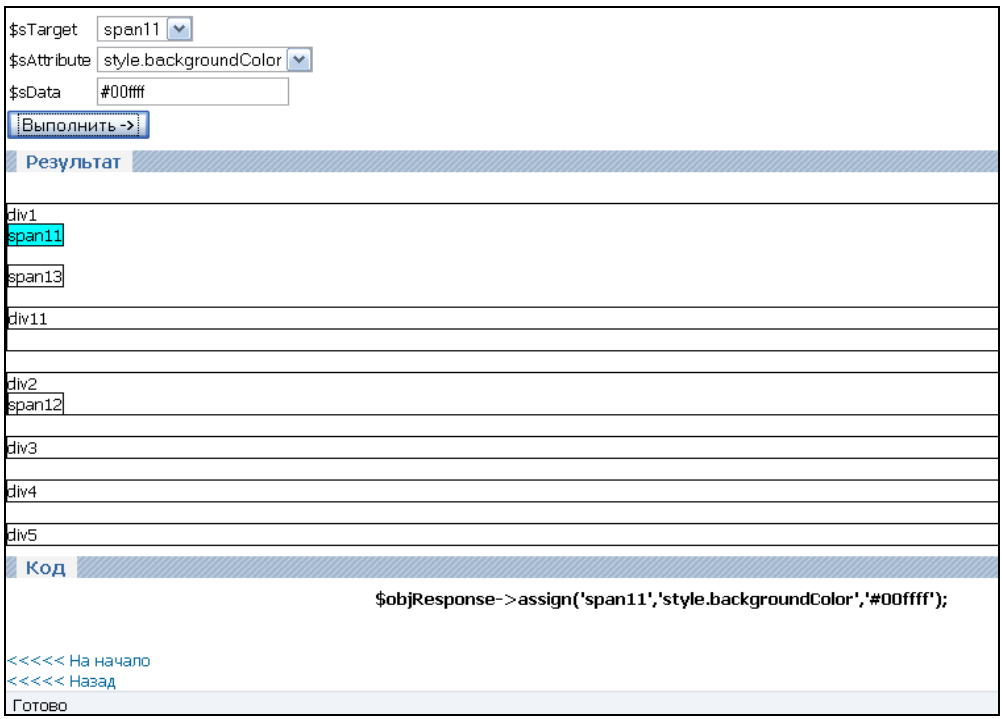


Рис. 1.5. Вид страницы после выполнения функции `assign()`

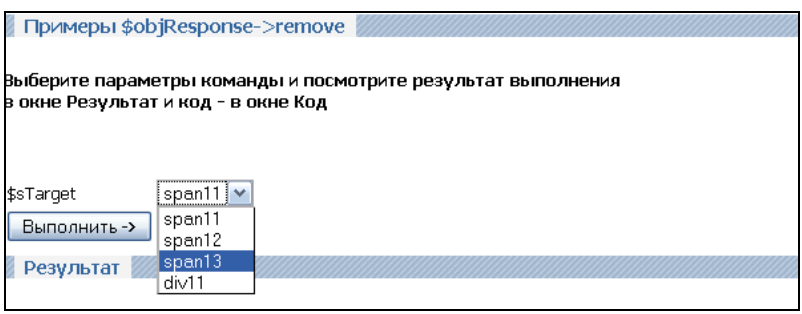


Рис. 1.6. Выбор параметров функции `remove()`

Нажимаем на кнопку **Выполнить** и видим результат выполнения функции (рис. 1.7). Элемент `span13` удален.

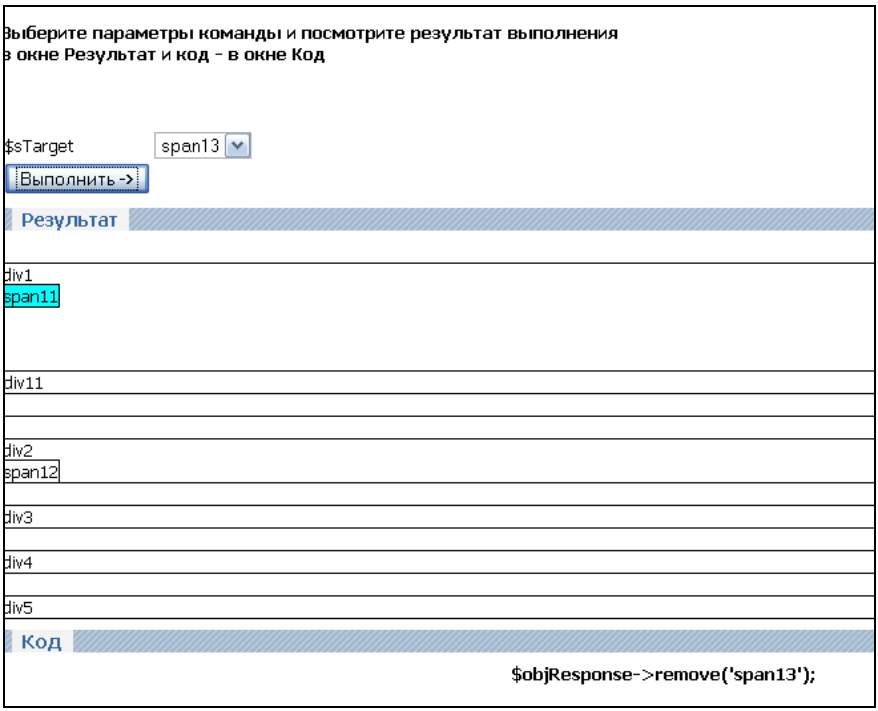


Рис. 1.7. Вид страницы после выполнения функции `remove()`

Думаю, принцип работы со стендом понятен. Надеюсь, он поможет вам лучше изучить работу асинхронных функций библиотеки xAjax.

> ***Примечание***
>
> При создании или удалении блоков вся информация сохраняется в базе данных, поэтому вид блока **Результат** может отличаться от вышеприведенного. С другой стороны, стенд не позволяет удалить все блоки, и блок **Результат** никогда не будет пустым.

1.2.6. Глобальные переменные xAjax

Знание xAjax состоит не только в способности написать веб-страницу, которая будет работать без перезагрузки, но и в умении настраивать и изменять много параметров, которые могут кардинально изменить работу xAjax.

Глобальные константы

Глобальные константы объекта xAjax:

- `XAJAX_DEFAULT_CHAR_ENCODING string` (по умолчанию `"utf-8"`) используется как в классах `xAjax`, так и `xajaxResponse`. Вы можете сами задать значение этой константы;
- `XAJAX_GET int` (по умолчанию `0`) показывает, что используется метод GET HTTP-запроса, применяемый в xAjax;

- `XAJAX_POST int` (по умолчанию `1`) — метод запроса, используемый в POST-методе HTTP-запроса xAjax.

Методы объекта *xajax*

Объект `xajax` имеет следующие методы.

- `xajax (string $sRequestURL="", string $sWrapperPrefix="xajax_", string $sEncoding=XAJAX_DEFAULT_CHAR_ENCODING, boolean $bDebug=false)` — конструктор. Вы можете сразу установить нужные вам значения в конструкторе или же позже посредством методов.

  Параметры:

  - `$sRequestURL` — тип `string`, по умолчанию текущий URL браузера;
  - `$sWrapperPrefix` — тип `string`, по умолчанию `"xajax_"`;
  - `$sEncoding` — тип `string`, по умолчанию равно глобальной константе;
  - `$bDebug Mode` — тип `boolean`, по умолчанию `false`.

- `setRequestURI (string $sRequestURL)` — устанавливает URI, к которому будет сделан запрос.

  Параметр `$sRequestURL` — это URL (может быть абсолютным или относительным) в PHP-коде скрипта, который должен быть запрошен объектом `xajax`.

  Применение:

  ```
  $xajax->setRequestURL("http://www.site.ru");
  ```

- `debugOn()` — устанавливает вывод отладочных сообщений (JavaScript alerts) для запроса xAjax.
- `debugOff()` — скрывает вывод отладочных сообщений для запроса xAjax.
- `statusMessagesOn()` — включает вывод сообщений в строку состояния браузера xAjax.
- `statusMessagesOff()` — отключает вывод сообщений в строку состояния браузера xAjax (установлено по умолчанию).
- `waitCursorOn()` — включает ожидание показа курсора браузером (установлено по умолчанию).
- `waitCursorOff()` — отключает ожидание показа курсора браузером.
- `exitAllowedOn()` — включает прерывание процесса xAjax после того, как запрос был выслан и ответ был получен браузером (установлено по умолчанию).
- `exitAllowedOff()` — отключает прерывание процесса xAjax после того, как запрос был выслан и ответ был получен браузером.
- `errorHandlerOn()` — включает обработку ошибок системой после того, как произошла ошибка в ходе приема запроса и отправки ответа сервером. `errorHandlerOff()` — отключает обработку полученных ошибок xAjax (установлено по умолчанию).

- `setLogFile(string $sFilename)` — определяет log-файл, в который будет записана ошибка при исполнении запроса xAjax. Если вы не вызываете этот метод, то log-файл создаваться не будет.

  Применение:

  ```
  $xajax->setLogFile("/xajax_logs/errors.log");
  ```

- `setWrapperPrefix($sPrefix)` — устанавливает префикс, который будет добавлен к функциям JavaScript (по умолчанию `"xajax_"`).

- `setCharEncoding(string $sEncoding)` — устанавливает кодировку для вывода HTML. Обычно нет необходимости в применении этого метода, т. к. кодировка устанавливается автоматически, на основе глобальной константы `XAJAX_DEFAULT_CHAR_ENCODING`.

  Применение:

  ```
  $xajax->setCharEncoding("windows-1251");
  // или:
  $xajax->setCharEncoding("utf-8");
  ```

- `registerFunction(mixed $mFunction, string $sRequestType=XAJAX_POST)` — регистрация PHP-функции (или метода) для того, чтобы она могла быть вызвана через xAjax в вашем JavaScript-коде. Если вы хотите зарегистрировать статический метод, введите следующий массив:

  ```
  array("myFunctionName", "myClass", "myMethod")
  ```

  > ***Примечание***
  >
  > Для метода экземпляров класса используйте объектную переменную в качестве второго аргумента. (В PHP 4 проверьте, что перед переменной поставлен знак `&`.) Функцию, которую вы вызываете через JavaScript-код, называйте так, чтобы она имела уникальное имя и не конфликтовала с другими функциями.

  Параметры:

  - `$mFunction` — строка, содержащая имя функции или вызываемый массив объектов;
  - `$sRequestType` — тип запроса (`XAJAX_GET`/`XAJAX_POST`), который будет использоваться функцией. По умолчанию `XAJAX_POST`.

  Применение:

  ```
  $xajax->registerFunction("myFunction");
  // или:
  $xajax->registerFunction(array("myFunctionName", &$myObject, "myMethod"));
  ```

- `registerExternalFunction (mixed $mFunction, string $sIncludeFile, string $sRequestType=XAJAX_POST)` — регистрация PHP-функции, которая может вызываться из xAjax и расположена в другом файле. Если функция запрошена, внешний файл будет включен в описание перед тем, как будет вызвана функция.

Параметры:

- `$mFunction` — строка, содержащая имя функции или вызываемый массив объектов (см. `registerFunction()` для более полной информации по вызываемому массиву объектов);
- `$sIncludeFile` — строка, содержащая путь и имя включаемого файла;
- `$sRequestType` — тип запроса (`XAJAX_GET`/`XAJAX_POST`), использованный для этой функции. По умолчанию `XAJAX_POST`.

Применение:

```
$xajax->registerExternalFunction("myFunction","myFunction.inc.php",
                                 XAJAX_POST);
```

- `registerCatchAllFunction(mixed $mFunction)` — регистрирует функцию PHP, которая будет вызвана, если xAjax не может найти функцию, вызываемую посредством JavaScript. Потому что это технически невозможно, когда используются функции "оболочки", метод действует, только когда вызывается напрямую JavaScript-метод `xajax.call()`. Используйте эту возможность для обработки неописанных ситуаций. `$mFunction` строка, содержащая имя функции или вызываемый массив объектов (см. `registerFunction()` для более полной информации по вызываемому массиву объектов).

  Применение:

  ```
  $xajax->registerCatchAllFunction("myCatchAllFunction");
  ```

- `registerPreFunction(mixed $mFunction)` — регистрация функции PHP для исполнения, перед тем как xAjax выполнит запрашиваемую функцию. xAjax будет автоматически добавлять запрос функции на возвращенный ответ "предфункции" для того, чтобы собрать из них один ответ. Другая возможность — это не ждать ответа. Но тогда будет возвращаться массив, где первый элемент равен `false` (`boolean`), а второй является ответом. В этом случае ответ от "предфункции" будет возвращен в браузер без вызова функции, запрошенной xAjax.

  Параметр `$mFunction` — строка, содержащая имя функции или вызываемый массив объектов (см. `registerFunction()` для более полной информации по вызываемому массиву объектов).

  Применение:

  ```
  xajax->registerPreFunction("myPreFunction");
  ```

- `canProcessRequests()` возвращает `true`, если xAjax может обработать запрос, иначе — `false`. Вы можете использовать этот метод для того, чтобы узнать, прошел запрос или нет.
- `getRequestMode()` возвращает текущий вид (`XAJAX_GET` или `XAJAX_POST`).
- `processRequests()` — основной движок для коммуникации в xAjax. Движок обрабатывает все входящие запросы xAjax, вызывает соответствующие PHP-функции (или методы class/object) и выводит XML-ответы назад в JavaScript-

обработчик ответов. Если ваш запрашиваемый URI отличается от URI вашей страницы, то эта функция должна быть вызвана перед выводом заголовков или HTML.

- `printJavascript(string $sJsURI="", string $sJsFile=NULL, string $sJsFullFilename=NULL)` — добавляет в вашу страницу JavaScript-код, который является результатом вывода метода `getJavascript()`. Он вызывается только между тегами в вашей HTML-странице. Помните, если вы хотите получить только результат этой функции, используйте другой метод — `getJavascript()`.

  Параметры:

  - `$sJsURI` — относительный адрес папки, где установлен xAjax. Для PHP-файла, находящегося по адресу **http://www.myserver.com/myfolder/mypage.php**, и xAjax, который был установлен в **http://www.myserver.com/anotherfolder**, по умолчанию предполагается, что xAjax находится в той же папке, что и ваш PHP-файл;
  - `$sJsFile` — относительный путь к папке/файлам библиотеки xAjax JavaScript, расположенным в инсталляционной папке. По умолчанию — xajax_js/xajax.js;
  - `$sJsFullFilename` — может быть установлен абсолютный путь к файлу xajax.js. Этот аргумент нужно использовать, только если вы переместили папку xajax_js.

- `getJavascript(string $sJsURI="", string $sJsFile=NULL, string $sJsFullFilename=NULL)` — возвращает JavaScript-код, который должен быть добавлен в вашу HTML-страницу между тегами. Аргументы функции такие же, как и в методе `printJavascript()`.

1.3. Примеры использования xAjax

Приступим теперь к рассмотрению примеров использования xAjax.

1.3.1. Форма регистрации с проверкой правильности заполнения полей "на лету"

Классический пример применения технологии Ajax — проверка правильности заполнения полей формы до отправки данных на сервер. Немного усложним этот пример. Во-первых, заблокируем возможность нажатия кнопки `submit` до правильного заполнения всех полей формы и, во-вторых, отправим данные на сервер без перезагрузки страницы. Файлы примера расположены на диске в каталоге glava_01\2-1. В Интернете пример доступен по адресу **http://examples-api.bazakatalogov.ru/glava_01/2-1**. В файле index.php идет подключение файлов библиотеки xAjax, создание объекта xAjax, регистрация функций, к которым можно обращаться из JavaScript-кода (`xajax_Control`, `xajax_Result`), и разрешение xAjax обрабатывать асинхронные запросы. Это серверная часть файла index.php. Клиентская часть — HTML-код формы регистрации. Проверка полей формы происходит

следующим образом. Для предотвращения отправки непроверенных данных изначально кнопка `submit` (ее роль играет кнопка **Зарегистрироваться ->**) неактивна (свойство `disable=true`). По событию `onchange` каждого элемента формы `Form1` происходит вызов функции `xajax_Control()`, в которую передаются все данные формы (`xajax.getFormValues("Form1")`). Функция проверяет правильность заполнения каждого поля. В случае неверного заполнения выводится сообщение **ERROR**, при правильном заполнении — **OK**. В случае правильного заполнения всех полей делаем кнопку `submit` активной. Фрагмент файла index.php представлен в листинге 1.1.

Листинг 1.1. Файл index.php

```
...
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
    <title> Пример 2-1 (глава 1) </title>
    <?php $xajax->printJavascript(''); ?>
  </head>
<body>
  <div id=header1><b>Примеры (глава 1 пример 2-1)<br>
              Форма регистрации с проверкой полей "на лету"</b></div>
  <br>
  <!-- Форма -->
  <div id='div1'>
  <form id='Form1' action='JavaScript:void(null);' onsubmit='
               xajax.$("Button_Form1").disabled=true;
               xajax.$("Button_Form1").value="Подождите...";
               xajax_Result(xajax.getFormValues("Form1"));'>
  <table>
    <!-- login -->
    <tr>
      <td>Логин (5-15 букв, цифры)</td>
      <td><input type='text' name='login' id='login' value=''
             onchange='xajax_Control(xajax.getFormValues("Form1"));'></td>
      <td><div id='error_login'><font color='red'>no</font></div></td>
    </tr>
    <!-- password -->
    <tr>
      <td>Пароль (5-15 букв, цифры)</td>
      <td><input type='password' name='password' id='password'
             onchange='xajax_Control(xajax.getFormValues("Form1"));'></td>
      <td><div id='error_password'><font color='red'>no</font></div></td>
    </tr>
    <!-- подтверждение password -->
    <tr>
```

```
      <td>Повторите пароль</td>
      <td><input type='password' name='password1' id='password1' value=''
            onchange='xajax_Control(xajax.getFormValues("Form1"));'></td>
      <td><div id='error_password1'><font color='red'>no</font></div></td>
    </tr>
    <!-- email -->
    <tr>
      <td>Ваш e-mail</td>
      <td><input type='input' name='email' id='email' value=''
            onchange='xajax_Control(xajax.getFormValues("Form1"));'></td>
      <td><div id='error_email'><font color='red'>no</font></div></td>
    </tr>
    <!-- submit -->
    <tr>
      <td></td>
      <td><input type='submit' id='Button_Form1' disabled=true
                 value='Зарегистрироваться ->' ></td>
      <td></td>
    </tr>
  </table>
  </div>
  <!-- блок для вывода результата -->
  <div id='result'></div>
</body>
</html>
```

Функция `Control()` находится в файле control.php. Проверку данных формы проводим, используя регулярные выражения. Если данные поля не соответствуют шаблону, в блок `error` для этого поля выводим сообщение об ошибке (рис. 1.8).

При правильном заполнении всех полей активируется кнопка `submit` (рис. 1.9).

Содержимое файла control.php вы найдете на компакт-диске.

Для отправки значений формы на сервер без перезагрузки страницы блокируем событие `action` формы. Отправку значений будем осуществлять по событию `onsubmit`. Данные передаются xAjax-функции `Result()`, которая находится в файле result.php. Функция формирует контент и выводит его в блок `result` (рис. 1.10). Так же обнуляются поля формы, и кнопка `submit` делается неактивной.

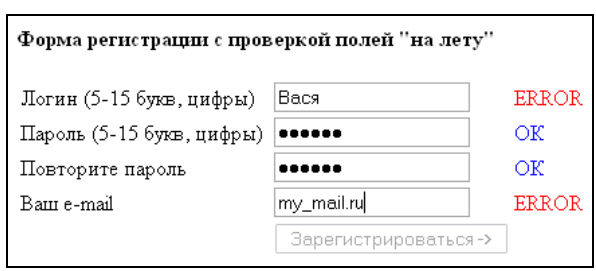


Рис. 1.8. Результат проверки полей формы

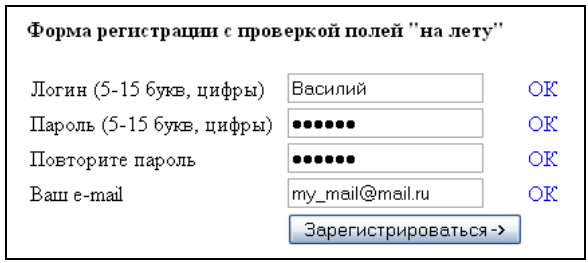


Рис. 1.9. Активация кнопки формы

Рис. 1.10. Отправка значений формы и ответ сервера

Содержимое файла result.php вы найдете на компакт-диске.

Внеся необходимые изменения, можно использовать этот пример в своих проектах.

1.3.2. Динамически подгружаемые *select*-элементы

На многих сайтах вам наверняка приходилось видеть динамически подгружаемые многоуровневые `select`-элементы, когда содержимое нижних элементов динамически меняется в зависимости от выбора в вышестоящем элементе. Пример — выбор населенного пункта на сайте **www.mail.ru** (**Регион -> Район -> Город**).

В качестве примера создадим подобный скрипт и мы. Файлы примера расположены на диске в папке glava_01\2-2. В Интернете пример доступен по адресу **http://examples-api.bazakatalogov.ru/glava_01/2-2**. Для этого нам понадобится готовая база населенных пунктов России. Вы думаете, что найти такую базу в сети нереально? Отчасти это так. Но на сайте Главного научно-исследовательского вычислительного центра Федеральной Налоговой Службы (**www.gnivc.ru**) в свободном доступе находится КЛАДР (Классификатор адресов России). Он состоит из таких значений, как индекс, регион (субъект РФ), район, город, пункт, улица, дом. Справочник представлен в формате XLS. Работу по переводу данных для региона (субъект РФ), район, город, пункт я уже предварительно провел. Дамп базы данных объемом 24 Мбайт находится на диске в каталоге glava_01\2-2\kladr.sql.

Структура таблицы `primer_kladr`:

- `id` — первичный ключ;
- `id_region` — идентификатор региона;
- `id_rayon` — идентификатор района;
- `id_town` — идентификатор города;
- `id_punkt` — идентификатор населенного пункта;
- `name` — название объекта;
- `socr` — наименование типа (край, область, хутор и пр.);
- `zindex` — почтовый индекс.

Дамп для создания структуры таблицы `primer_kladr` представлен в листинге 1.2.

Листинг 1.2. Дамп для создания структуры таблицы `primer_kladr`

```
CREATE TABLE `book_xajax1`.`primer_kladr` (
`id` bigint(12) NOT NULL AUTO_INCREMENT,
`id_region` int(2) NOT NULL default '0',
`id_rayon` int(3) NOT NULL default '0',
`id_town` int(3) NOT NULL default '0',
`id_punkt` int(3) NOT NULL default '0',
`name` varchar(100) CHARACTER SET cp1251 default NULL,
`socr` varchar(10) CHARACTER SET cp1251 default NULL,
`zindex` int(6) default NULL,
UNIQUE KEY `id` (`id`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

Загружаем таблицу `primer_kladr` в базу данных и приступаем к программированию примера. В файле index.php создадим первый уровень вложения — регионы — и добавим еще три пустых блока для подгрузки нижестоящих уровней списка. Кнопка **Выбрать** неактивна до выбора нижнего уровня. По событию `onchange` происходит вызов xAjax-функции `Select_Region()` с передачей всех значений формы. Каждое поле с раскрывающимся списком имеет значение, равное значению КЛАДР для этого пункта. Для значений, предлагающих выбор, оно равно 0. Перед вызовом функции предварительно устанавливается значение поля `number`, равное уровню списка `select`, вызывающему функцию. Содержимое файла index.php вы найдете на компакт-диске.

Функция `Select_Region()` расположена в файле select_region.php. Функция получает значения формы `FormSelectRegion` и в зависимости от значения поля `number` выбирает запрос к базе данных. Из базы выбираются значения для следующего уровня `select`. Для региона — районы и города регионального значения. Для районов — населенные пункты, подчиненные району. Для городов — объекты, подчиненные городу (если есть). По запросу формируется список для нижестоящего уровня и динамически подгружается в блок (рис. 1.11). При этом поле `vibor`, равное текуще-

му выбранному значению, устанавливается в 0. Если объект КЛАДР не имеет нижестоящих объектов, активируется кнопка **Выбрать** и значение поля `vibor` устанавливается равным значению объекта КЛАДР.

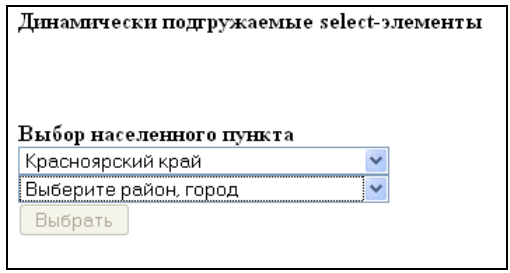


Рис 1.11. Динамическая подгрузка в `select`

Фрагмент файла select_region.php представлен в листинге 1.3.

Листинг 1.3. Файл select_region.php

```
<?php
function Select_Region($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  switch($Id[number])
  { case 1:  //регион
      $objResponse->assign("vibor","value",$Id[selectregion1]);
      if($Id[selectregion1]>0) // получение списка районов, городов
      { $query1="SELECT name,socr FROM ".TABLE1." WHERE
        id=".$Id[selectregion1]." ";
        $text1=mysql_result(mysql_query($query1),0,"name")." ";
        $text1.=mysql_result(mysql_query($query1),0,"socr");
        // формирование select
        $text2="<select name=selectregion2 id='selectregion2'
              onchange='document.getElementById(\"number\").value=2;
               xajax_Select_Region(xajax.getFormValues
               (\"FormSelectRegion\"));'>";
        $text2.="<option value='0' selected>Выберите район, город";
        // формирование options — районы, города областного назначения
        .........

        $text2.="</select>";
        // вывод контента
        $objResponse->assign("divselectregion2","innerHTML",$text2);
        $objResponse->assign("divselectregion3","innerHTML","<table></table>");
        $objResponse->assign("divselectregion4","innerHTML","<table></table>");
        $objResponse->assign("ButtonFormSelectRegion","disabled", true);
      }
```

```
  else // выбор Все
  { $objResponse->assign("vibor","value","0");
    $objResponse->assign("divselectregion2","innerHTML","<table></table>");
    $objResponse->assign("divselectregion3","innerHTML","<table></table>");
    $objResponse->assign("divselectregion4","innerHTML","<table></table>");
    $objResponse->assign("ButtonFormSelectRegion","disabled",true);
  }
  break;
case 2: //
  $objResponse->assign("vibor","value",$Id[selectregion2]);
  if($Id[selectregion2]>0) //
  { if(substr($Id[selectregion2],3,3)<>"000")   // район
    {
      .........
      $text2.="<option value='0' selected>Выберите нас.пункт";
      $query2="SELECT id,name,socr FROM ".TABLE1." WHERE
                  id_region=".substr($Id[selectregion2],1,2)."
                  && id_rayon=".substr($Id[selectregion2],3,3)."
                  && (id_punkt>0 || id_town>0)
                  ORDER BY id ASC ";
      $rez2=mysql_query($query2);
      // формирование select
      .........
      // формирование options — сельские поселения
      .........
      // вывод контента
      .........
    }
    else  // город областного подчинения
    { $query2="SELECT id,name,socr FROM ".TABLE1." WHERE
                  id_region=".substr($Id[selectregion2],1,2)."
                  && id_rayon=0
                  && id_town=".substr($Id[selectregion2],6,3)."
                  ORDER BY id ASC";
      $rez2=mysql_query($query2);
      if(mysql_num_rows($rez2)==1) // нет вложенных
      { $objResponse->assign("ButtonFormSelectRegion","disabled",false);
        $objResponse->assign
           ("divselectregion3","innerHTML","<table></table>");
        $objResponse->assign
           ("divselectregion4","innerHTML","<table></table>");
      }
      else // есть вложенные
      { // формирование select
        .........
        // формирование options — сельские поселения
        .........
      }
```

```
          $text2.="</select>";
          // вывод контента
          .........
        }
      }
    }
      else // Все (районы, города областного значения)
      {
        .........
      }
      break;
    case 3:  // выбор сельских поселений
      if($Id[selectregion3]>0)
      { $objResponse->assign("ButtonFormSelectRegion","disabled",false);
        $objResponse->assign("vibor","value",$Id[selectregion3]);
        $objResponse->assign("divselectregion4","innerHTML","<table></table>");
      }
      else
      { $objResponse->assign("ButtonFormSelectRegion","disabled",true);
        $objResponse->assign("vibor","value","0");
        $objResponse->assign("divselectregion4","innerHTML","<table></table>");
      }
    default: break;
    }
  return $objResponse;
  }
?>
```

При нажатии на кнопку **Выбрать** происходит вызов xAjax-функции `Result_Select()`. Функции передается значение объекта КЛАДР из поля `vibor`. По этому коду формируем полный путь к объекту КЛАДР и выводим результат на страницу (рис. 1.12). Функция `Result_Select()` расположена в файле result_select.php. Содержимое файла result_select.php представлено в листинге 1.4.

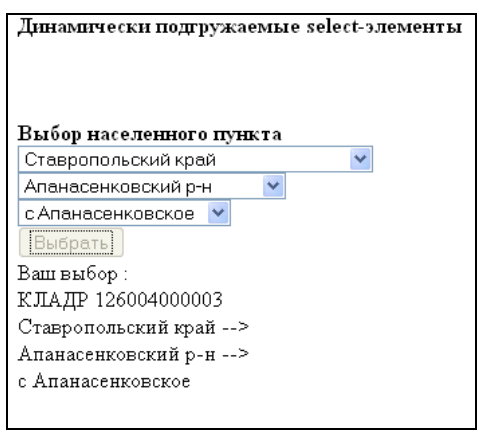


Рис 1.12. Вывод результата выбора

Листинг 1.4. Файл result_select.php

```
<?php
function Result_Select($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");

  $text="Ваш выбор :<br>";
  $text.="КЛАДР ".$Id."<br>";
  $id_region=substr($Id,1,2);
  $id_rayon=substr($Id,3,3);
  $id_town=substr($Id,6,3);
  $id_punkt=substr($Id,9,3);
  // регион
  $query1="SELECT name,socr FROM ".TABLE1." WHERE id_region='".$id_region."' ";
  $text.=mysql_result(mysql_query($query1),0,"name");
  $text.="  ".mysql_result(mysql_query($query1),0,"socr");
  // район
  if($id_rayon>0)
  { $query1="SELECT name,socr FROM ".TABLE1." WHERE
         id_region='".$id_region."' && id_rayon='".$id_rayon."' ";
    $text.=" --> <br>".mysql_result(mysql_query($query1),0,"name");
    $text.="  ".mysql_result(mysql_query($query1),0,"socr");
  }
  // город
  if($id_town>0)
  { $query1="SELECT name,socr FROM ".TABLE1." WHERE
         id_region='".$id_region."' && id_rayon='".$id_rayon."'
         && id_town='".$id_town."' ";
    $text.=" --> <br>".mysql_result(mysql_query($query1),0,"socr");
    $text.="  ".mysql_result(mysql_query($query1),0,"name");
  }
  // нас.пункт
  if($id_punkt>0)
  { $query1="SELECT name,socr FROM ".TABLE1." WHERE
         id_region='".$id_region."' && id_rayon='".$id_rayon."'
         && id_town='".$id_town."' && id_punkt='".$id_punkt."' ";
    $text.=" --> <br>".mysql_result(mysql_query($query1),0,"socr");
    $text.="  ".mysql_result(mysql_query($query1),0,"name");
  }
  $objResponse->assign("div_result","innerHTML",$text);
  $objResponse->assign("ButtonFormSelectRegion","value","Выбрать");
  $objResponse->assign("ButtonFormSelectRegion","disabled",false);

  return $objResponse;
  }
?>
```

1.3.3. Многоуровневый неоднородный каталог

Представим, что на сайте большого магазина есть каталог, который имеет сложную многоуровневую структуру. Допустим, что количество уровней вложенности меняется от одной товарной группы к другой. Это напоминает путешествие по папкам в программе Проводник. Чтобы этот пример можно было сразу использовать в своих проектах, создадим его с использованием баз данных MySQL. Файлы примера расположены на диске в папке glava_01\2-3. В сети пример доступен по адресу **http://examples-api.bazakatalogov.ru5/glava_01/2-3**.

Создадим в базе данных таблицу `primer_katalog`. Дамп для создания таблицы представлен в листинге 1.5.

Структура таблицы `primer_katalog`:

- `id` — первичный ключ;
- `id_parent` — идентификатор родительской категории;
- `name` — наименование товара (категории);
- `type` — тип (категория, товар).

Листинг 1.5. Дамп для создания таблицы базы данных `primer_katalog`

```
CREATE TABLE `book_xajax1`.`primer_katalog` (
`id` int(5) NOT NULL AUTO_INCREMENT ,
`id_parent` int(5) NOT NULL default '0',
`name` varchar(30) NOT NULL default '',
`type` set('tovar', 'kategory') NOT NULL default 'kategory',
UNIQUE KEY `id` (`id`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

Загружаем таблицу `primer_katalog` в базу данных и приступаем к программированию примера. В файле index.php создадим первый уровень каталога. Для этого из базы данных выбираем позиции с `id_parent=0`. Для каждого элемента создаем блок с `id=menu.$id`, где `.$id` — идентификатор элемента в базе данных. Элементы у нас двух типов — каталог и товар, для каждого из них свой значок. Кроме того, для каталога может быть два вида значков (для раскрытой — open_dir.ico, для закрытой — close_dir.ico). При щелчке по значку каталога происходит вызов xAjax-функции `Open_Kategory()` либо `Close_Kategory()` — в зависимости от вида значка. Содержимое файла index.php представлено в листинге 1.6.

Листинг 1.6. Файл index.php

```
<?php
  // подключение библиотеки
  require_once ("xajax_core/xajax.inc.php");
  // подключение внешних файлов
  require_once ("open_kategory.php");
  require_once ("close_kategory.php");
```

```
  // создать новый xajax-объект
  $xajax = new xajax();
  // регистрация функций
  $xajax->register(XAJAX_FUNCTION,"Open_Kategory");
  $xajax->register(XAJAX_FUNCTION,"Close_Kategory");
  // разрешаем обрабатывать xajax асинхронные запросы
  $xajax->processRequest();
  // подключение к базе данных
  require_once("mybaza.php");
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<title> Пример 2-3 (глава 1)  </title>
<?php $xajax->printJavascript(''); ?>
<style type="text/css">
   img { border-style: none }
   div.menu { margin-left:16px;font-size:small; vertical-align:center }
</style>
</head>
<body>
<!-- шапка -->
<div id=header1><b>Примеры к книге (глава 1 пример 2-3)<br>
               Многоуровневый неоднородный каталог </b></div>
<br>
<div id=menu0 class=menu>
  <!-- Вывод первого уровня -->
  <?php
    $text1="";
    $query1="SELECT * FROM ".TABLE1." WHERE id_parent=0 ORDER BY id ASC";
    $rez1=mysql_query($query1);
    while($row1=mysql_fetch_assoc($rez1))
    { if($row1[type]=='kategory')
      $text1.="<div class='menu' id='menu".$row1[id]."'>
          <a href='JavaScript:void();' onclick='
          xajax_Open_Kategory(".$row1[id].")'>
          <img src='open_dir.ico'></a>   ".$row1[name]."</div>";
      else
        $text1.="<div class='menu' id='menu".$row1[id]."'>
          <a href='JavaScript:void();' onclick=''>
          <img src='tovar.ico'></a>   ".$row1[name]."</div>";
    }
    echo $text1;
  ?>
```

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

Функция `Open_Kategory()`, расположенная в файле open_kategory.php, вызывается для раскрытия содержимого элемента каталога. При этом передается идентификатор в базе данных раскрываемой категории. Из базы выбираем данные `cid_parent` равными идентификатору раскрываемой категории. Затем формируются блоки для каждого элемента и записываются в конец блока раскрываемого элемента. Содержимое файла open_kategory.php представлено в листинге 1.7.

Листинг 1.7. Файл open_kategory.php

```
<?php
function Open_Kategory($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  $text1="";
  // текущая категория
  $query0="SELECT * FROM ".TABLE1." WHERE id='".$Id."' ";
  $rez0=mysql_query($query0);
  $row0=mysql_fetch_assoc($rez0);
  $text1.="<a href='JavaScript:void();' onclick='
           xajax_Close_Kategory(".$Id.")'>
           <img src='close_dir.ico'></a>   ".$row0[name]."</a>";
  // получение списка вложенных категорий
  $query1="SELECT * FROM ".TABLE1." WHERE id_parent='".$Id."' ";
  $rez1=mysql_query($query1);
  while($row1=mysql_fetch_assoc($rez1))
  { if($row1[type]=='kategory')
      $text1.="<div class='menu' id='menu".$row1[id]."'>
        <a href='JavaScript:void();' onclick='
        xajax_Open_Kategory(".$row1[id].")'>
        <img src='open_dir.ico'></a>   ".$row1[name]."</div>";
    else
      $text1.="<div class='menu' id='menu".$row1[id]."'>
        <a href='JavaScript:void();' onclick=''>
        <img src='tovar.ico'></a>   ".$row1[name]."</div>";
  }
  $objResponse->assign("menu".$Id,"innerHTML",$text1);
  return $objResponse;
}
?>
```

Примечание

На рис. 1.13 видно постоянное смещение вправо на 16 пикселов для позиций нижестоящего уровня относительно вышестоящего. Это достигается установкой стиля для блоков `div.menu {margin-left:16px}`.

Функция `Close_Kategory()`, расположенная в файле close_kategory.php, вызывается для закрытия содержимого каталога. В блок для закрываемого элемента записываем только контент для этого элемента. Данные для вложенных элементов при этом исчезают. Содержимое файла close_kategory.php представлено в листинге 1.8. Вид каталога можно увидеть на рис. 1.13.

Листинг 1.8. Файл close_kategory.php

```
<?php
function Close_Kategory($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  $text1="";
  // удалить вложенные категории
  $query1="SELECT * FROM ".TABLE1." WHERE id='".$Id."' ";
  $rez1=mysql_query($query1);
  $row1=mysql_fetch_assoc($rez1);
  $text1.="<a href='JavaScript:void();' onclick='
           xajax_Open_Kategory(".$Id.")'>
           <img src='open_dir.ico'></a>   
           ".$row1[name]."</a>";
  $objResponse->assign("menu".$Id,"innerHTML",$text1);
  return $objResponse;
}
?>
```

Рис. 1.13. Многоуровневый каталог

1.3.4. Динамическое управление количеством полей формы

На сайтах частенько приходится решать задачу добавления полей в уже существующую форму или удаления полей из существующей формы. Создадим форму заказа экскурсии для группы людей. В форму необходимо вводить фамилию, имя, отчество (ФИО) каждого человека. Поля для ввода ФИО каждого человека будем добавлять динамически при щелчке по ссылке **Добавить человека**. Также может возникнуть необходимость в удалении из списка данных о человеке. Файлы примера расположены на диске в каталоге glava_01\2-4. В Интернете пример доступен по адресу **http://examples-api.bazakatalogov.ru/glava_01/2-4**. В файле index.php создадим форму с необходимыми полями. Начальный вид страницы представлен на рис. 1.14. Фрагмент файла index.php представлен в листинге 1.9. Экскурсии выбираем из выпадающего списка. Этот пример будем разрабатывать без использования баз данных, поэтому каждое значение `option` будет содержать цену и название экскурсии, разделенные символом `;`. Первое поле `input` для введения ФИО имеет `id=name1`. Последующие созданные будут иметь `id=name2`, `id=name3` и т. д. Так как эти поля могут создаваться и удаляться в произвольном порядке, необходимо иметь указатель на последний созданный `id`. Оно будет храниться в скрытом поле `input` с `id=counted`. Необходимо иметь скрытое поле с указанием удаляемого элемента, его значение будет устанавливаться при щелчке на ссылке **Удалить** перед передачей значений формы xAjax-функции `Delete_Input()`.

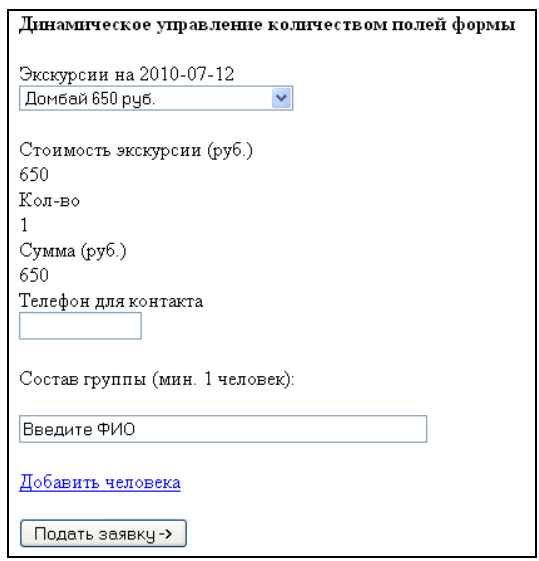


Рис. 1.14. Первоначальный вид формы

Листинг 1.9. Файл index.php

```
// подключение библиотеки
  .........
?>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<title> Пример 2-4 (глава 1) </title>
<?php $xajax->printJavascript(''); ?>
</head>
<body>
<!-- шапка -->
<div id=header1><b>Примеры к книге (глава 1 пример 2-4)<br>
               Динамическое управление количеством полей формы </b></div>
<br>
<!-- Форма -->
<div id='div1'>
<form id='Form1' action='JavaScript:void(null);' onsubmit='
                xajax.$("Button_Form1").disabled=true;
                xajax.$("Button_Form1").value="Подождите...";
                xajax_Result(xajax.getFormValues("Form1"));'>
<table>
<!-- маршрут -->
Экскурсии на
<?php
  echo date('Y-m-d',strtotime("today + 1 day"));
?>
<br>
<select name='select_path' id='select_path'
  onchange='xajax_New_Path(xajax.getFormValues("Form1"));'>
<option value="550;Домбай" selected>Домбай 650 руб.
.........
</select>
<br><br>
<!-- скрытый счетчик -->
<input type='hidden' name='count' id='count' value='1'>
<input type='hidden' name='pay' id='pay' value='650'>
<input type='hidden' name='countid' id='countid' value='1'>
<input type='hidden' name='delete' id='delete' value='0'>
<!-- вывод инфо -->
Стоимость экскурсии (руб.)
<div name='pay1' id='pay1'>650</div>
Кол-во
<div name='people' id='people'>1</div>
Сумма (руб.)
<div name='payall' id='payall'>650</div>
Телефон для контакта<br>
<input type='text' name='phone' id='phone' size=12 maxlength=12><br><br>
<!-- состав группы -->
Состав группы (мин. 1 человек): <br>
```

```
<br>
<input type='text' name='name1' id='name1'
    value='Введите ФИО' size=50 maxlength=50><br><br>
<!-- ссылка Добавить -->
<a href='JavaScript:void();' onclick='
  xajax_Add_Input(xajax.getFormValues("Form1"));'>
Добавить человека</a>
<br><br>
<input type='submit' id='Button_Form1' value='Подать заявку ->' >
</div>
<div id='result'>
</div>
</body>
</html>
```

При щелчке на ссылке **Добавить человека** вызывается xAjax-функция `Add_Input()`, расположенная в файле add_input.php. При этом для добавления поля `input` используем Response-метод `insertInputAfter()`. Затем добавляем элементы `<br>` и `<span>`, а в элемент `<span>` вставляем ссылку на удаление `input`-элемента. Также пересчитываем заново сумму платежа. Содержимое файла add_input.php приведено в листинге 1.10. Вид формы после добавления нового поля представлен на рис. 1.15.

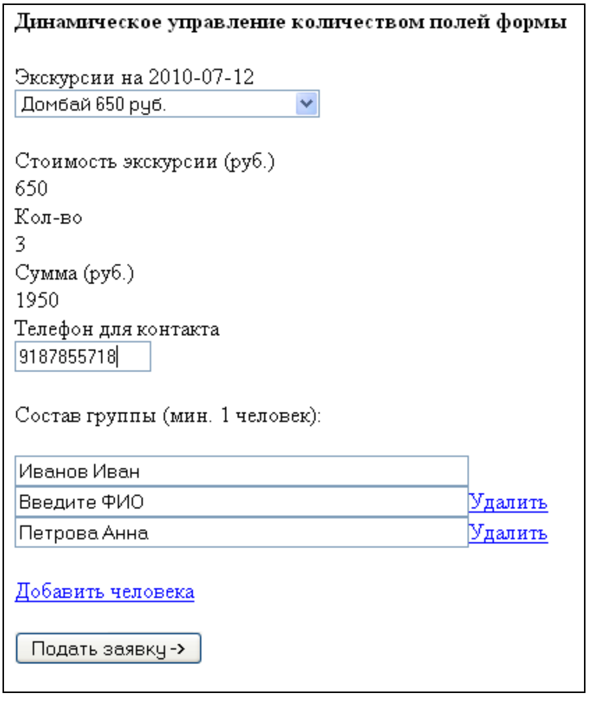


Рис. 1.15. Вид формы после добавления полей input

Примечание

Элеметы и даже
 должны иметь свой уникальный идентификатор, чтобы при удалении поля не изменился вид формы.

Листинг 1.10. Файл add_input.php

```
<?php
function Add_Input($Id)
{ $objResponse = new xajaxResponse();
  // создать input
  $countid=$Id[countid]+1;
  $objResponse->insertInputAfter
      ("name1","input","name".$countid,"name".$countid);
  // вставить <br>
  $objResponse->insertAfter("name1","br","br".$countid);
  // установить стили
  $objResponse->assign("name".$countid,"size","50");
  $objResponse->assign("name".$countid,"value","Введите ФИО");
  // вставить <span> для ссылки Удалить
  $objResponse->insertAfter("name".$countid,"span","span".$countid);
  // ссылка Удалить
  $a="<a href='JavaScript:void();'
      onclick='document.getElementById(\"delete\").value=".$countid.";
      xajax_Delete_Input(xajax.getFormValues(\"Form1\"))'>Удалить</a>";
  $objResponse->assign("span".$countid,"innerHTML",$a);
  // установить новый countid
  $objResponse->assign("countid","value",$countid);
  // установить новый count
  $objResponse->assign("count","value",$Id[count]+1);
  // получить цену
  //$pay1=substr($Id[select_path],0,strpos($Id[select_path],";"));
  // установить цену, кол-во, сумму
  $objResponse->assign("pay1","innerHTML",$Id[pay]);
  $objResponse->assign("people","innerHTML",$Id[count]+1);
  $objResponse->assign("payall","innerHTML",$Id[pay]*($Id[count]+1));
  return $objResponse;
}
?>
```

При щелчке по ссылке **Удалить** вызывается xAjax-функция `Delete_Input()`, расположенная в файле delete_input.php. При этом для удаления поля `input` используем Response-метод `remove()`. Необходимо удалить не только элемент `input`, но и элементы
 и . Так следует пересчитать заново сумму платежа. Содержимое файла delete_input.php представлено в листинге 1.11.

Примечание

При щелчке по ссылке удаления перед вызовом функции `Delete_Input()` необходимо изменить значение скрытого поля `delete` на `id` удаляемого элемента `input`.

Листинг 1.11. Файл delete_input.php

```
<?php
function Delete_Input($Id)
{ $objResponse = new xajaxResponse();
  // удалить input
  $objResponse->remove("name".$Id[delete]);
  // удалить ссылку
  $objResponse->remove("span".$Id[delete]);
  // удалить br
  $objResponse->remove("br".$Id[delete]);
  // установить новый count
  $objResponse->assign("count","value",$Id[count]-1);
  // получить цену
  $pay1=substr($Id[select_path],0,strpos($Id[select_path],";"));
  // установить цену, кол-во, сумму
  $objResponse->assign("pay1","innerHTML",$pay1);
  $objResponse->assign("people","innerHTML",$Id[count]-1);
  $objResponse->assign("payall","innerHTML",$pay1*($Id[count]-1));
  return $objResponse;
}
?>
```

При изменении экскурсии по событию `onchange` вызывается xAjax-функция `New_Path()`, расположенная в файле new_path.php. При этом необходимо пересчитать сумму платежа. Вид формы после выполнения xAjax-функции представлен на рис. 1.16. Содержимое файла new_input.php приведено в листинге 1.12.

Листинг 1.12. Файл new_path.php

```
<?php
function New_Path($Id)
{ $objResponse = new xajaxResponse();
  // получить цену
  $pay1=substr($Id[select_path],0,strpos($Id[select_path],";"));
  // установить новый pay
  $objResponse->assign("pay","value",$pay1);
  // установить цену, кол-во, сумму
  $objResponse->assign("pay1","innerHTML",$pay1);
  $objResponse->assign("people","innerHTML",$Id[count]);
  $objResponse->assign("payall","innerHTML",$pay1*$Id[count]);
  return $objResponse;
}
?>
```

При нажатии кнопки **Подать заявку** вызываем xAjax-функцию `Result()`, передавая ей все значения формы. В реальном проекте эти данные надо заносить в базу дан-

ных; в примере мы формируем контент подтверждения о приеме заказа в блоке `result` (рис. 1.17). Функция `Result()` расположена в файле result.php, содержимое которого представлено в листинге 1.13.

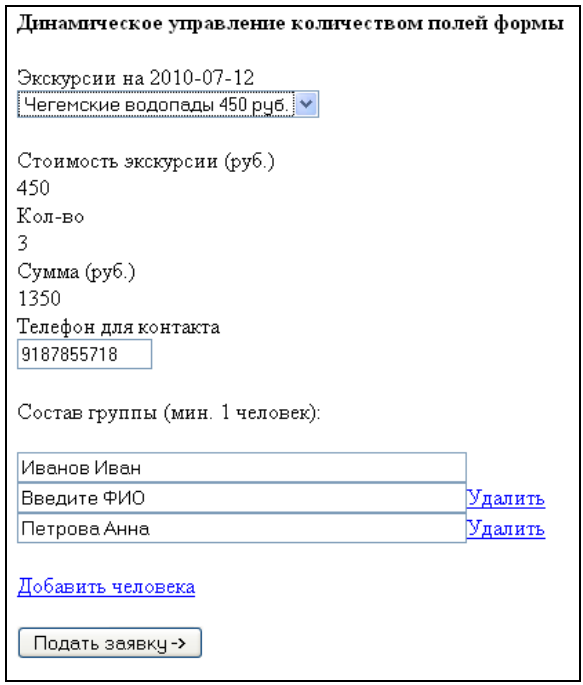


Рис. 1.16. Изменение данных после выбора другой экскурсии

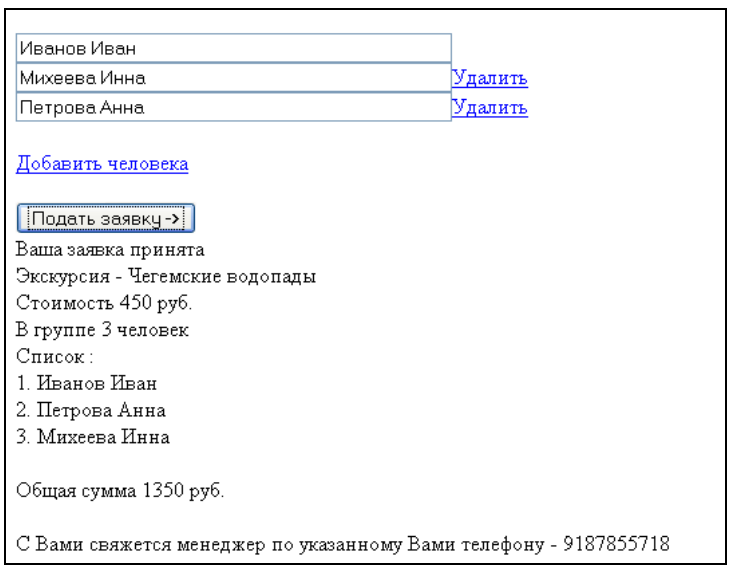


Рис. 1.17. Вывод результата заявки

Примечание

Для переноса блока вывода результата в зону видимости используем JavaScript-функцию `scrollIntoView()`. При этом используется Response-метод `script()`.

Листинг 1.13. Файл result.php

```
<?php
function Result($Id)
{ $objResponse = new xajaxResponse();
  $text="";
  $text="Ваша заявка принята<br>";
  // получить название экскурсии и цену
  $pay=substr($Id[select_path],0,strpos($Id[select_path],";"));
  $path=str_replace($pay.";","",$Id[select_path]);
  $text.="Экскурсия — ".$path."<br>";
  $text.="Стоимость ".$Id[pay]." руб. <br>";
  // список людей
  $text.="В группе ".$Id[count]." человек <br>";
  $text.="Список: <br>";
  for($i=1,$j=1;$i<=$Id[countid];$i++)
  { if(isset($Id['name'.$i]))
    { $text.=$j.". ".$Id['name'.$i]."<br>";
      $j++;  }
  }
  // общая сумма
  $text.="<br>Общая сумма ".$Id[pay]*$Id[count]." руб. <br>";
  // телефон
  $text.="<br>С Вами свяжется менеджер по указанному
    Вами телефону — ".$Id[phone];
  // вывести в блок result
  $objResponse->assign("result","innerHTML",$text);
  // кнопку установить
  $objResponse->assign("Button_Form1","value","Подать заявку ->");
  $objResponse->assign("Button_Form1","disabled",false);
  // блок в поле видимости
  $objResponse->script("document.getElementById('result').scrollIntoView();");
  return $objResponse;
}
?>
```

1.4. Библиотека jQuery

Что такое jQuery? jQuery — это JavaScript-библиотека, которая появилась в январе 2006 года. На сайте разработчиков можно увидеть лозунг: "jQuery is designed to change the way that you write JavaScript". Если переводить это буквально, то получится примерно следующее: "jQuery разработан, чтобы изменить путь, которым Вы

пишете на JavaScript". jQuery помогает легко получать доступ к любому элементу (набору элементов) объектной модели документа (DOM), обращаться к атрибутам и содержимому элементов DOM и, конечно, манипулировать ими. Причем благодаря своему интуитивно понятному синтаксису, схожему в чем-то с CSS1, CSS2 и XPath, эта работа становится не просто легкой, а, я бы сказал, приятной. Также библиотека jQuery предоставляет удобный API (интерфейс программирования приложений) по работе с Ajax. Саму библиотеку можно скачать на сайте разработчиков **http://jquery.com**. На момент написания книги доступна версия 1.5.2. На сайте представлены: хорошо проработанная документация, масса примеров и большое количество плагинов (основные включены отдельными файлами в архив библиотеки), причем плагинов, предназначенных для создания на их основе элементов пользовательских интерфейсов.

1.4.1. Возможности jQuery

Библиотека имеет следующие возможности:

- переход по дереву DOM;
- события;
- визуальные эффекты;
- Ajax-дополнения;
- JavaScript-плагины.

Библиотека jQuery содержит функционал, полезный для максимально широкого круга задач. Тем не менее разработчиками библиотеки не ставилась задача совмещения в jQuery функций, которые подошли бы всем, поскольку это привело бы к большому коду, бóльшая часть которого не востребована. Поэтому была реализована архитектура компактного универсального ядра библиотеки и плагинов. Это позволяет собрать для ресурса именно тот JavaScript-функционал, который на нем был бы востребован.

1.4.2. Использование jQuery

Для скачивания библиотеки jQuery заходим по адресу **http://docs.jquery.com/Downloading_jQuery** и нажимаем правой кнопкой мыши на ссылку **Uncompressed**. Выбираем пункт **Сохранить объект как** (рис. 1.18) и сохраняем файл на компьютере.

jQuery, как правило, включается в веб-страницу как один внешний JavaScript-файл:

```
<head>
  <script type="text/JavaScript" src="js/jQuery-1.5.2.js"></script>
</head>
```

Работу с jQuery можно разделить на 2 типа:

- получение jQuery-объекта с помощью функции `$`;
- вызов глобальных методов у объекта `$`.

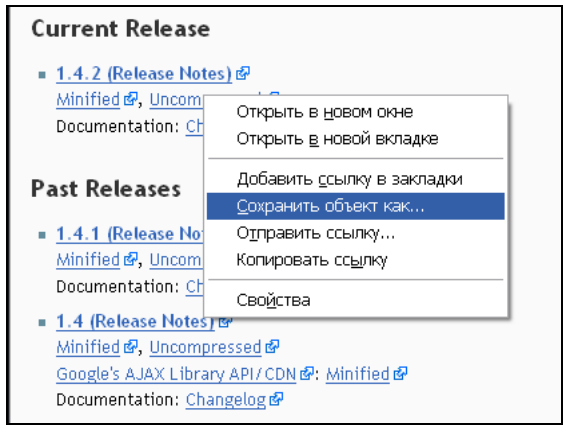


Рис. 1.18. Скачиваем файл библиотеки jQuery

Функция $

Вся работа с jQuery ведется с помощью функции `$`. Если на сайте применяются другие JavaScript-библиотеки, где `$` может использоваться для своих нужд, то можно использовать ее синоним — `jQuery`. Второй способ считается более правильным. А чтобы код не получался слишком громоздким, можно записывать его следующим образом:

```
jQuery(function($)
{
  // Тут код скрипта, где в $ будет jQuery.
})
```

Вне зависимости от параметров, переданных в функцию, знак доллара вернет список объектов, над которым уже определены все доступные jQuery-функции (а их немало). Это позволяет работать с любыми объектами — уже существующими на странице, созданными динамически или полученными через Ajax — так, как будто это одни и те же элементы, уже существующие на странице. jQuery реализован очень интересный механизм поиска элементов, использующий CSS и XPath. То есть для нахождения требуемого элемента вы можете воспользоваться как механизмом селекторов CSS, так и запросами по документу в стиле XPath.

Селекторы

Для того чтобы понимать, как работает селектор, все же необходимы базовые знания CSS, т. к. именно от принципов CSS отталкивается селектор jQuery. Список поддерживаемых селекторов CSS представлен в табл. 1.1.

Таблица 1.1. *Поддерживаемые селекторы CSS*

Селектор	Описание
`*`	Все элементы
`E`	Элемент типа `E`

Таблица 1.1 *(окончание)*

Селектор	Описание
`E:nth-child(n)`	Элемент `E`, являющийся *n*-м дочерним элементом своего родительского элемента
`E:first-child`	Элемент `E`, являющийся первым дочерним элементом своего родительского элемента
`E:last-child`	Элемент `E`, являющийся последним дочерним элементом своего родительского элемента
`E:only-child`	Элемент `E`, являющийся единственным дочерним элементом своего родительского элемента
`E:empty`	Элемент `E`, у которого нет дочерних элементов (включая текстовые узлы)
`E:enabled`	Активный элемент `E` пользовательского интерфейса
`E:disabled`	Неактивный элемент `E` пользовательского интерфейса
`E:checked`	Отмеченный элемент `E` пользовательского интерфейса (например, переключатель)
`E.warning`	Элемент `E` с классом "warning" (`class="warning"`)
`E#myid`	`E` с идентификатором, равным `myid` (выберет максимум один элемент)
`E:not(s)`	Элемент `E`, не соответствующий простому селектору `s`
`E F`	Элемент `F`, являющийся потомком элемента `E`
`E > F`	Элемент `F`, являющийся дочерним элементом элемента `E`
`E + F`	Элемент `F`, которому непосредственно предшествует элемент `E`
`E ~ F`	Элемент `F`, которому предшествует элемент `E`
`E,F,G`	Выбрать все элементы `E`, `F` и `G`

Примеры:

```
$('#element1');
// выбор элемента с id=element1
$('.class1');
// выбор элементов с class=class1
$('div#element1');
// выбор div-элемента с id=element1
$('div.class1');
// выбор div-элементов с class=class1
$('div span');
// выбор всех span-элементов в элементах div
$('div > a');
// выбор всех a-элементов в элементах div, где span является прямым
// потомком div
$('div, a');
// выбор всех div- и a-элементов
```

```
$('a + img');
// выбор всех img-элементов, перед которыми идут a-элементы
$('a ~ img');
// выбор всех img-элементов после первого элемента a
```

Селекторы атрибутов представлены в табл. 1.2. Они должны записываться в стиле XPath, т. е. с предваряющим символом `@`.

Таблица 1.2. *Селекторы атрибутов jQuery*

Селектор	Описание
`E[@foo]`	Элемент `E` с атрибутом `foo`
`E[@foo=bar]`	Элемент `E`, у которого значение атрибута `foo` равно `bar`
`E[@foo^=bar]`	Элемент `E`, у которого значение атрибута `foo` начинается с `bar`
`E[@foo$=bar]`	Элемент `E`, у которого значение атрибута `foo` оканчивается на `bar`
`E[@foo*=bar]`	Элемент `E`, у которого значение атрибута `foo` содержит `bar`

jQuery поддерживает некоторые селекторы, которые не являются частью стандартов CSS или XPath, но их использование облегчает жизнь. Их список представлен в табл. 1.3.

Таблица 1.3. *Собственные селекторы jQuery*

Селектор	Описание
`:even`	Выбирает все четные элементы из коллекции
`:odd`	Выбирает все нечетные элементы из коллекции
`:eq(`*N*`)` и `:nth(`*N*`)`	Выбирает элемент с индексом *N* из коллекции
`:gt(`*N*`)`	Выбирает элементы с индексом большим, чем *N*
`:lt(`*N*`)`	Выбирает элементы с индексом меньшим, чем *N*
`:first`	Выбирает первый элемент из коллекции
`:last`	Выбирает последний элемент из коллекции
`:parent`	Выбирает все элементы, у которых есть дочерние элементы (включая текст)
`:contains('`*text*`')`	Выбирает все элементы, которые содержат указанный текст
`:visible`	Выбирает все видимые элементы (включая элементы со стилями `display` равными `block` или `inline`, `visibility` равным `visible`, а также элементы форм, не являющиеся скрытыми — `hidden`)
`:hidden`	Выбирает все невидимые элементы, включая элементы со стилями `display` равными `none`

Примеры:

```
$('div:even');
// выбираем четные div
$('div:odd');
// выбираем нечетные div
$('div:eq(N)');
// выбираем div, идущий под номером N в DOM
$('div:gt(N)');
// выбираем div, индекс которых больше чем N в DOM
$('div:lt(N)');
// выбираем div, индекс которых меньше чем N в DOM
$('div:contains(text)');
// выбираем div, содержащие текст
$('div:empty');
```

Для работы с элементами форм есть ряд селекторов, позволяющий выбирать по типу элемента и фильтров — `enabled`/`disabled`/`selected`/`checked`. Их список представлен в табл. 1.4.

Таблица 1.4. *Селекторы форм*

Селектор	Описание
`:input`	Выбирает все элементы формы (`input`, `select`, `textarea`, `button`)
`:text`	Выбирает все текстовые поля (`type="text"`)
`:password`	Выбирает все поля для ввода паролей (`type="password"`)
`:radio`	Выбирает все переключатели (`type="radio"`)
`:checkbox`	Выбирает все флажки (`type="checkbox"`)
`:submit`	Выбирает все кнопки для отсылки формы (`type="submit"`)
`:image`	Выбирает все изображения на форме (`type="image"`)
`:reset`	Выбирает все кнопки очистки формы (`type="reset"`)
`:button`	Выбирает все остальные кнопки (`type="button"`)

Примеры:

```
$(":text");
// выбор всех input-элементов с типом, равным text
$(":radio");
// выбор всех input-элементов с типом, равным radio
$("input:enabled");
// выбор всех включенных элементов input
$("input:checked");
// выбор всех отмеченных флажков
```

Методы jQuery

jQuery предлагает разработчику большое количество методов для манипуляции элементами документа и их свойствами. Вот некоторые из этих методов:

- `append(content)` — добавить переданный элемент или выражение в конец выбранного элемента;
- `prepend(content)` — добавить переданный элемент или выражение в начало выбранного элемента;
- `appendTo(expr)` — добавить выбранный элемент в конец переданного элемента;
- `prependTo(expr)` — добавить выбранный элемент в начало переданного элемента;
- `attr(name)` — получить значение атрибута;
- `attr(params)` — установить значение атрибутов, котрые передаются в виде `{ключ1:значение1[, ключ2:значение2[, ...]]}`;
- `attr(name, value)` — установить значение одного атрибута;
- `css(name)` — получить/установить значение отдельных параметров CSS;
- `css(params)` — установить значение отдельных параметров CSS;
- `css(name, value)` — установить значение одного параметра CSS;
- `text()` — получить текст элемента;
- `text(val)` — задать текст элемента;
- `html()` — получить HTML-код элемента;
- `html(val)` — задать HTML-элемент;
- `empty()` — удалить все подэлементы текущего элемента.

Рассмотрим примеры использования этих методов:

```
$("<b>Текст</b>").appendTo($("#div1"));
// добавить <b>Текст</b> в конец элемента с id=div1
$(#div1).empty();
// очистить содержимое элемента с id=div1
$(#div1).prepend("<b>Текст</b>");
// добавить <b>Текст</b> в начало элемента с id=div1
$("#div1").css({backgroundColor: "#F00",color: "#00F"});
// для элемента с id=div1 установить значение цвета и фона
```

Главная особенность большинства методов jQuery — это возможность связывать их в цепочки. Методы, манипулирующие элементами документа, обычно возвращают эти объекты для дальнейшего использования, что позволяет писать примерно следующее:

```
// найти <select id="select1">...</select>
var sel = $("select1");
// добавляем к нему <option value="1">Пример опции</option>
$("<option></option>")
```

```
// создаем требуемый элемент
  .attr("value", 1)
// устанавливаем значение одного из его атрибутов
  .html("Пример опции")
// записываем в него текст
  .appendTo(sel);
// прикрепляем к уже существующему элементу
```

Таким образом, можно легко описать все действия, происходящие с выбранным элементом, не затрудняясь введением большого количества временных переменных.

Обработка событий в jQuery

Далее представлен список событий, поддерживаемых в jQuery. При этом запись в формате

событие()

означает вызов указанного события для каждого элемента набора, вторая запись в формате

событие(*функция*)

определяет назначение функции указанному событию для каждого элемента набора:

- blur(), blur(*функция*);
- change(), change(*функция*);
- click(), click(*функция*);
- dblclick(), dblclick(*функция*);
- error(), error(*функция*);
- focus(), focus(*функция*);
- keydown(), keydown(*функция*);
- keypress(), keypress(*функция*);
- keyup(), keyup(*функция*);
- load(*функция*);
- mousedown(*функция*);
- mouseenter(*функция*);
- mouseleave(*функция*);
- mousemove(*функция*);
- mouseout(*функция*);
- mouseover(*функция*);
- mouseup(*функция*);
- resize(*функция*);
- scroll(*функция*);
- select(), select(*функция*);
- submit(), submit(*функция*);
- unload(*функция*).

Эффекты в jQuery

Грамотная манипуляция свойствами элементов на странице позволила разработчиками JavaScript-библиотек создавать визуальные эффекты, которые раньше были возможны только при использовании технологии Flash. Это и плавное появление, и скрытие объектов, и плавное изменение различных свойств этих объектов (фонового цвета, размеров), и реализация всевозможных элементов интерфейса вроде разделителей (сплиттеров), деревьев, перетягиваемых объектов и сортируемых списков.

jQuery предлагает следующие методы для показа и скрытия элементов:

- `show([speed[, callback]])` — показать элемент;
- `hide([speed[, callback]])` — скрыть элемент;
- `fadeIn(speed[, callback])` — показать элемент путем изменения его прозрачности;
- `fadeOut(speed[, callback])` — скрыть элемент путем изменения его прозрачности;
- `slideDown(speed, callback)` — показать элемент, спустив его сверху;
- `slideUp(speed, callback)` — показать элемент, подняв его снизу.

Здесь: *speed* — скорость в миллисекундах или одно из значений: `slow` (600 мс) или `fast` (200 мс); *callback* — функция, которая будет вызвана после выполнения анимации.

1.4.3. PHP и jQuery

jQuery является на данный день инфраструктурой, к которой написано очень много интересных плагинов, но это все-таки JavaScript-библиотека, а написать большой проект на JavaScript очень сложно. Поддержка Ajax средствами jQuery не может полностью решить эту проблему, т. к. требует написания JavaScript-функций обработки ответов с сервера. Нам нужна PHP-инфраструктура, которая в нужный момент будет осуществлять динамическую подгрузку библиотеки jQuery и плагинов на страницу. Для этой цели будем использовать xAjax. Рассмотрим примеры использования связки xAjax и jQuery.

Динамическая подгрузка jQuery и плагина jCarousel

Для начала создадим простой пример динамической подгрузки на страницу плагина jQuery. Плагин будем подгружать на страницу при щелчке на ссылке, используя библиотеку xAjax. Файлы примера находятся на диске в папке glava_01\2-5. В Интернете пример доступен по адресу **http://examples-api.bazakatalogov.ru/glava_01/2-5** (рис. 1.19).

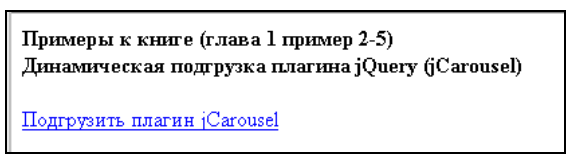


Рис. 1.19. Вид при загрузке примера 2-5

При щелчке на ссылке **Подгрузить js-файл библиотеки jCarousel** подгружаем внешний файл библиотеки jCarousel (вызов xAjax-функции с аргументом `1`). Страница принимает вид, представленный на рис. 1.20.

При щелчке на ссылке **Подгрузить контент** подгружаем контент для плагина (вызов xAjax-функции с аргументом `2`). Страница принимает вид, представленный на рис. 1.21.

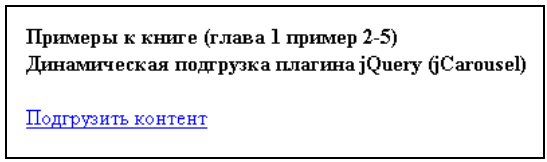


Рис. 1.20. Вид страницы после загрузки js-библиотеки jCarousel

Рис. 1.21. Вид страницы после загрузки контента для плагина jCarousel

При щелчке на ссылке **Запустить плагин** выполняем соответствующее действие (вызов xAjax-функции с аргументом `2`). Страница принимает вид, представленный на рис. 1.22.

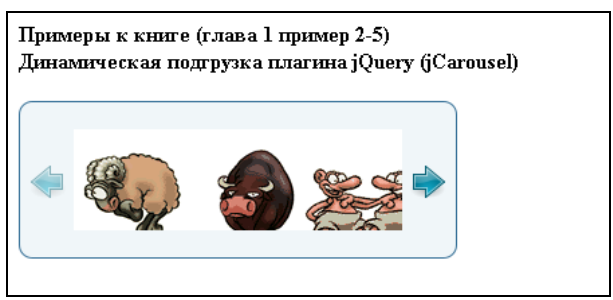


Рис. 1.22. Плагин jCarousel загружен

Содержимое файла index.php представлено в листинге 1.14. Из файла идет запуск xAjax-функции `Plugin()`, которая расположена в файле plugin.php (листинг 1.15).

Листинг 1.14. Файл index.php

```
<?php
require_once ("xajax_core/xajax.inc.php");
  require_once ("plugin.php");
  $xajax = new xajax();
```

```
  // регистрация функций
  $xajax->register(XAJAX_FUNCTION,"Plugin");
  $xajax->processRequest();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<link rel="stylesheet" type="text/css" href="js/skins/tango/skin.css" />
<title> Пример 2-5 (глава 1) к книге </title>
<script type="text/JavaScript" src="js/jquery-1.4.2.js"></script>
<?php $xajax->printJavascript(''); ?>
</head>
<body>
<!-- шапка -->
<div id=header1><b>Примеры к книге (глава 1 пример 2-5)<br>
               Динамическая подгрузка плагина jQuery (jCarousel)</b></div>
<br>
<!-- Форма -->
<div id='div0'>
  <a href='JavaScript:void();' onclick='xajax_Plugin(1);'>
  Подгрузить js-файл библиотеки jCarousel</a>
</div>
  <div id='div1'><!-- Место для загрузки плагина-->
</div>
</body>
</html>
```

Листинг 1.15. Файл plugin.php

```
<?php
// Подгрузка плагина jQuery
function Plugin($Id)
{ $objResponse = new xajaxResponse();
  switch($Id)
  { case 1:
      $objResponse->includeScript("js/jquery.jcarousel.js");
      $text="<a href='JavaScript:void();'
                onclick='xajax_Plugin(2);'>
                Подгрузить контент</a>";
      $objResponse->assign("div0","innerHTML",$text);
      break;
    case 2:
      $text1="
          <ul id='mycarousel' class='jcarousel-skin-tango'>
```

```
          <li><img src='img/img1.gif' /></li>
          <li><img src='img/img2.gif' /></li>
          <li><img src='img/img3.gif' /></li>
          <li><img src='img/img4.gif' /></li>
          <li><img src='img/img5.gif' /></li>
          <li><img src='img/img6.gif' /></li>
          <li><img src='img/img7.gif' /></li>
          <li><img src='img/img8.gif' /></li>
          <li><img src='img/img9.gif' /></li>
          <li><img src='img/img10.gif' /></li>
          <li><img src='img/img11.gif' /></li>
          <li><img src='img/img12.gif' /></li>
          </ul>";
      $objResponse->assign("div1","innerHTML",$text1);
      $text2="<a href='JavaScript:void();' onclick='xajax_Plugin(3);'>
                Запустить плагин</a>";
      $objResponse->assign("div0","innerHTML",$text2);
      break;
    case 3:
      $script="jQuery('#mycarousel').jcarousel();";
      $objResponse->script($script);
      $text2="";
      $objResponse->assign("div0","innerHTML",$text2);
      break;
    default: break;
  }
  return $objResponse;
}
?>
```

Совместное использование jQuery UI-виджетов Tabs и Accordion

jQuery UI — надстройка над JavaScript-библиотекой jQuery. Она помогает создавать по-настоящему интерактивные веб-приложения. Мы будем использовать виджеты Accordion и Tabs. На рис. 1.23 представлен пример использования виджета Accordion. Щелчок по заголовку скрывает/отображает содержимое, разбитое на логические секции. При отображении содержимого одной секции открытая ранее секция обязательно закрывается.

На рис. 1.24 изображен пример использования виджета Tabs. Виджет Tabs помогает разделить информационное наполнение между несколькими вкладками. Это может быть полезно при дефиците свободного места на веб-странице.

В этом примере покажем совместное использование виджетов Tabs и Accordion. Пример можно будет использовать как шаблон для написания большого проекта. Accordion будем использовать как главное меню двух уровней. Заголовки — группы главного меню. В содержимое секций поместим пункты главного меню. При

выборе пункта главного меню в виджете Tabs будет динамически создаваться несколько вкладок со своим содержимым. В Интернете данный пример можно найти по адресу **http://examples-api.bazakatalogov.ru/glava_01/2-6**. Файлы примера находятся на диске в папке glava_01\2-6. Для создания примера создадим в нашей базе данных таблицу `primer_2_6_1`.

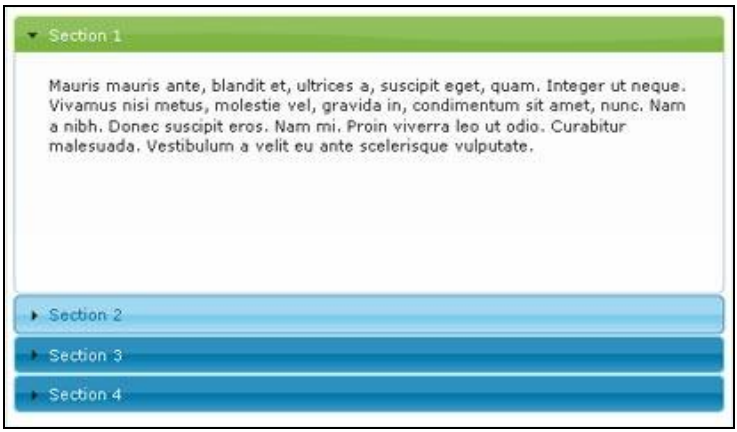


Рис. 1.23. jQuery UI-виджет Accordion

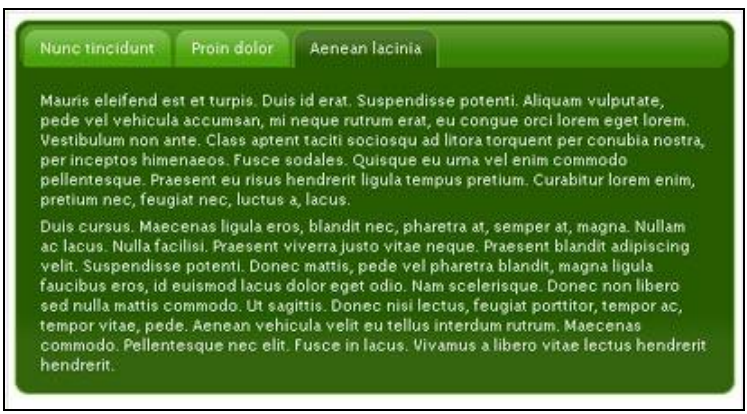


Рис. 1.24. jQuery UI-виджет Tabs

Структура таблицы `primer_2_6_1`:

- `id` — первичный ключ;
- `id_parent` — идентификатор родительского пункта;
- `name` — название пункта;
- `sort` — для сортировки;
- `prgzag` — список названий вкладок, разделенных точкой с запятой (`;`);
- `prgprg` — список программ для заполнения вкладок, разделенных точкой с запятой (`;`).

Дамп для создания структуры таблицы `primer_2_6_1` представлен в листинге 1.16, а также на компакт-диске в папке glava_01\2-6\baza.sql.

Листинг 1.16. Дамп для создания структуры таблицы `primer_2_6_1`

```
CREATE TABLE `primer_2_6_1` (
`id` int(9) NOT NULL AUTO_INCREMENT ,
`id_parent` int(11) default NULL ,
`name` varchar(50) default NULL ,
`sort` int(5) NOT NULL ,
`prgzag` varchar(120) default NULL ,
`prgprg` varchar(120) default NULL ,
UNIQUE KEY `id` (`id`) ,
KEY `sort` (`sort`)
) ENGINE = MYISAM AUTO_INCREMENT = 135DEFAULT CHARSET = cp1251
```

Загружаем таблицу `primer_2_6_1` в базу данных и приступаем к программированию примера.

В файле index.php создадим только три блока: один — для загрузки виджета Accordion, второй — для загрузки виджета Tabs, а третий — для заголовка (для какого пункта Accordion будет загружаться содержимое в Tabs). Фрагмент файла index.php представлен в листинге 1.17. Подключаем библиотеку jQuery, файл библиотеки jQuery UI, функции xAjax. Также подключаем файлы стилевого оформления. В примере выбрана тема `sunny` для jQuery UI. Вы можете выбрать иную, раскомментировав другую строку, все эти темы есть на компакт-диске. По событию `onload` документа сделаем начальную загрузку главного меню — виджета Accordion.

Листинг 1.17. Файл index.php

```
// подключение библиотеки xajax
.........
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<link type="text/css" href="css/sunny/jquery-ui-1.7.3.custom.css"
      rel="stylesheet" />
<title> Пример 2-6 (глава 1) к книге </title>
<script type="text/JavaScript" src="js/jquery-1.4.2.js"></script>
<script src="js/jquery-ui-1.7.3.custom.min.js" type="text/JavaScript"></script>
<?php $xajax->printJavascript(''); ?>
</head>
<body onload='xajax_Create_Accordion();'>
```

```
<!-- шапка -->
<div id=header1><b>Примеры к книге (глава 1 пример 2-6)<br>
   Совместное использование jQuery UI-виджетов Tabs и Accordion </b></div>
<br>
<!--  -->
<table width=100%>
<tr>
<td width=20%>
  <!-- место для Accordion-->
  <div id='accordion1'></div>
</td>
<td width=5%></td>
<td width=75% style='vertical-align:top'>
  <!-- заголовок -->
  <div id='zag1'></div>
  <!-- место для Tabs-->
  <div id='alltabs1'></div>
</td>
</tr>
</table>
</body>
</html>
```

Вызываем xAjax-функцию `Create_Accordion()`, которая расположена в файле create_accordion.php. Для создания Accordion выбором из базы данных создается следующий контент:

```
<h4><a href='#'>Рубрика 1</a></h4>
<div>
<div style='cursor:pointer' onclick='xajax_New_Tabs(4);'>Пункт 1</div>
<div style='cursor:pointer' onclick='xajax_New_Tabs(5);'>Пункт 2</div>
<div style='cursor:pointer' onclick='xajax_New_Tabs(6);'>Пункт 3</div>
</div>
<h4><a href='#'>Рубрика 2</a></h4>
<div>
<div style='cursor:pointer' onclick='xajax_New_Tabs(7);'>Пункт 4</div>
<div style='cursor:pointer' onclick='xajax_New_Tabs(8);'>Пункт 5</div>
</div>
<h4><a href='#'>Рубрика 3</a></h4>
<div>
<div style='cursor:pointer' onclick='xajax_New_Tabs(9);'>Пункт 6</div>
<div style='cursor:pointer' onclick='xajax_New_Tabs(10);'>Пункт 7</div>
</div>
```

Затем этот контент записывается в `div`-блок с `id=accordion1`. И js-код для запуска виджета Accordion:

```
$('#accordion1').accordion({autoHeight:false});
```

Далее созданием вкладку для **Пункт 1**. Выбором из базы данных создается следующий контент:

```
<ul>
  <li id='li1'><a href='#tabs1'>Вкладка 1-1</a></li>";
  <li id='li2'><a href='#tabs2'>Вкладка 1-2</a></li>";
  <li id='li3'><a href='#tabs3'>Вкладка 1-3</a></li>";
  <li id='li4'><a href='#tabs4'>Вкладка 1-4</a></li>";
  <li id='li5'><a href='#tabs5'>Вкладка 1-5</a></li>";
</ul>
<div id='tabs1'>Результат программы для заполнения prg1-1</div>
<div id='tabs2'>Результат программы для заполнения prg1-2</div>
<div id='tabs3'>Результат программы для заполнения prg1-3</div>
<div id='tabs4'>Результат программы для заполнения prg1-4</div>
<div id='tabs5'>Результат программы для заполнения prg1-5</div>
```

Затем этот контент записывается в `div`-блок с `id=alltabs1`. И JavaScript-код для запуска виджета Tabs:

```
$('#alltabs1').tabs({fxFade: true,fxSpeed:'slow'})
```

Содержимое файла create_accordion.php представлено в листинге 1.18.

Листинг 1.18. Файл create_accordion.php

```
<?php
function Create_Accordion()
{ $objResponse = new xajaxResponse();
  // подключиться к базе данных
  require_once("mybaza.php");
  // сформировать контент для accordion
  $content1="";
  $query11="SELECT * FROM ".TABLE1." WHERE id_parent='0' ORDER BY sort ASC ";
  $rez11=mysql_query($query11);
  while($row11=mysql_fetch_assoc($rez11))
  { $content1.="<h4><a href='#'>".$row11[name]."</a></h4>";
    $query12="SELECT * FROM ".TABLE1." WHERE id_parent='".$row11[id]."'
             ORDER BY sort ASC ";
    $rez12=mysql_query($query12);
    $content1.="<div>";
    while($row12=mysql_fetch_assoc($rez12))
    { $content1.="<div style='cursor:pointer'
      onclick='xajax_New_Tabs(".$row12[id].");'>".$row12[name]."</div>";
    }
    $content1.="</div>";
  }
  // подгрузить контент для accordion
  $objResponse->assign("accordion1","innerHTML",$content1);
  // запустить accordion
  $objResponse->script("$('#accordion1').accordion({autoHeight false})");
```

```
  // создаем Tabs для этого пункта Accordion
  // первый пункт первого меню
  $query13="SELECT * FROM ".TABLE1." WHERE id_parent='1'
           ORDER BY sort ASC LIMIT 1, 2 ";
  $rez13=mysql_query($query13);
  $row13=mysql_fetch_assoc($rez13);
  // заголовок
  $objResponse->assign("zag1","innerHTML","<h3>".$row13[name]."</h3>");
  $content21="<ul>";$content22="";
  $arrzag=explode(";",$row13[prgzag]);
  $arrprg=explode(";",$row13[prgprg]);
  for($i=1;$i<count($arrzag);$i++)
  for($i=1;$i<count($arrzag);$i++)
  { if($arrzag[$i-1]<>"")
    { $content21.="<li id='li".$i."'>
                  <a href='#tabs".$i."'>".$arrzag[$i-1]."</a></li>";
      $content22.="<div id='tabs".$i."'>".do_prg($arrprg[$i-1])."</div>";
    }
    else
    { $content21.="<li id='li".$i."'></li>";
      $content22.="<div id='tabs".$i."'></div>";  }
  }
  $content21.="</ul>";
  // загрузка контента для tabs
  $objResponse->assign("alltabs1","innerHTML",$content21.$content22);
  // запуск виджета Tabs
  $objResponse->script("$('#alltabs1').tabs({fxFade: true,fxSpeed:'slow'})");
  return $objResponse;
}
function do_prg($arg)
{ return "Результат программы для заполнения ".$arg; }
?>
```

И результат загрузки страницы изображен на рис. 1.25.

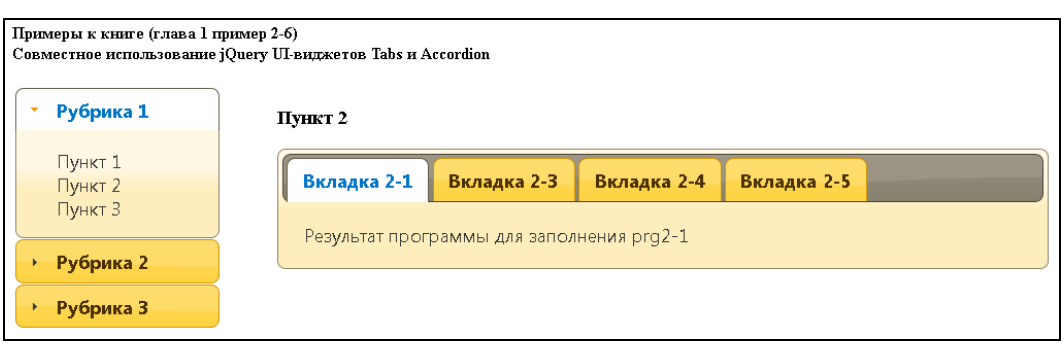


Рис. 1.25. Начальная страница примера 2-6

При выборе другого пункта меню в Accordion вызывается xAjax-функция `New_Tabs()`, расположенная в файле new_tabs.php. Сначала функция очищает содержимое блока с `id=alltabs1`, затем удаляет виджет Tabs, посылая JavaScript-код:

```
$('#alltabs1').tabs('destroy')
```

А затем заново создает новое заполнение вкладок для нового пункта меню и запускает виджет Tabs для блока с `id=alltabs1`. Результат выбора другого пункта меню приведен на рис. 1.26. Содержимое файла new_tabs.php представлено в листинге 1.19.

Листинг 1.19. Файл new_tabs.php

```
<?php
function New_Tabs($Id)
{ $objResponse = new xajaxResponse();
  // очистить содержимое блока alltabs1
  $objResponse->assign("alltabs1","innerHTML","");
  // destroy tabs
  $objResponse->script("$('#alltabs1').tabs('destroy')");
  // подключиться к базе данных
  require_once("mybaza.php");
  // найти пункт меню
  $query13="SELECT * FROM ".TABLE1." WHERE id='".$Id."' ";
  $rez13=mysql_query($query13);
  $row13=mysql_fetch_assoc($rez13);
  // заголовок
  $objResponse->assign("zag1","innerHTML","<h3>".$row13[name]."</h3>");
  // список названий и программ
  $arrzag=explode(";",$row13[prgzag]);
  $arrprg=explode(";",$row13[prgprg]);
  $content21="<ul>";
  for($i=1;$i<count($arrzag);$i++)
  { if($arrzag[$i-1]<>"")
    { $content21.="<li id='li".$i."'><a href='#tabs".$i."'>
                  ".$arrzag[$i-1]."</a></li>";
      $content22.="<div id='tabs".$i."'>".do_prg($arrprg[$i-1])."</div>";
    }
    else
    { $content21.="<li id='li".$i."'></li>";
      $content22.="<div id='tabs".$i."'></div>";  }
  }
  $content21.="</ul>";
  // новый контент в блок alltabs1
  $objResponse->assign("alltabs1","innerHTML",$content21.$content22);
  // запустить tabs
  $objResponse->script("$('#alltabs1').tabs({fxFade: true, fxSpeed: 'slow'})");
  return $objResponse;
}
?>
```

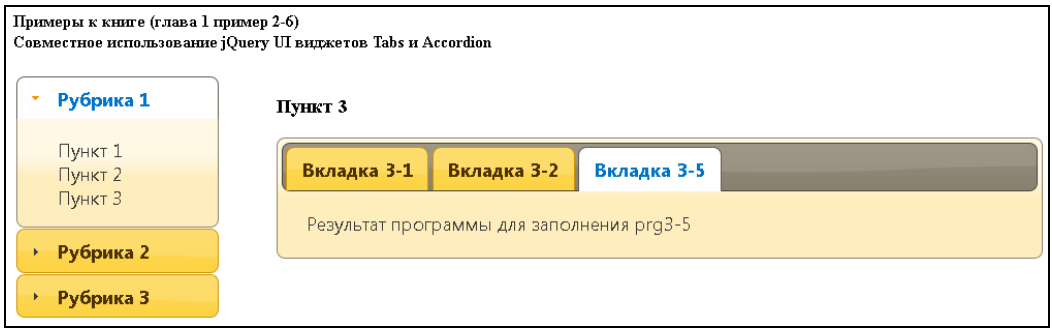


Рис. 1.26. Выбор другого пункта меню

Заполнение контента вкладок Tabs происходит следующим образом. Каждой вкладке в базе данных для каждого пункта меню соответствует свое название программы, которое передается в функцию `do_prg()`, расположенную в файле create_accordion.php. В нашем примере она просто возвращает строку **Результат программы для заполнения...**. Здесь вы можете записать свой обработчик, например:

```
function do_prg($arg)
{ switch($arg)
  { case prg1-1: $result=f11();
         break;
    ...
    case prg5-5: $result=f55();
         break;
  }
  $result;
}
```

ГЛАВА 2

API Яндекса

Яндекс предлагает владельцам сайтов и блогов удобный инструментарий. С ним можно применять функционал сервисов Яндекса в своем проекте и делать его более интересным и информативным. Хотя Яндекс начал предоставлять доступ API своих сервисов позднее Google, сделано за это время немало. И в этой области Яндекс является лидером Рунета. На момент написания книги доступны следующие API Яндекса:

- API Яндекс.Карт;
- API Поиска по блогам;
- виджетная платформа;
- API Яндекс.Фоток;
- API Яндекс.Спеллера;
- Яндекс.Сервер;
- API Яндекс.Детектор;
- хостинг JavaScript-библиотек;
- API Яндекс.Бара;
- API Яндекс.Локатора;
- "Поделиться" в социальных сетях;
- API Яндекс.Директа;
- API Яндекс.Подписок;
- OAuth-авторизация;
- API Я.ру.

По количеству сервисов и доступного API к ним Яндекс немного отстает от Google, но не надо требовать всего и сразу. К тому же в некоторых вещах Яндекс не просто не уступает, а делает проще и лучше (например, Яндекс.Карты или Яндекс.Фотки). К тому же вся документация Яндекс-сервисов на русском языке, что тоже немаловажно. Далее приведено краткое описание некоторых перечисленных API и рассмотрены примеры. API Яндекс.Карт является, пожалуй, самым востребованным и привлекательным для разработчиков и владельцев сайтов, поэтому API Яндекс.Карт рассмотрим наиболее подробно в следующей главе и создадим несколько проектов, использующих API Яндекс.Карт.

2.1. API Яндекс.Бара

API Яндекс.Бара — это инструментарий, который позволяет создавать различные компоненты для панели Яндекса, начиная простой кнопкой и заканчивая сложным функционалом. Любой пользователь Яндекс.Бара сможет добавить ваш компонент в свой браузер.

В качестве примера создадим для Яндекс.Бара свою кнопку с меню выбора ссылок на примеры из книги для *главы 3*. Для создания кнопки с использованием API Яндекс.Бара необходимо выполнить следующие шаги:

1. Создать описание кнопки.
2. Подготовить необходимые ресурсы.
3. Создать пакет.
4. Создать манифест.
5. Создать сборку, описывающую кнопку.

2.1.1. Создание описания

Создадим файл описания кнопки с шестью пунктами подменю. При выборе пункта меню будем переходить на внешний ресурс с примером из книги. Сохраним файл под именем simpleBar.xml. Содержимое файла представлено в листинге 2.1.

Листинг 2.1

```
<?xml version="1.0" encoding="utf-8"?>
<widget name="Примеры из книги API Яндекс.Карт" icon="simpleBar.png"
        xmlns="http://bar.yandex.ru/dev/gui"
        xmlns:f="http://bar.yandex.ru/dev/functional">
    <!-- описываем внешний вид кнопки -->
    <button>
        <icon>simpleBar.png</icon>
        <text>Примеры из книги</text>
        <tooltip>Примеры из книги API Яндекс.Карт</tooltip>
        <url>http://examples-api.bazakatalogov.ru/yandex</url>
        <!-- описываем меню кнопки -->
        <menu>
            <!-- пункт меню -->
            <menuitem>
                <!-- текст пункта меню -->
                <text>Пример 3-2-2</text>
                <!-- адрес перехода браузера при клике на пункт меню  -->
                <url>http://examples-api.bazakatalogov.ru/yandex/</url>
            </menuitem>
            <menuitem>
                <text>Пример 3-2-4</text>
                <url>http://examples-api.bazakatalogov.ru/yandex/</url>
            </menuitem>
            <menuitem>
                <text>Пример 3-2-4</text>
                <url>http://examples-api.bazakatalogov.ru/yandex/</url>
            </menuitem>
```

```
            <menuitem>
                <text>Пример 3-4-3</text>
                <url>http://examples-api.bazakatalogov.ru/yandex/</url>
            </menuitem>
            <menuitem>
                <text>Пример 3-4-4</text>
                <url>http://examples-api.bazakatalogov.ru/yandex/</url>
            </menuitem>
            <menuitem>
                <text>Пример 3-4-5</text>
                <url>http://examples-api.bazakatalogov.ru/yandex/</url>
            </menuitem>
        </menu>
    </button>
</widget>
```

2.1.2. Подготовка необходимых ресурсов

Необходимо подготовить все ресурсы, которые будет использовать кнопка. В нашем случае это значок из файла simpleBar.png (локальный ресурс) и URL, отдающие данные для кнопки в правильном формате.

2.1.3. Создание пакета

Необходимо упаковать файл описания кнопки и локальные ресурсы в ZIP-архив. Ресурсы можно разместить в удобных для вас папках. Пути до файлов локальных ресурсов в описании кнопки указываются относительно корня архива, где лежит файл описания кнопки. Разместим наш архив по адресу **http://examples-api.bazakatalogov.ru/yandex/simpleBar.zip**, откуда он будет скачиваться.

2.1.4. Создание манифеста

Создадим файл манифеста под именем simpleBar.manifest.xml. В теге `<package>` указываем версию пакета (атрибут `version`), поддерживаемую версию платформы (атрибут `platform-min`, на сегодня версия платформы должна быть равна `"1"`) и обязательно адрес, по которому можно скачать пакет в атрибуте `url`. Содержимое файла simpleBar.manifest.xml приведено в листинге 2.2.

Листинг 2.2

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <!-- ссылаемся на нужный пакет -->
    <package version="1" platform-min="1"
       url="http://examples-api.bazakatalogov.ru/yandex/simpleBar.zip"/>
</manifest>
```

Размещаем файл манифеста по адресу **http://examples-api.bazakatalogov.ru/yandex/simpleBar.manifest.xml**, откуда он будет доступен всем желающим.

2.1.5. Создание сборки

Содержимое файла сборки simpleBar.xb.xml представлено в листинге 2.3.

Листинг 2.3

```
<?xml version="1.0" encoding="utf-8"?>
<preset>
    <!-- название набора кнопок -->
    <name>Примеры из книги "API Яндекс, Google и других популярных веб-
сервисов"</name>
    <!-- автор набора кнопок -->
    <author>Петин В. А.</author>
    <!-- значок набора кнопок для отображения в окнах установки и настройки -->
    <icon>http://download.yandex.ru/bar/api/examples/test.png</icon>
    <!-- кнопка (кнопки) в наборе -->
    <widget id="http://examples-
api.bazakatalogov.ru/yandex/simpleBar.manifest.xml#simpleBar"/>
</preset>
```

Здесь тег `<name>` определяет название подборки. В случае с одной кнопкой имеет смысл назвать подборку так же, как кнопку. Тег `<author>` определяет имя автора данной подборки. Это обязательный тег. В теге `<icon>` задается значок, который отображается в окне установки сборки. Значок можно не указывать, в этом случае Яндекс.Бар покажет в окне картинку по умолчанию. Тег `<widget>` определяет кнопку в наборе через атрибут `id` — уникальный идентификатор компонента. Он должен быть уникальным как внутри сборки, так и в рамках всей платформы. Для этого в первой части идентификатора (до символа `#`) используется URL манифеста пакета кнопки, а после символа `#` — идентификатор кнопки, уникальный в рамках пакета. Идентификатор кнопки — это имя файла описания кнопки без расширения xml.

Теперь при переходе по ссылке, указывающей на наш файл сборки — **http://examples-api.bazakatalogov.ru/yandex/simpleBar.xb.xml**, будет предложено установить данный компонент на панель Яндекс.Бара (рис. 2.1).

Теперь в браузере на панели Яндекс.Бара появилась наша кнопка (рис. 2.2). Можете делиться ссылкой на файл проекта **http://examples-api.bazakatalogov.ru/yandex/simpleBar.xb.xml**, чтобы другие могли расположить в своем браузере такую же кнопку.

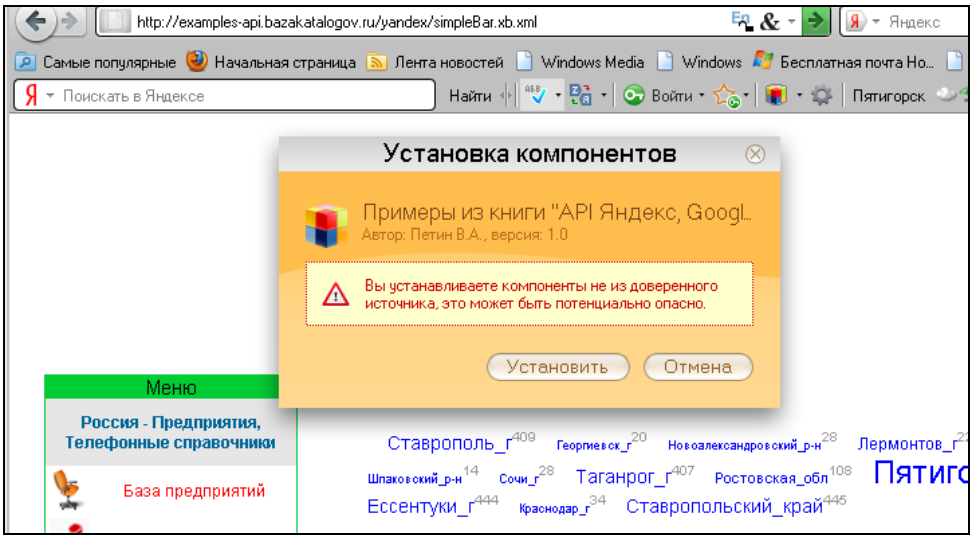


Рис. 2.1. Установка пользовательского компонента Яндекс.Бара

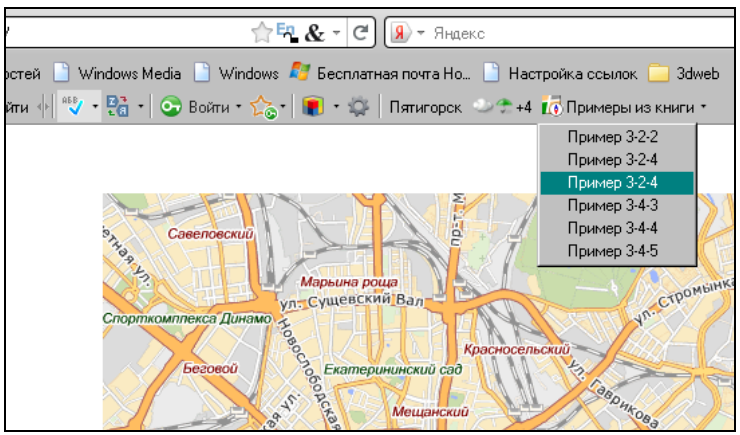


Рис. 2.2. Новая кнопка в панели Яндекс.Бара

2.2. Виджетная платформа

Виджетная платформа позволит вам изменить привычный облик сервисов, устанавливать виджеты Яндекса и его партнеров и создавать собственные виджеты. Виджет — это информационный блок, который содержит фрагмент сайта. Все виджеты, сделанные с помощью технологии API Яндекс.Виджетов, предназначены для установки на главную страницу Яндекса. Пример страницы Яндекса с выбранными пользователем виджетами представлен на рис. 2.3. Для добавления нужного виджета на свою страницу Яндекса необходимо перейти по адресу вида **http://www.yandex.ru/?add=*n*** (***n*** — идентификатор виджета в каталоге) и щелкнуть по ссылке **Оставить**. На данной странице присутствуют виджеты **Калорийность**

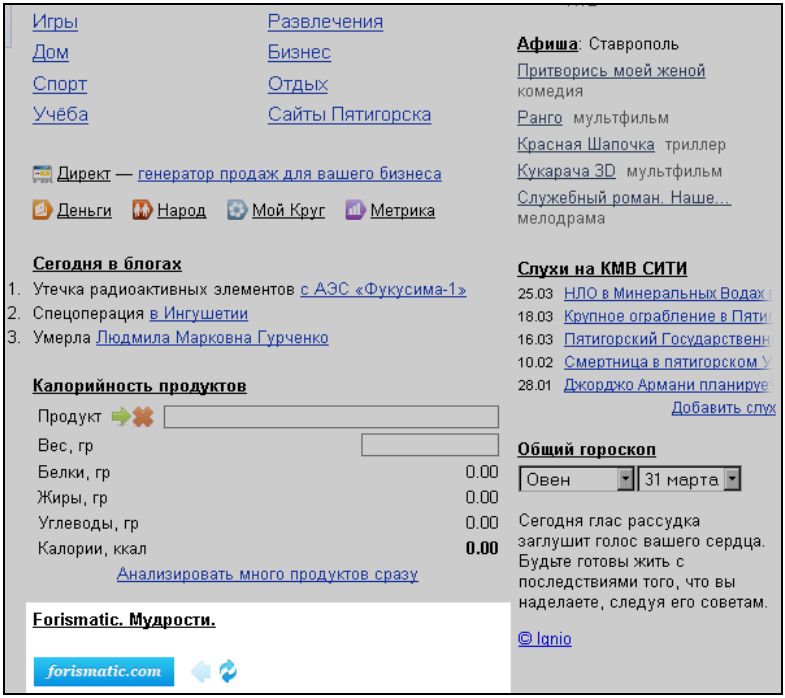


Рис. 2.3. Вид страницы Яндекса с виджетами

продуктов, **Слухи на КМВ СИТИ**, **Общий гороскоп** и добавляется виджет **Forismatic.Мудрости**.

Создадим свой виджет. Будем создавать iFrame-виджет. Авторизуемся в Яндексе и перейдем по адресу **http://wdgt.yandex.ru/add/?type=iframe** (рис. 2.4). Заполним форму, загрузим картинку и виджет, а затем нажмем кнопку **Загрузить виджет**.

Система сообщит нам об успешно созданном виджете (рис. 2.5). По ссылке **Ссылка для установки на Яндекс** (в правом нижнем углу) находим ссылку для добавления нашего виджета на страницу Яндекса — **http://www.yandex.ru/?add=55271**. Эту ссылку можете предлагать для добавления вашего виджета в Яндекс-страницу пользователя. При переходе по этой ссылке пользователь сможет добавить ваш виджет на свою страницу Яндекса (рис. 2.6).

В листинге 2.4 приведен код созданного виджета (файл my_widjet_11.html).

Листинг 2.4

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:widget="http://widget.yandex.ru/ns/">
  <head>
    <meta name="title" content="Сайты на xAjax"/>
    <meta name="type" content="iframe"/>
```

```
    <meta name="description"
       content="Показывает ссылки на сайты, созданные по технологии xAjax"/>
    <meta name="language" content="ru"/>
    <style type="text/css">
      body {
        background: transparence;
        margin: 0;
        font-family: Arial;
        font-size: 13px;
        color: #000000;
      }
      a {color:#1a3dc1;}
    </style>
    <meta name="height" content="150"/>
  </head>
  <body onload>
    <p><a href='http://goodtovars.ru' target='_blank'>Магазин цифровых
          товаров</a></p>
    <p><a href='http://bazakatalogov.ru' target='_blank'>Online
          справочники</a></p>
    <p><a href='http://lermontov-kmv.ru' target='_blank'>Городской
         портал</a></p>
  </body>
</html>
```

Подробную документацию и руководство разработчика вы можете найти по адресу **http://api.yandex.ru/wdgt/doc**.

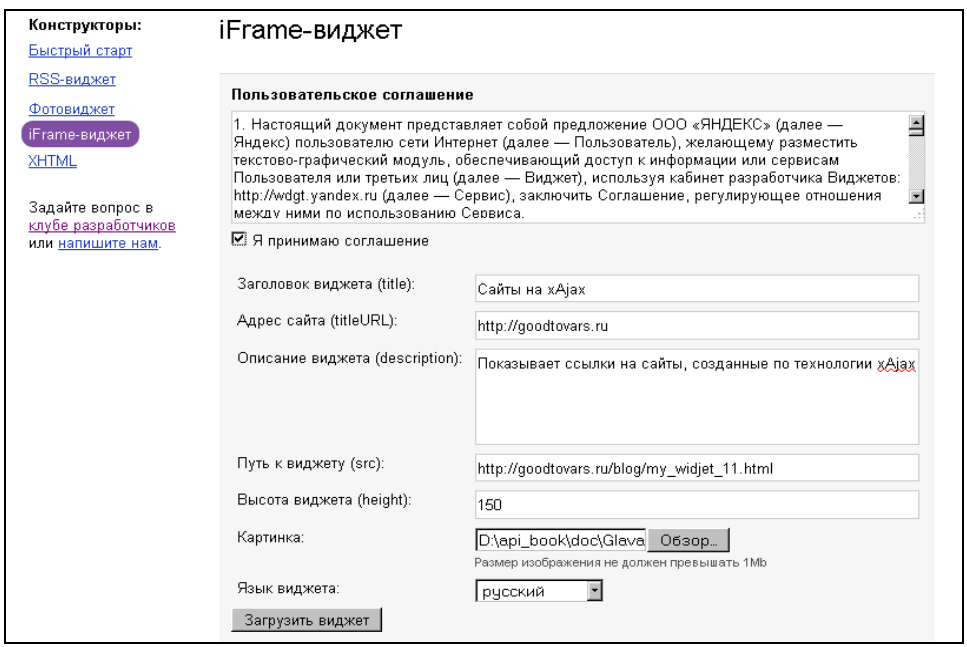


Рис. 2.4. Страница загрузки виджета

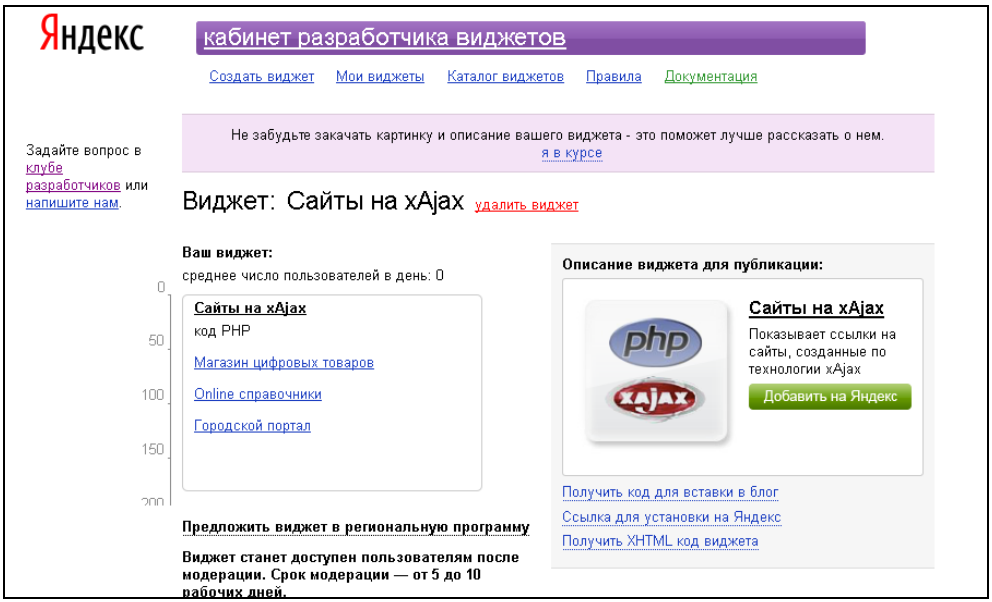


Рис. 2.5. Виджет создан

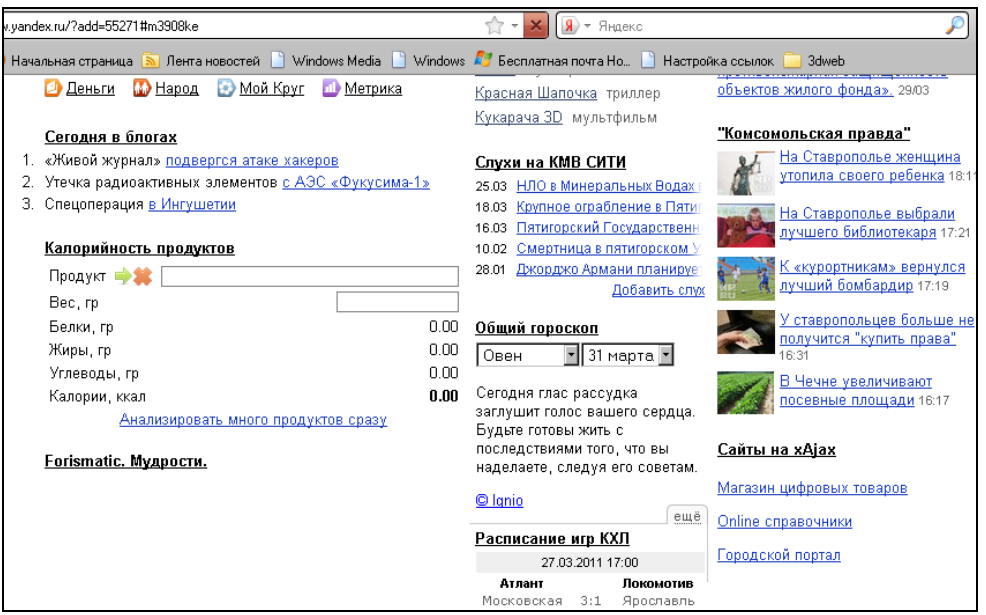


Рис. 2.6. Виджет добавлен на страницу

2.3. API Яндекс.Спеллера

API Яндекс.Спеллера — это сервис проверки правописания, который помогает находить и исправлять орфографические ошибки в ваших текстах. Работа сервиса основана на использовании орфографического словаря. В настоящее время

Яндекс.Спеллер проверяет тексты на русском, украинском и английском языках. Этот сервис анализирует слова, основываясь на правилах орфографии и лексике современного языка. В качестве словарного источника используется орфографический словарь, содержащий правильные написания большинства наиболее употребимых слов.

Сервис в настоящее время поддерживает три языка:

- русский — словарь содержит 3,6 млн словоформ;
- украинский — словарь содержит 1,8 млн словоформ;
- английский — словарь содержит 150 тыс. словоформ.

Яндекс.Спеллер выявляет и исправляет только орфографические ошибки, такие как:

- неправильные, пропущенные или лишние буквы;
- ошибки капитализации (неправильное употребление прописных и строчных букв);
- повторы слов.

Пунктуационные, грамматические (ошибки согласования слов) и стилистические ошибки не исправляются.

Яндекс.Спеллер позволяет встроить функцию проверки правописания в свое приложение с помощью API Яндекс.Спеллера (JavaScript или HTTP).

2.3.1. Web Service API

Web Service API Яндекс.Спеллер поддерживает HTTP GET- и POST-запросы и работает в кодировке UTF-8. Чтобы изменить кодировку входного запроса, следует передать дополнительный параметр `ie` (от англ. *input encoding*). Возможные значения: `ie=utf-8` или `ie=1251`.

Для доступа к Яндекс.Спеллеру по HTTP предлагаются XML-, SOAP-, JSON- и JSONP-интерфейсы. Все интерфейсы обеспечивают одинаковую функциональность и используют одни и те же входные параметры.

XML- и SOAP-интерфейсы возвращают ответ в виде XML-документа, JSON-интерфейс вместо XML-элементов возвращает JavaScript-объекты с теми же именами и семантикой, а JSONP-интерфейс возвращает те же самые JavaScript-объекты, но в виде функций обратного вызова с заданным именем.

Доступ к API предоставляется по следующим URL:

- XML- и SOAP-интерфейс — **http://speller.yandex.net/services/spellservice**;
- WSDL-документ для SOAP доступен по адресу **http://speller.yandex.net/services/spellservice?WSDL**;
- JSON- и JSONP-интерфейс — **http://speller.yandex.net/services/spellservice.json**.

Для включения JSONP-интерфейса требуется передать дополнительный параметр `callback` с именем функции обратного вызова. Web Service API включает в себя два

метода, которые позволяют проверять правописание в одном или нескольких фрагментах текстов:

- метод `checkText()` — проверка орфографии в указанном фрагменте текста;
- метод `checkTexts()` — проверка орфографии в нескольких указанных фрагментах текста. Для каждого фрагмента возвращается отдельный массив ошибок с подсказками.

Пример запроса по методу `checkText()` с использованием XML-интерфейса:

```
http://speller.yandex.net/services/spellservice/checkText?text=
праграммирование+элиментов+форм&lang=ru&format=plain&options=5
```

Обязательно URL-кодирование, поэтому правильная ссылка выглядит так:

```
http://speller.yandex.net/services/spellservice/checkText?text=
%20%20%EF%F0%E0%E3%F0%E0%EC%EC%E8%F0%EE%E2%E0%ED%E8%E5+%FD%EB%E8%EC%E5%ED%F2%EE
%E2+%F4%EE%F0%EC&lang=ru&format=plain&options=5
```

Назначение параметров следующее:

- `text` — текст для проверки;
- `lang` — языки проверки (перечисляются через запятую):
  - `ru` — русский;
  - `uk` — украинский;
  - `en` — английский;
- `format` — формат проверяемого текста:
  - `plain` — текст без разметки (значение по умолчанию);
  - `html` — HTML-текст;
- `options` — сумма опций:
  - `1` (IGNORE_UPPERCASE) — пропускать слова, написанные прописными буквами;
  - `2` (IGNORE_DIGITS) — пропускать слова с цифрами;
  - `4` (IGNORE_URLS) — пропускать интернет-адреса, почтовые адреса и имена файлов;
  - `8` (FIND_REPEAT_WORDS) — подсвечивать повторы слов, идущие подряд;
  - `16` (IGNORE_LATIN) — пропускать слова, написанные латиницей;
  - `32` (NO_SUGGEST) — только проверять текст, не выдавая вариантов для замены;
  - `128` (FLAG_LATIN) — отмечать слова, написанные латиницей, как ошибочные;
  - `256` (BY_WORDS) — не использовать словарное окружение (контекст) при проверке;

- 512 (IGNORE_CAPITALIZATION) — игнорировать неверное употребление прописных/строчных букв.

В XML-интерфейсе возвратится ответ в виде XML-документа с корневым элементом SpellResult (листинг 2.5).

Листинг 2.5

```
<?xml version="1.0" encoding="utf-8" ?>
<SpellResult>
  <error code="1" pos="2" row="0" col="2" len="16">
     <word>праграммирование</word>
     <s>программирование</s>
  </error>
  <error code="1" pos="19" row="0" col="19" len="9">
     <word>элиментов</word>
     <s>элементов</s>
  </error>
</SpellResult>
```

2.3.2. JavaScript API

JavaScript API предназначен для подключения Яндекс.Спеллера к клиентской HTML-странице. Интерфейс реализован в файле spell.js и состоит из единственного класса Speller.

Web Service API Яндекс.Спеллер состоит из двух частей — клиентской и серверной.

Клиентская часть включает три диалоговых окна (проверки правописания, настроек и редактирования пользовательского словаря) и скрипт spell.js. Клиентская часть должна быть установлена на том же сервере, где размещается сайт.

HTML-код клиентской части доступен для загрузки по адресу:

http://speller.yandex.net/speller/1.0/spell-1.0.zip

Серверная часть — это XML-сервис проверки орфографии Яндекса — Web Service API *(см. разд. 2.3.1)*.

Для подключения Веб-Спеллера необходимо выполнить следующие шаги:

1. Загрузить архив spell-1.0.zip и распаковать его в каталог \speller вашего веб-приложения, в архиве находятся все необходимые для демонстрации возможностей Веб-Спеллера файлы:
   - spell.css — файл со стилями;
   - spell.js — JavaScript-файл клиентской части Веб-Спеллера;
   - spelldlg.html — диалоговое окно проверки правописания;
   - spellopt.html — диалоговое окно настроек Веб-Спеллера;

- userdic.html — диалоговое окно редактирования пользовательского словаря (работает в Internet Explorer 8 и Firefox 3);
- test.html — простая HTML-страница с формой для ввода текста и двумя кнопками для вызова диалоговых окон.

2. На страницу с проверяемым текстом добавить клиентский скрипт и функцию `spellCheck()` (листинг 2.6).
3. Добавить кнопки вызова диалоговых окон для проверки правописания и настроек Веб-Спеллера:

```
<button name="cmdSpell" type="button" onclick="spellCheck()">
Проверить правописание</button>
<button type="button" onclick="speller.optionsDialog()">
Параметры...</button>
```

Для проверки работоспособности приложения наберите в поле ввода произвольный текст (разумеется, с ошибками) и нажмите кнопку **Проверить правописание**. Пример использования можно посмотреть на странице **http://speller.yandex.net/speller/1.0/index.html** (рис. 2.7).

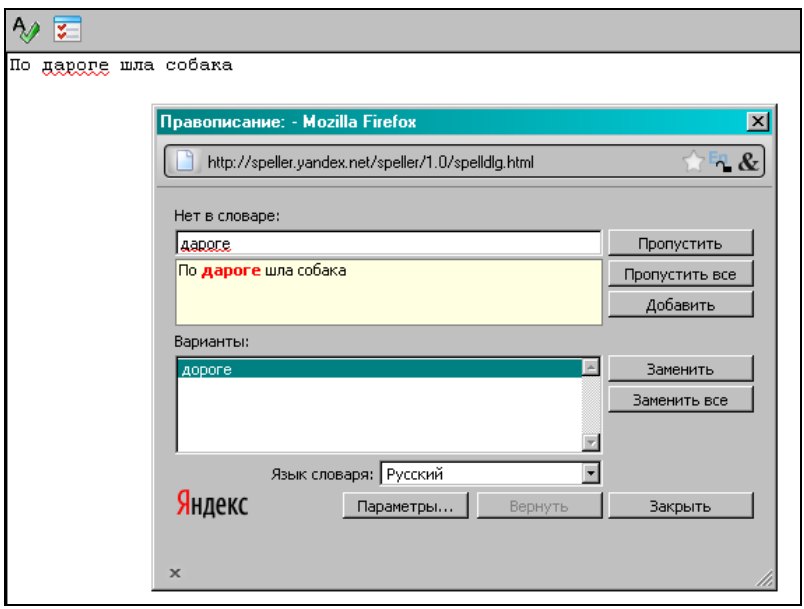


Рис. 2.7. Проверка правописания с помощью JavaScript API Яндекс.Спеллера

Листинг 2.6

```
<script type="text/javascript" src="speller/spell.js"></script>
<script type="text/javascript">
  var speller = new Speller({ url:"speller", lang:"ru",
                  options:Speller.IGNORE_URLS });
```

```
  function spellCheck()
  { var form = document.forms["myform"];
    speller.check([ form.ctrl_1, form.ctrl_2, ..., form.ctrl_N ]);
  }
</script>
```

2.4. API Поиска по блогам

API Поиска по блогам Яндекса поможет организовать детальный поиск по вашему форуму, блогу или даже всему блогохостингу. С помощью API Поиска по блогам можно организовать детальный поиск:

- по личному блогу, включая поиск по комментариям;
- по своему блогохостингу;
- по профилям на своем блогохостинге или в социальной сети;
- по форуму.

Для поиска по записям и комментариям необходимо отправить запрос на адрес **http://blogs.yandex.ru/search.rss**. В ответ вернется RSS-поток с результатами. Запрос можно сформировать при помощи GET-параметров адресной строки или при помощи языка запросов Поиска по блогам. GET-параметры, которые могут быть использованы для формирования запросов к Поиску по блогам, представлены в табл. 2.1.

Таблица 2.1. *GET-параметры*

Категория	Параметр	Значения
Текст запроса	**text**	Текст запроса, значение параметра должно быть URL-кодировано (URL-encoded)
	Запрос "состав зенит 2011": **http://blogs.yandex.ru/search.rss?text=состав+зенит+2011**	
Область поиска	**ft**	**popular** — популярные блоги
		blog — записи из блогов
		community — сообщества
		personal — личные блоги
		comments — комментарии из блогов
		forum — только форумы
		all — все блоги и форумы
	Можно указать несколько значений через запятую. Поиск слова "парламент" только в записях блогов: **http://blogs.yandex.ru/search.rss?text=парламент&ft=blog** Поиск слова "Англия" в записях блогов и комментариях: **http://blogs.yandex.ru/search.rss?text=Англия&ft=blog,comments**	

Таблица 2.1 *(продолжение)*

Категория	Параметр	Значения
Дата	**from_day**	С указанного дня
	from_month	С указанного месяца
	from_year	С указанного года
	to_day	По указанный день
	to_month	По указанный месяц
	to_year	По указанный год
	Поиск всех записей со словами "новый год", созданных в период с 29 по 31 декабря 2010 года: **http://blogs.yandex.ru/search.rss?text=новый%20год&from_day=29&from_month=12&from_year=2010&to_day=31&to_month=12&to_year=2010**	
Сервер	**server**	Для поиска только на указанном сервере, имена нескольких серверов перечисляются через запятую
	x_server=on	Чтобы исключить сервер из поиска, нужно передать дополнительный параметр **x_server** со значением **on**
	Поиск на серверах livejournal.com и diary.ru записей, в которых встречается слово "Лермонтов": **http://blogs.yandex.ru/search.rss?text=Лермонтов&server=км.ru,kmvcity.ru** Аналогичный поиск на всех серверах, кроме указанных: **http://blogs.yandex.ru/search.rss?text=Лермонтов&server=км.ru,kmvcity.ru&x_server=on**	
Записи и комментарии одного блоггера	**author**	Ник автора записей и комментариев или адрес блога автора
	x_author = on	Искать по всем блогам, кроме блогов автора, указанного в параметре **author**
	journal	Журнал, в котором нужно искать записи и комментарии. В качестве параметра следует указать ник или адрес блога автора журнала
	Поиск всех записей и комментариев автора anton: **http://blogs.yandex.ru/search.rss?author=anton** Поиск записей и комментариев в определенном журнале: **http://blogs.yandex.ru/search.rss?journal=kukutz.livejournal.com**	
Журналы друзей	**f_user**	Поиск среди друзей указанного блоггера
	f_server	Адрес блогохостинга друзей блоггера, ник которого указан в параметре **f_user**
	Поиск слова "аршавин" среди блогов всех друзей пользователя fan1983 на сервере livejournal.com: **http://blogs.yandex.ru/search.rss?text=аршавин&f_user=fan1983&f_server=livejournal.com**	

Таблица 2.1 *(окончание)*

Категория	Параметр	Значения
Пол, возраст и место жительства	**gender**	Пол автора блога: **male** — мужской, **female** — женский
	age_from	Возрастная категория авторов, нижняя граница
	age_to	Возрастная категория авторов, верхняя граница
	geo	Указывает местонахождение пользователя. В качестве значения можно передавать названия городов, регионов и стран
	Поиск слова "Rammstein" в регионе "Россия" у блоггеров мужского пола в возрасте от 15 до 20 лет: **http://blogs.yandex.ru/search.rss?text=Rammstein&geo=Россия&gender=1&age_from=15&age_to=20**	
Категории, настроение, музыка	**music**	Позволяет искать информацию о том, какую музыку слушал автор в момент написания сообщения (не все блоги поддерживают такую возможность)
	mood	Позволяет искать информацию о том, какое настроение было у автора в момент написания сообщения (не все блоги поддерживают такую возможность)
	category	Позволяет искать по категориям (тегам) сообщения
	x_category = on	Исключает указанную в параметре **category** категорию из поиска
	Поиск записей блоггеров, содержащих обсуждения матчей команды "Спартак". Мы не хотим, чтобы в выдаче обсуждался одноименный фильм (категория "кино"): **http://blogs.yandex.ru/search.rss?text=спартак&category=кино&x_category=on**	
Параметры выдачи	**numdoc**	Количество найденных записей на странице результата поиска
	p	Начальная страница поиска
	Результаты по запросу "футбол", начиная с третьей страницы. Установим количество результатов на странице равным десяти: **http://blogs.yandex.ru/search.rss?text=футбол&numdoc=10&p=3**	

Поиск по блогам поддерживает все стандартные операторы языка запросов Яндекса и предоставляет несколько дополнительных конструкций, оптимизированных для поиска по блогам, комментариям и форумам. Конструкции, сформированные при помощи языка запросов, можно передавать программно при помощи GET-параметра `text`. Яндекс обладает мощным языком запросов, позволяющим наиболее точно формулировать свой запрос поисковой системе, учитывая малейшие нюансы ее поведения. Знание языка запросов дает возможность решать самые сложные поисковые задачи. Памятка по использованию языка запросов приведена в табл. 2.2.

Таблица 2.2. *Памятка по использованию языка запросов Яндекса*

Пример	Значение
"К нам на утренний рассол"	Слова идут подряд в точной форме
"Прибыл * посол"	Пропущено слово в цитате
полгорбушки & мосол	Слова в пределах одного предложения
снаряжайся && добудь	Слова в пределах одного документа
глухаря | куропатку | кого-нибудь	Поиск любого из слов
не смогешь << винить	Неранжирующее "и": выражение после оператора не влияет на позицию документа в выдаче
я должон /2 казнить	Расстояние в пределах двух слов в любую сторону (т. е. между заданными словами может встречаться одно слово)
государственное дело && /3 улавливаешь нить	Расстояние в 3 предложения в любую сторону
нешто я ~~ пойму	Исключение слова "пойму" из поиска
при моем /+2 уму	Расстояние в пределах двух слов в прямом порядке
чай ~ лаптем	Поиск предложения, где слово "чай" встречается без слова "лаптем"
щи /(-1 +2) хлебаю	Расстояние от одного слова в обратном порядке до двух слов в прямом
!Соображаю !что !чему	Слова в точной форме с заданным регистром
получается && (+на | !мне)	Скобки формируют группы в сложных запросах
!!политика	Словарная форма слова
title:(в стране)	Поиск по заголовкам документов
url:ptici.narod.ru/ptici/kuropatka.htm	Поиск по URL
беспременно inurl:vojne	Поиск с учетом фрагмента URL
host:lib.ru	Поиск по хосту
rhost:ru.lib.*	Поиск по хосту в обратной записи
site:http://www.lib.ru/PXESY/FILATOW	Поиск по всем поддоменам и страницам заданного сайта
mime:pdf	Поиск по одному типу файлов
lang:en	Поиск с ограничением по языку
domain:ru	Поиск с ограничением по домену
date:200712*	Поиск с ограничением по дате
date:20071215..20080101, date:>20091231	Поиск с ограничением по интервалу дат
cat:11000051	Поиск по рубрике Яндекс.Каталога

Создадим в качестве примера сервис-тренажер составления запросов при поиске по блогам с выдачей результатов запроса. Вид тренажера при запуске представлен на рис. 2.8. Форма выбора параметров позволяет указать различные параметры запроса к API Яндекс.Поиск по блогам. Параметры берем из табл. 2.1.

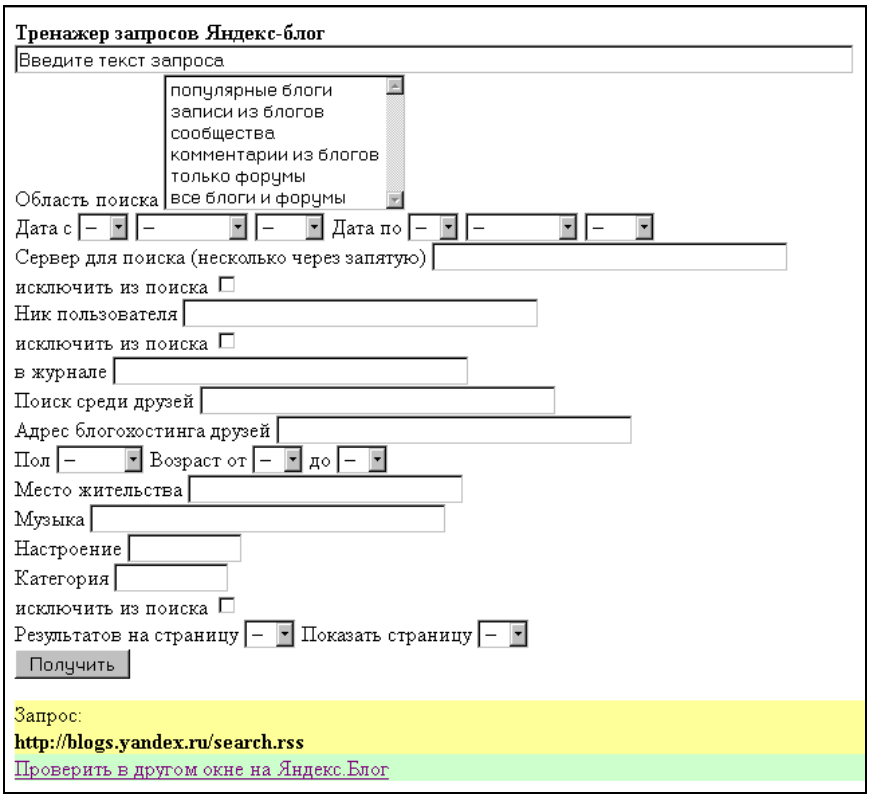


Рис. 2.8. Форма выбора параметров запроса для тренажера Яндекс.Поиск по блогам

После выбора параметров нажимаем кнопку **Получить**. Скрипт, получая данные формы, выполняет следующие действия:

1. Формирует URL запроса к API Яндекс.Поиск по блогам.
2. Выводит URL для просмотра.
3. Осуществляет запрос к сервису Яндекс.Поиск по блогам, обрабатывает результаты запроса и выводит их в удобном виде (рис. 2.9).
4. Создает ссылку на сервис Яндекс.Поиск по блогам для вывода результатов такого же запроса через оригинальный сервис для сравнения (рис. 2.10).

В ответ на запрос сервис Яндекс.Поиск по блогам выдает XML-документ. Нас интересует информация о записях, находящаяся внутри тегов `<item>`. О каждой записи мы будем выводить следующую информацию:

- заголовок записи;
- ссылку для перехода на запись;

- дату и время записи;
- описание записи;
- автора записи;
- количество комментариев.

Поэтому нам нужны будут следующие вложенные элементы тега `<item>`:

- `title` — заголовок поста;
- `link` — ссылка на пост;
- `pubDate` — дата и время публикации;
- `description` — короткое описание (обычно пусто);
- `author` — ссылка на блог/страницу автора;
- `yablogs:comments` — общее количество комментариев к записи за все время ее существования.

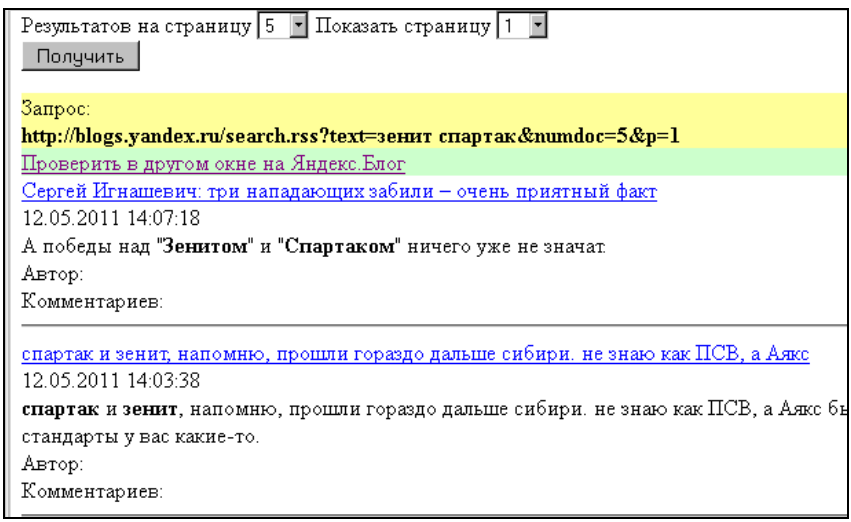


Рис. 2.9. Вывод результатов запроса через наш сервис

Рис. 2.10. Переход по ссылке запроса на сервис Яндекс.Поиск по блогам

Одним из самых простых, удобных и в то же время мощных решений для работы с XML-данными является расширение SimpleXML, которое поставляется с PHP, начиная с 5-й версии, и включено по умолчанию. С ним мы и будем работать.

Для получения нужной ветви XML-документа в SimpleXML будем использовать XPath — язык запросов к элементам XML-дерева. В классе `SimpleXMLElement` для этого имеется метод `xpath()`, возвращающий массив экземпляров класса `SimpleXMLElement` или `FALSE` в случае ошибки:

```
$xml = simplexml_load_file($url);
$records=$xml->xpath('channel/item');
```

А теперь мы можем получить доступ к нужным полям, например:

```
$author=$record->xpath('yablogs:author');
$comments=$record->xpath('yablogs:comments');
```

И выводить полученные значения в нужном виде. Так как результат возвращается кодировке UTF-8, при выводе необходима перекодировка в Windows-1251:

```
echo iconv("utf-8", "windows-1251", $record->title);
```

Вы можете посмотреть работу тренажера в Интернете, перейдя по адресу **http://examples-api.bazakatalogov.ru/yandex/yandex_blog/index_blog.php**. Весь скрипт находится в одном файле index_blog.php, который находится в папке glava_02 на компакт-диске.

2.5. API Яндекс.Фоток

Программный интерфейс к службе Яндекс.Фотки (API Яндекс.Фоток) дает возможность разработчикам создавать приложения для работы с пользовательскими данными, находящимися на сервисе Яндекс.Фотки.

API Яндекс.Фоток располагает следующими основными возможностями:

- получение, изменение, удаление фотографии;
- получение, изменение, удаление альбома;
- получение коллекции альбомов, добавление нового альбома;
- получение общей коллекции фотографий, добавление новой фотографии в общую коллекцию;
- получение коллекции фотографий некоторого альбома, добавление новой фотографии в альбом.

Подробную документацию можно посмотреть, перейдя по ссылке **http://api.yandex.ru/fotki/doc/concepts/About.xml**. А мы в качестве примера создадим небольшой проект — галерею фотографий пользователя с использованием API Яндекс.Фоток. На персональном сайте будем просматривать альбомы и фотографии пользователя, находящиеся в его профиле на сайте Яндекс.Фотки.

Для начала немного информации. API Яндекс.Фоток реализует протокол AtomPub (Atom Publishing Protocol) — универсальный протокол, предназначенный для пуб-

ликации и редактирования создаваемых пользователями данных в WWW с использованием HTTP и XML. Клиентское приложение работает с сервером API Яндекс.Фоток, посылая XML-сообщения разными HTTP-методами в соответствии с набором принципов, известным как REST: ресурсы фотографий и альбомов получаются GET-методом HTTP-запроса, публикуются или создаются POST-методом HTTP-запроса, изменяются PUT-методом HTTP-запроса и удаляются DELETE-методом HTTP-запроса.

API Яндекс.Фоток реализует AtomPub-сервис для каждого пользователя Яндекс.Фоток по адресу: **http://api-fotki.yandex.ru/api/users/*{login}*/**.

Для получения нужных данных для нашего сайта с сервиса Яндекс.Фотки необходимо отправлять GET-запросы по адресам:

- **http://api-fotki.yandex.ru/api/users/*{login}*/photos/** — для получения всех фотографий;
- **http://api-fotki.yandex.ru/api/users/*{login}*/albums/** — для получения всех альбомов пользователя;
- **http://api-fotki.yandex.ru/api/users/*{login}*/albums/*{id_альбома}*/photos** — для получения фотографий конкретного альбома.

В ответ на запрос сервер возвращает XML-документ. Для работы с данными XML-документа будем использовать расширение SimpleXML. Создадим класс `YandexFotkiGet` для формирования запросов и получения результатов при обращении к API Яндекс.Фотки. Код создания класса представлен в листинге 2.7. У класса два метода:

- `get_albums()` — для получения списка альбомов;
- `get_photos()` — для получения последних фотографий пользователя или фотографий альбома.

При разборе XML-ответа API применяется расширение SimpleXML.

Листинг 2.7

```
<?
// Класс для получения данных пользователя (альбомы, фотографии)
// через API Яндекс.Фотки
Class YandexFotkiGet{
  public $login;
  public $url = 'http://api-fotki.yandex.ru/api/users/';
  // конструктор
  function __construct($login)
  {  $this->login = $login;  }
  // возвращает массив альбомов
  public function get_albums()
  { $url=$this->url.$this->login."/albums/";
    $records=array();
```

```
    //$xml = @file_get_contents($url);      // получаем atom feed
    //$data = @simplexml_load_string($xml); // разбираем atom
    $xml = simplexml_load_file($url);
    foreach ($xml->entry as $entry)
    { $records[]=array('id' => $entry->id, 'title' => $entry->title); }
    return $records;
  }
  // возвращает массив фотографий альбома
  public function get_photos($id)
  { $url=$this->url.$this->login;
    if($id>0) $url.="/album/".$id."";
    $url.="/photos/";
    $records=array();
    $xml = simplexml_load_file($url);
    foreach ($xml->entry as $entry)
    { $records[]=array(
        'title' => iconv("utf-8", "windows-1251",$entry->title),
        'src' => iconv("utf-8", "windows-1251",$entry->content[0]
                        ->attributes()->src));
    }
    return $records;
  }
}
```

Страница примера представлена на рис. 2.11. В Интернете пример можно посмотреть по адресу **http://examples-api.bazakatalogov.ru/yandex/yandex_fotki/**.

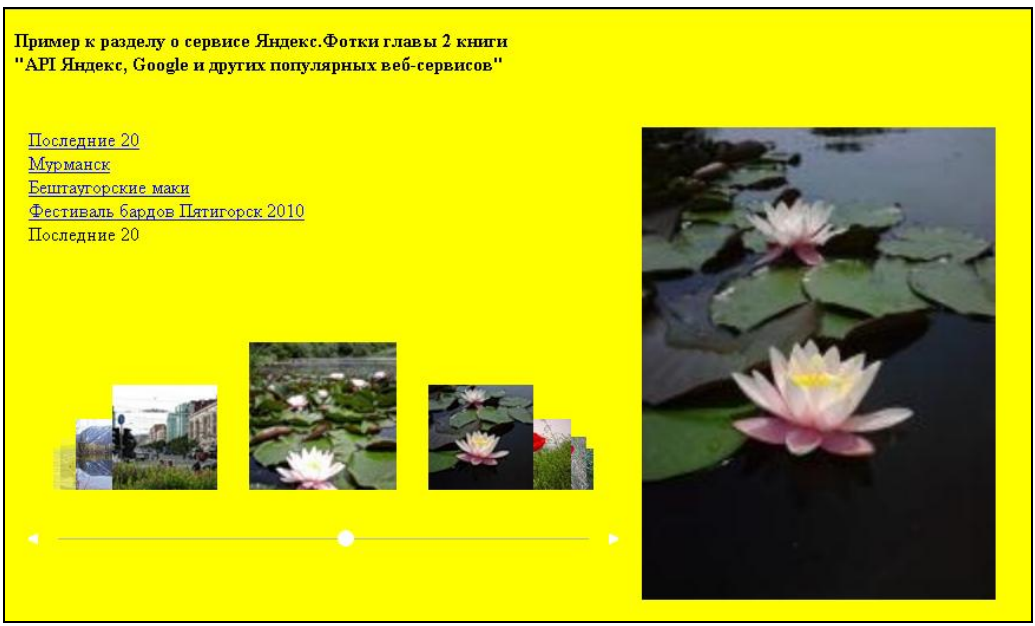


Рис. 2.11. Страница примера использования API Яндекс.Фотки

При загрузке страницы получаем список альбомов. Формируем ссылки на js-функцию загрузки содержимого конкретного альбома. При первой загрузке выводим последние фото пользователя. Создаем галерею фото с использованием js-библиотеки imageflow. Фрагмент файла index.php представлен в листинге 2.8.

Листинг 2.8

```
<?php
// подключение библиотеки xAjax
. . . . . .
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="ru" xml:lang="ru">
<head>
. . . . . .
<script type="text/javascript">
var link_tek=1;
function zagruzka()
{ var instanceOne = new ImageFlow();
  instanceOne.init(
  { ImageFlowID:'myImageFlow',reflections: false,
    reflectionP: 0.0,startID: 3,buttons: true,aspectRatio: 1.618,
    opacity: true,captions: true,imageFocusM: 1.1,scrollbarP: 0.9 });
};
function f1(arg)
{ if(arg!=link_tek)
  { xajax_View_Album(arg);d_in=document.getElementById("link"+arg);
    d_out=document.getElementById("title_album");
    d_out.innerHTML=d_in.innerHTML;link_tek=arg;}
}
</script>
<?php $xajax->printJavascript(''); ?>
</head>
<body onload='zagruzka();' style='background-color:yellow'>
  <p><b>Пример к разделу о сервисе Яндекс.Фотки главы 2 книги<br>
        "API Яндекс, Google и других популярных веб-сервисов"</b></p>
  <!-- меню — список всех альбомов -->
  <div id=albums>
    <?php
    // получение списка альбомов по api Яндекс.Фотки
    $text1="<a id=link0 href='javascript:void();'
            onclick='f1(0);' >Последние 20</a><br>";
    foreach ($fotki->get_albums() as $records)
    { $arr_id=explode(":",$records[id]);
      $id_album=$arr_id[5];
```

```
      $text1.="<a id=link".$id_album." ";
      $text1.="href='javascript:void();' ";
      $text1.="onclick='f1(".$id_album.");'>";
      $text1.=iconv("utf-8", "windows-1251",$records[title])."</a><br>";  }
    echo $text1;
    ?>
    <!-- заголовок текущей -->
    <div id="title_album">Последние 20</div>
    <!-- мини-галерея -->
    <div id="myImageFlow" class="imageflow">
      <?php
      // галерея последних 20 фото
      $images="";$i=1;
      foreach ($fotki->get_photos(0) as $records)
      { if($i==3)
        $img_src1=$records[src];
        $a1=array(".jpg",".png");$a2=array("","");
        $images.="<img src='".substr($records[src],0,-4)."XXS"."' ";
        $images.="longdesc='".$records[src]."' ";
        $images.="alt='".str_replace($a1,$a2,$records[title])."' />";
        $i++;
        if($i>20) break;  }
     echo $images;
    ?>
  </div>
  </div>
  <!-- фото крупно -->
  <div id=photo>
  <img id=photo_orig src=" <?php echo $img_src1; ?>"
       style="vertical-align:center;width:300px">
  <div>
</body>
</html>
```

При выборе другого альбома вызывается xAjax-функция `View_Album()`, в качестве аргумента передается идентификатор альбома. Функция `View_Album()` выполняет следующие действия:

- ❒ создание URL-запроса к API;
- ❒ запрос и получение XML-результата;
- ❒ обработка результата и формирование контента;
- ❒ вывод контента на страницу.

xAjax-функция `View_Album()` находится в файле view_album.php, содержимое которого представлено в листинге 2.9.

Листинг 2.9

```
<?php
function View_Album($id)
{ $objResponse = new xajaxResponse();
  // подключение файла класса
  require_once("YandexFotkiGet.php");
  // создание экземпляра
  $fotki = new YandexFotkiGet('victoruni');
  // формирование контента для галереи
  $images="";$i=1;
  foreach ($fotki->get_photos($id) as $records)
  { if($i==3) $img_src1=$records[src];
    $a1=array(".jpg",".png");$a2=array("","");
    $images.="<img src='".substr($records[src],0,-4)."XXS"."' ";
    $images.="longdesc='".$records[src]."' ";
    $images.="alt='".str_replace($a1,$a2,$records[title])."' />";
    $i++;
    if($i>20) break;
  }
  // вывод контента
  // содержимое галереи
  $objResponse->assign("myImageFlow","innerHTML",$images);
  // создание галереи
  $objResponse->script("zagruzka();");
  // вывод большого фото
  $objResponse->assign("photo_orig","src",$img_src1);
  $objResponse->assign("photo_orig","style.width","300px");
  return $objResponse;
}
?>
```

Мы рассмотрели малую часть возможностей API Яндекс.Фотки. Хотите писать клиента Яндекс.Фоток или клиентскую библиотеку, собираетесь использовать API в своем проекте или уже используете, хотите рассказать об этом, если непонятна документация, нашли проблему или способ решения, можете обратиться в клуб API Яндекс.Фоток. Адрес клуба — **http://clubs.ya.ru/api-fotki/**.

ГЛАВА 3

API Яндекс.Карт

API Яндекс.Карт — это бесплатный инструментарий, позволяющий встраивать карты Яндекса на ваш сайт. С помощью API Яндекс.Карт вы сможете управлять картами и их содержимым, а также создавать различные приложения — от простых интерактивных схем проезда к офису до сложных геоинформационных сервисов.

3.1. Как установить Яндекс.Карты на сайт

JavaScript API Яндекс.Карт — это набор классов, позволяющий разместить интерактивную карту на странице сайта, добавить на карту необходимые элементы управления и расположить географические объекты. Для того чтобы вставить карту на страницу, выполните описанные ниже шаги:

1. Получите уникальный API-ключ.
2. Загрузите API.
3. Создайте контейнер для карты.
4. Создайте карту.

3.1.1. Получение API-ключа

Чтобы иметь возможность пользоваться API Яндекс.Карт, вам необходимо получить уникальный ключ. Для этого вы должны иметь логин и пароль на Яндексе и быть авторизованным. После этого перейдите по ссылке **http://api.yandex.ru/maps/form.xml** — вы попадете на форму получения API-ключа (рис. 3.1).

Далее необходимо ввести URL вашего сайта в регистрационной форме. В качестве адреса сайта лучше всего указывать только домен (например, **mydomain.com**). Ключ, зарегистрированный на отдельный домен, будет действителен для всех URL внутри этого домена и для специальных поддоменов (например, для "www"). Ключ, зарегистрированный на домен **http://mydomain.com/**, будет действителен для:

- **http://mydomain.com/**;
- **http://www.mydomain.com/**;

- **http://www.mydomain.com/page/**;
- **http://host1.mydomain.com/**;
- **http://host2.mydomain.com/page/**.

Обратите внимание, что ключ, зарегистрированный на домен **http://www.mydomain.com/**, будет действителен только для домена **http://www.mydomain.com/** и его разделов (например, **http://www.mydomain.com/page/**).

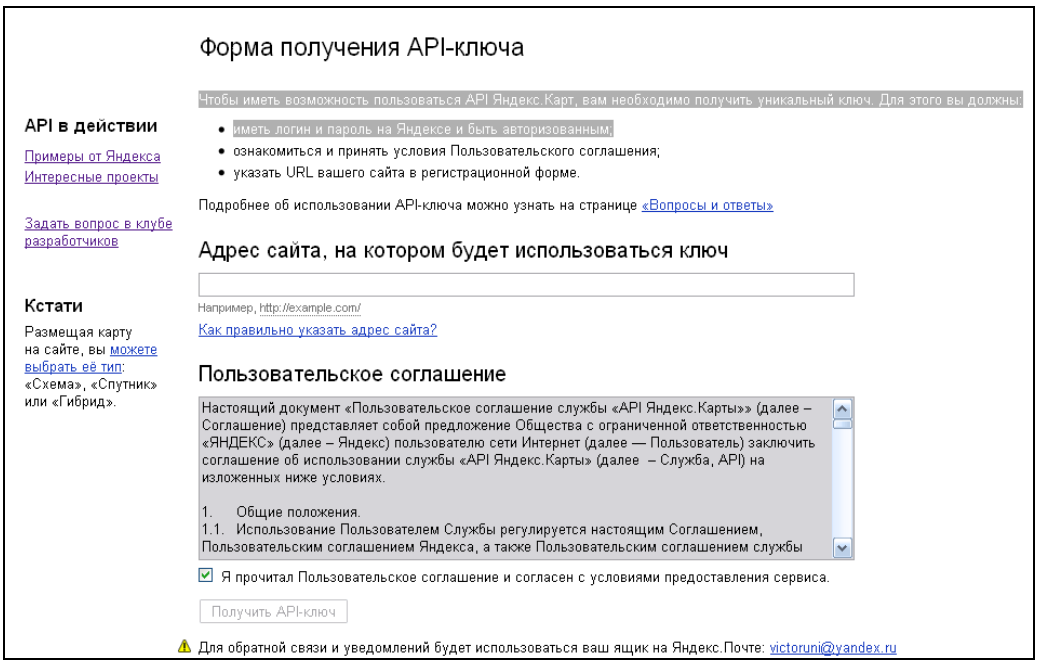


Рис. 3.1. Форма получения API-ключа

Необходимо ознакомиться с условиями пользовательского соглашения, принять их и щелкнуть по ссылке **Получить API-ключ** (рис. 3.2).

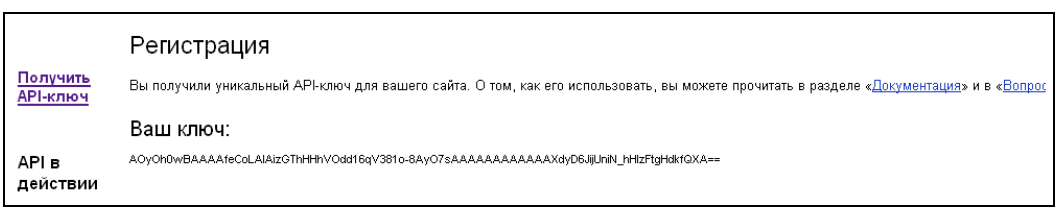


Рис. 3.2. Получение API-ключа

Полученный ключ потребуется указать в HTML-коде веб-страницы, на которую планируется добавить карту. При переходе по ссылке **Мои ключи** попадаем на страницу просмотра списка полученных ключей (рис. 3.3), где можно проверить их статус или удалить ключи, которые больше не нужны.

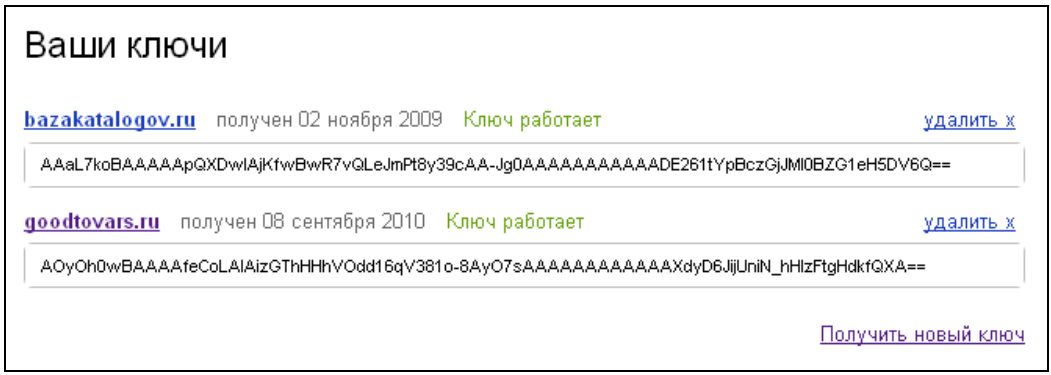


Рис. 3.3. Просмотр всех API-ключей

3.1.2. Загрузка API

Использовать методы API можно только после того, как его модули загружены в память. Загрузить API на веб-страницу можно двумя способами:

- полностью — загружается весь API целиком (используется по умолчанию);
- по требованию — загружается только специальный инициализирующий скрипт, сам API подгружается по требованию.

Если карту требуется отобразить сразу после загрузки страницы, то следует воспользоваться полной загрузкой. Если же при загрузке страницы в браузер API не используется, например, в случаях, когда карта показывается только после того, как пользователь совершит какие-либо действия на странице, то следует использовать загрузку по требованию. Для того чтобы загрузить API, необходимо добавить скрипт загрузки в заголовок (тег <head>) HTML-страницы. Пример полной загрузки приведен в листинге 3.1, загрузки по требованию — в листинге 3.2.

Листинг 3.1

```
<head>
... ... ...
<script src="http://api-maps.yandex.ru/1.1/index.xml?key=API-ключ"
        type="text/javascript"></script>
... ... ...
</head>
```

Листинг 3.2

```
<head>
... ... ...
<script src="http://api-maps.yandex.ru/1.1/index.xml?loadByRequire=1&key=API-
ключ" type="text/javascript"></script>
... ... ...
</head>
```

По окончании полной загрузки доступна вся функциональность API.

Чтобы получить возможность загружать API по требованию, а не в момент загрузки страницы, необходимо в URL скрипта загрузки добавить параметр `loadByRequire=1`. В этом случае вместо API будет загружен специальный инициализирующий скрипт, дающий возможность подгрузить API в любой момент с помощью метода `YMaps.load()`. В метод передается функция-обработчик, которая будет вызвана по окончании загрузки API. Метод `YMaps.load()` может быть вызван неограниченное количество раз. При первом вызове метода загружается API, затем вызывается обработчик. При последующих вызовах обработчик выполняется сразу же, т. е. повторная загрузка API не происходит. Например, функцию создания карты можно вызвать следующим образом:

```
YMaps.load(init);
```

Предполагается, что функция с именем `init` уже существует и в ней определен скрипт, создающий карту.

3.1.3. Создание контейнера для размещения карты

Для размещения карты на странице необходимо создать контейнер, обычно создается `div`. Его уникальный идентификатор будет использоваться как указатель на контейнер для загрузки самой карты. Обычно задаются размеры контейнера с помощью HTML-атрибута `style`.

```
<div id="YMapsID" style="width:600px;height:480px"></div>
```

3.1.4. Создание карты

Создавать карту следует после того, как веб-страница загрузится целиком. Это даст уверенность в том, что контейнер для карты создан и к нему можно обращаться по идентификатору `id`. Чтобы добавить карту на страницу, создадим обработчик события окончания загрузки страницы `onLoad` и разместим его в теге `<head>` после скрипта загрузки API (листинг 3.3).

Листинг 3.3

```
<head>
... ... ...
<script src="http://api-maps.yandex.ru/1.1/index.xml?key=API-ключ"
        type="text/javascript"></script>
<script type="text/javascript">
// вызывается по событию window.onLoad
function ini_onload()
{ // Создает экземпляр карты и привязывает его к созданному контейнеру
  map = new YMaps.Map(document.getElementById("ypmap"));
  // Устанавливает начальные параметры отображения карты:
  // центр карты и коэффициент масштабирования
```

```
  map.setCenter(new YMaps.GeoPoint(37.64, 55.76), 10);
}
</script>
</head>
<body onload="ini_onload();">
... ... ...
```

Для создания обработчика события `onLoad` и ссылки на DOM-элемент, служащий контейнером карты, удобно использовать встроенную в API библиотеку jQuery (листинг 3.4).

Листинг 3.4

```
<head>
... ... ...
<script src="http://api-maps.yandex.ru/1.1/index.xml?key=API-ключ"
        type="text/javascript"></script>
<script type="text/javascript">
<script type="text/javascript">
  // Создает обработчик события window.onLoad
  YMaps.jQuery(function()
  { // Создает экземпляр карты и привязывает его к созданному контейнеру
    var map = new YMaps.Map(YMaps.jQuery("#YMapsID")[0]);
    // Устанавливает начальные параметры отображения карты:
    // центр карты и коэффициент масштабирования
    map.setCenter(new YMaps.GeoPoint(37.64, 55.76), 10);
  })
</script>
</head>
```

> ***Примечание***
>
> Для того чтобы не определять координаты центра карты вручную, удобно пользоваться геокодированием — получением географических координат объекта по его адресу или названию *(см. разд. 3.5.1)*.

3.1.5. Удаление карты

Чтобы удалить карту, вызовите метод `destructor()` объекта карты. Например, карту можно удалить по событию `onunload`:

```
<body onunload="map.destructor();">
```

3.2. Управление картой

После того как карта добавлена на страницу, изменить ее параметры можно с помощью элементов управления. Элемент управления — это элемент пользовательского интерфейса, обеспечивающий взаимодействие пользователя с картой.

Элементы управления позволяют выполнять следующие действия:

- изменять масштаб карты;
- изменять положение центра карты;
- переключать тип карты;
- измерять расстояния между объектами на карте;
- искать объекты на карте.

API позволяет добавить на карту любой из стандартных элементов управления либо создать пользовательский элемент.

Элементы управления делятся на две группы:

- встроенные — доступны сразу после создания карты;
- внешние — необходимо самостоятельно добавлять на карту и удалять с нее.

На карте может присутствовать неограниченное количество элементов управления.

3.2.1. Встроенные элементы

Для взаимодействия с картой API предоставляет ряд встроенных элементов управления, некоторые из этих элементов включены по умолчанию.

К встроенным элементам управления относятся:

- перемещение;
- масштабирование двойным щелчком мыши;
- масштабирование колесиком мыши;
- лупа;
- "горячие" клавиши;
- линейка.

Встроенные элементы присутствуют на карте всегда, их можно включить/выключить. Рассмотрим подробнее каждый из этих методов.

Перемещение

Инструмент для перетаскивания карты мышью (по умолчанию включен). С его помощью можно перетаскивать карту, удерживая нажатой левую кнопку мыши.

Чтобы выключить перетаскивание мышью, необходимо вызвать метод объекта карты `disableDragging()`, а для того чтобы включить — метод `enableDragging()`.

Масштабирование двойным щелчком мыши

Возможность изменения масштаба карты по двойному щелчку мыши по умолчанию включена. По двойному щелчку левой кнопкой мыши масштаб увеличивается на единицу, а по двойному щелчку правой кнопкой — уменьшается на единицу.

Чтобы включить масштабирование по двойному щелчку, необходимо вызвать метод `enableDblClickZoom()`, чтобы отключить — метод `disableDblClickZoom()`.

Масштабирование колесиком мыши

Возможность изменения масштаба карты с помощью колесика мыши по умолчанию выключена.

Чтобы включить масштабирование колесиком мыши, необходимо вызвать метод `enableScrollZoom()`, а чтобы отключить — метод `disableScrollZoom()`.

Лупа

Инструмент для увеличения масштаба выделенного участка карты по умолчанию выключен. Позволяет приблизить участок карты, выделенный обводкой при нажатой кнопке мыши.

Чтобы активировать инструмент для левой кнопки мыши, необходимо вызвать метод `enableMagnifier()`, а чтобы отключить — метод `disableMagnifier()`.

Чтобы активировать инструмент для правой кнопки мыши, необходимо вызвать метод `enableRightButtonMaginfier()`, а чтобы отключить — метод `disableRightButtonMagnifier()`.

> ***Примечание***
>
> Чтобы снять выделение с участка карты, пользователю требуется нажать клавишу <Esc>.

"Горячие" клавиши

Механизм "горячих" клавиш по умолчанию выключен.

На карте поддерживаются следующие "горячие" клавиши:

- клавиши <+> и <-> — увеличение и уменьшение масштаба соответственно;
- клавиши направления (стрелки) — перемещение карты.

Чтобы включить "горячие" клавиши, необходимо вызвать метод `enableHotKeys()`, чтобы выключить — метод `disableHotKeys()`.

Линейка

Инструмент для измерения расстояний на карте по умолчанию выключен. Он позволяет с помощью мыши проложить на карте маршрут и узнать его протяженность. Маршрут указывается специальными круглыми метками, последовательно соединенными красной линией. При наведении курсора на метку отображается всплывающая подсказка, указывающая расстояние между меткой и начальной точкой маршрута. Метки можно перетаскивать, при этом расстояние от начальной точки автоматически пересчитывается. Последняя метка маршрута имеет две особенности:

- всплывающая подсказка со значением расстояния отображается всегда;
- имеется кнопка для удаления всех меток (при удалении выводится предупреждение, если количество меток больше двух).

Примечание

При двойном щелчке на метке или на всплывающей подсказке к ней метка удаляется без предупреждения.

Чтобы включить линейку, необходимо вызвать метод `enableRuler()`, а чтобы выключить — метод `disableRuler()`.

Чтобы получить строку данных о точках маршрута, необходимо вызвать метод `getRulerState()`.

Чтобы восстановить маршрут из строки данных, необходимо вызвать метод `setRulerState()`.

Расстояние рассчитывается с помощью метода `distance()` и приводится к удобочитаемому виду с помощью метода `YMaps.humanDistance()`.

3.2.2. Пример со встроенными элементами управления

Разработаем пример для динамического управления встроенными элементами на созданной карте. Пример находится на компакт-диске в папке glava_03\3-2-2. Включать/отключать элементы на карте будем без перезагрузки страницы, используя библиотеку xAjax. Посмотреть пример в действии можно по адресу **http://examples-api.bazakatalogov.ru/yandex/3-2-2/** (рис. 3.4).

При загрузке карты по умолчанию подключены следующие встроенные функции:

- ❒ перемещение;
- ❒ масштабирование двойным щелчком мыши;

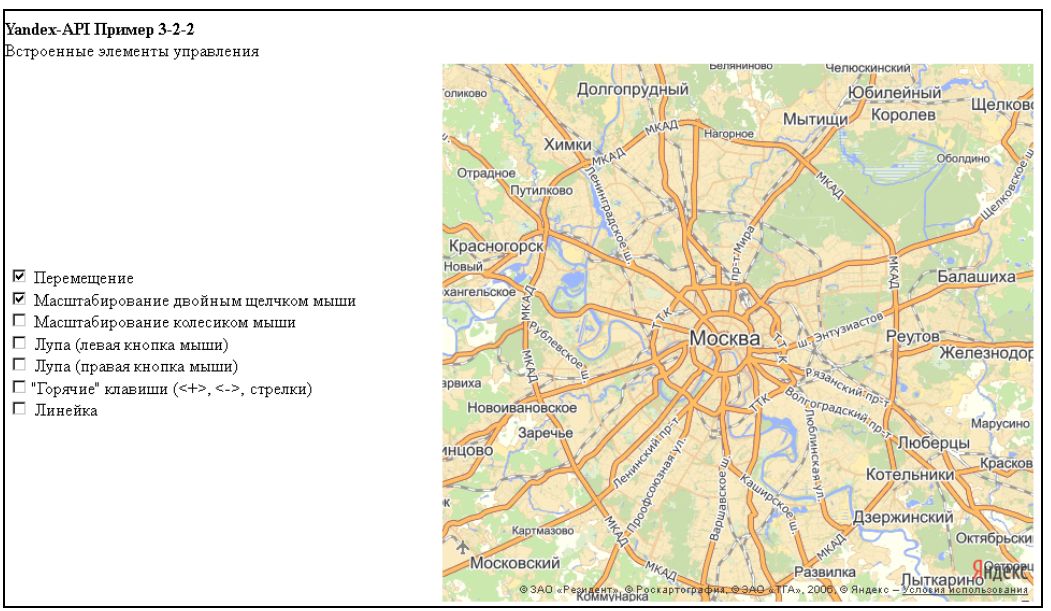


Рис. 3.4. Пример управления встроенными элементами на карте

Содержимое файла index.php представлено в листинге 3.5.

Листинг 3.5

```
... ... ...
<!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" xml:lang="ru" lang="ru">
<head>
... ... ...
<script type="text/javascript">
var map;
function ini()
{ // создание карты
  map = new YMaps.Map(document.getElementById("ymap"));
  map.setCenter(new YMaps.GeoPoint(37.64, 55.76), 10);
}
</script>
<?php $xajax->printJavascript(''); ?>
</head>
<body onload='ini();'>
    <div>
    <b>Yandex-API Пример 3-2-2</b><br>
    Встроенные элементы управления
    </div>
    <table ><tr>
    <td>
    <div id="options" style="width:400px;">
      <form id=form1 name=form1 action='javascript:void();'>
      <br><input id='elem1' name='elem1' type=checkbox checked
          onchange='xajax_Change_Option(1,xajax.getFormValues("form1"));'>
          Перемещение
      <br><input name='elem2' id='elem2' type=checkbox checked
          onchange='xajax_Change_Option(2,xajax.getFormValues("form1"));'>
          Масштабирование двойным щелчком мыши
      <br><input name='elem3' id='elem3' type=checkbox
          onchange='xajax_Change_Option(3,xajax.getFormValues("form1"));'>
          Масштабирование колесиком мыши
      ... ... ...
      </form>
    </div></td>
    <td>
    <div id="ymap" style="width:550px;height:500px"></div>
    </td>
    </tr></table>
</body>
</html>
```

При изменении состояния флажка вызывается xAjax-функция `Change_Option()`, расположенная в файле change_option.php. Содержимое файла change_option.php приведено в листинге 3.6.

Листинг 3.6

```
<?php
// $id1 — номер установленного флажка (1, 2, 3, 4, 5, 6, 7)
// $Id2 — массив значений формы
function Change_Option($id1,$Id2)
{ $objResponse = new xajaxResponse();
  $array_enable=array("",
            "map.enableDragging();","map.enableDblClickZoom();",
            "map.enableScrollZoom();","map.enableMagnifier();",
            "map.enableRightButtonMagnifier();","map.enableHotKeys();",
            "map.enableRuler();");
  $array_disable=array("",
            "map.disableDragging();","map.disableDblClickZoom();",
            "map.disableScrollZoom();","map.disableMagnifier();",
            "map.disableRightButtonMagnifier();","map.disableHotKeys();",
            "map.disableRuler();");
  if(isset($Id2['elem'.$id1])) $script=$array_enable[$id1];
  else                         $script=$array_disable[$id1];
  $objResponse->script($script);
  $objResponse->alert("js команда:\r\n".$script);
  return $objResponse;
}
?>
```

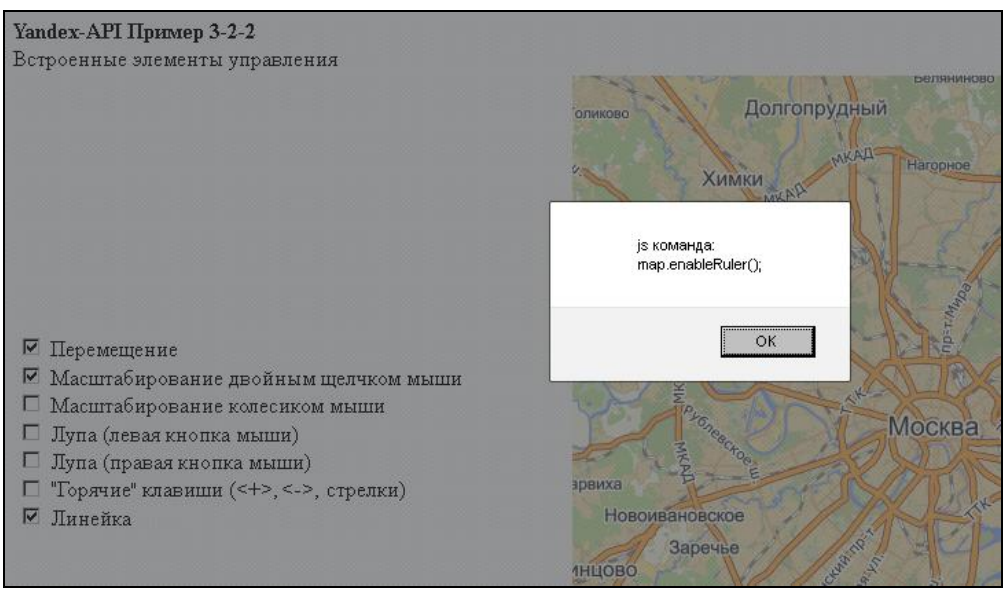


Рис. 3.5. Код JavaScript для управления встроенными элементами управления

Функция отправляет на выполнение JavaScript-код и выводит сообщение с текстом отправляемого кода (рис. 3.5).

Для использования примера на своем сайте измените константу `API_KEY` в файле my.php на значение своего ключа.

3.2.3. Внешние элементы управления

В отличие от встроенных элементов управления внешние можно добавлять или удалять с карты.

Стандартный набор внешних элементов управления картой включает:

- панель инструментов (класс `YMaps.ToolBar`) с кнопками, позволяющими перемещать карту, увеличивать ее, а также измерять расстояние на карте с помощью специальной линейки;
- элемент масштабирования (класс `YMaps.Zoom`) позволяет менять разрешение карты с определенным шагом;
- компактный элемент масштабирования (класс `YMaps.SmallZoom`) — элемент масштабирования без ползунка, содержащий только кнопки уменьшения и увеличения масштаба;
- обзорная карта (класс `YMaps.MiniMap`) — уменьшенная карта показываемой местности, масштаб которой на несколько пунктов отличается от основной;
- переключатель типа карты (класс `YMaps.TypeControl`) — кнопки **Карта**, **Спутник**, **Гибрид**;
- масштабная линейка (класс `YMaps.ScaleLine`) позволяет оценивать расстояние между объектами, не прибегая к вычислениям;
- поиск по карте (класс `YMaps.SearchControl`) позволяет искать географические объекты по их названию.

Все элементы управления основаны на интерфейсе `YMaps.IControl`. Реализовав этот интерфейс, можно создать пользовательский элемент управления.

Панель инструментов

`YMaps.ToolBar` — панель инструментов с кнопками (рис. 3.6), позволяющими перемещать карту, увеличивать ее, а также измерять расстояние на карте с помощью специальной линейки. По умолчанию панель инструментов содержит три кнопки для включения/выключения следующих элементов управления:

- **Переместить карту**;
- **Увеличить**;
- **Измерение расстояний на карте**.

Следующий код создает панель инструментов со всеми тремя кнопками по умолчанию:

```
var toolBar = new YMaps.ToolBar();
```

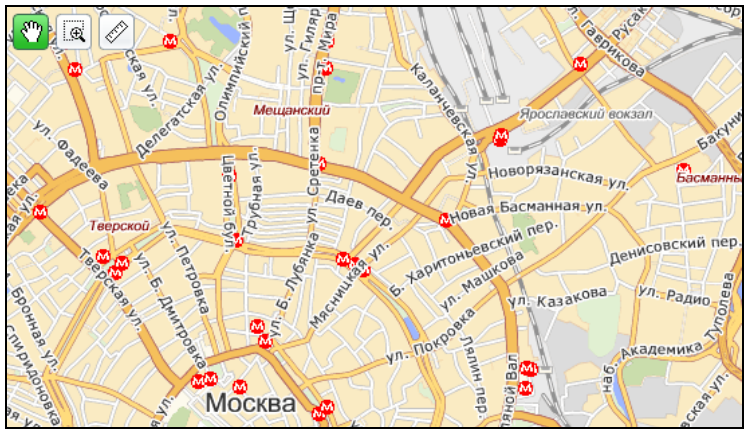


Рис. 3.6. Панель инструментов с кнопками

Если используются не все кнопки, то в конструкторе `YMaps.ToolBar` необходимо перечислить только те из них, которые должны быть добавлены на панель инструментов. Например:

```
var toolBar = new YMaps.ToolBar([
                new YMaps.ToolBar.MoveButton(),        // перемещение
                new YMaps.ToolBar.MagnifierButton(),   // лупа
                new YMaps.ToolBar. RulerButton()       // линейка
]);
```

Чтобы программно управлять состоянием кнопок панели инструментов, необходимо использовать методы `select()` и `deselect()`.

Элемент масштабирования

`YMaps.Zoom` — элемент масштабирования, позволяющий изменять масштаб карты с определенным шагом. Представляет собой две кнопки для уменьшения и увеличения масштаба и ползунок (рис. 3.7). Код для создания элемента масштабирования:

```
var zoomControl = new YMaps.Zoom();
```

По умолчанию смена масштаба происходит плавно. Чтобы отключить плавное масштабирование карты, необходимо передать в конструктор параметр `smooth` со значением `false`:

```
var zoomControl = new YMaps.Zoom({smooth: false});
```

При наведении курсора на ползунок появляется всплывающая подсказка. Чтобы отключить отображение подсказки, необходимо передать в конструктор параметр `noTips` со значением `false`:

```
var zoomControl = new YMaps.Zoom({noTips: true});
```

Для каждого коэффициента масштабирования возможно задать пользовательские всплывающие подсказки. Код для создания объекта `YMaps.Zoom` с пользовательскими подсказками для коэффициента масштабирования 1 — подсказка "Мелко", для

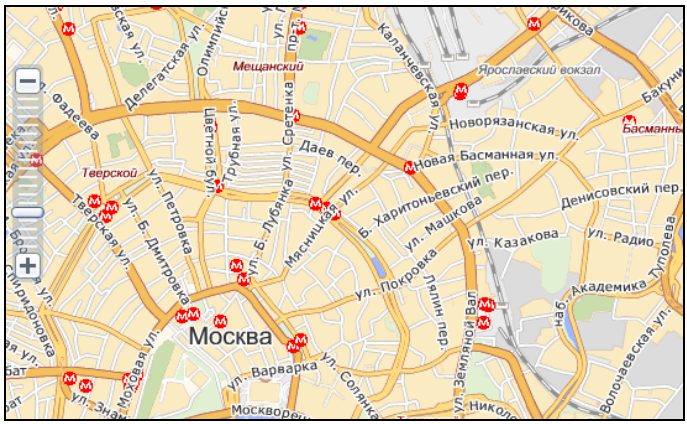


Рис. 3.7. Элемент масштабирования

коэффициента масштабирования 9 — подсказка "Средне", для коэффициента масштабирования 16 — подсказка "Крупно":

```
var zoom = new YMaps.Zoom({
  customTips: [
    { index:  1, value: "Мелко"  },
    { index:  9, value: "Средне" },
    { index: 16, value: "Крупно" }
  ]
});
```

Компактный элемент масштабирования

`YMaps.SmallZoom` — компактный элемент масштабирования, состоящий только из двух кнопок без ползунка (рис. 3.8). Код для создания компактного элемента масштабирования:

```
var smallZoomControl = new YMaps.SmallZoom();
```

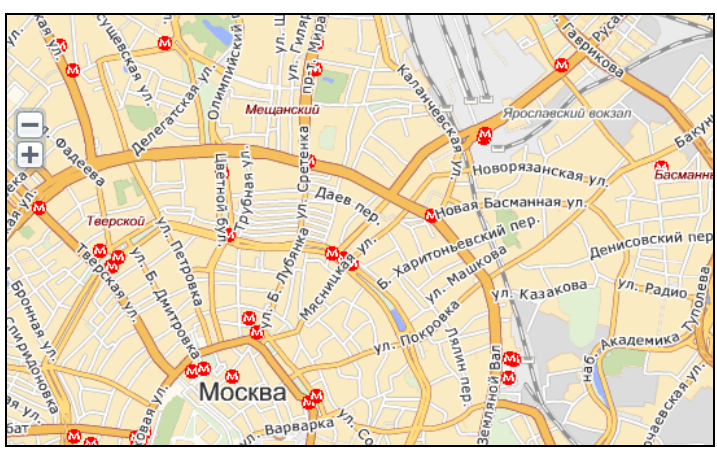


Рис. 3.8. Компактный элемент масштабирования

По умолчанию смена масштаба происходит плавно. Чтобы отключить плавное масштабирование карты, необходимо передать в конструктор параметр `smooth` со значением `false`:

```
var zoomControl = new YMaps.SmallZoom({smooth: false});
```

Обзорная карта

`YMaps.MiniMap` — элемент "обзорная карта", представляющий из себя мини-карту показываемой местности, масштаб которой на несколько пунктов меньше основной (рис. 3.9).

Чтобы задать величину смещения между коэффициентами масштабирования обзорной и основной карты, передайте необходимое значение в конструктор обзорной карты (по умолчанию 5). Если смещение положительно, то масштаб обзорной кар-

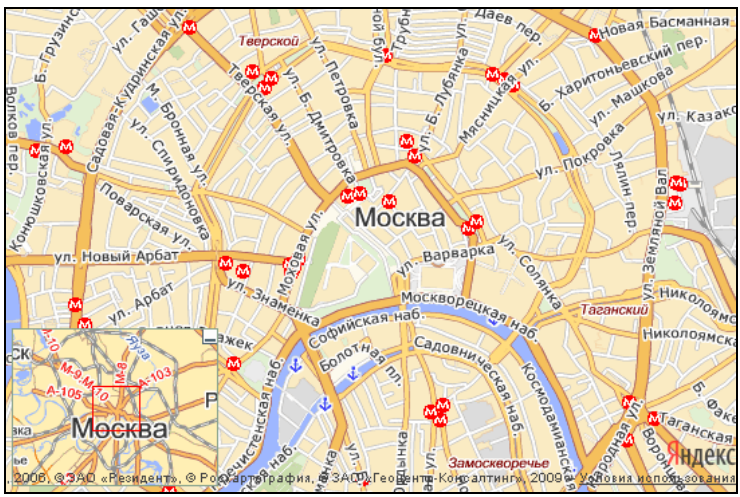


Рис. 3.9. Обзорная карта с положительным коэффициентом масштабирования

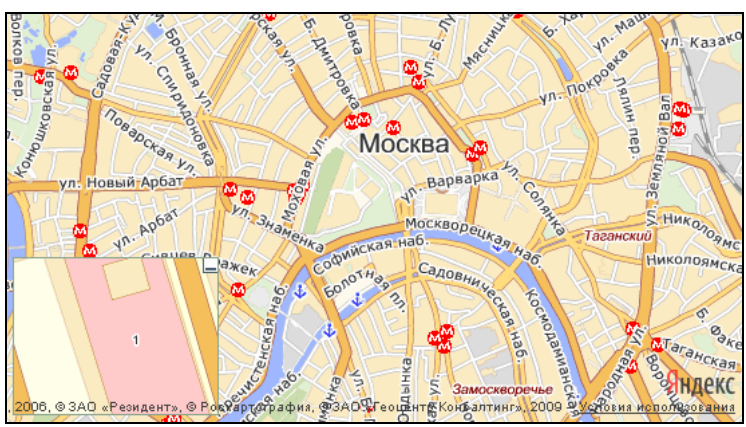


Рис. 3.10. Обзорная карта с отрицательным коэффициентом масштабирования

ты будет меньше основной, т. е. обзорная карта будет охватывать более крупный фрагмент, чем основная. Если смещение отрицательно, то обзорная карта будет работать по принципу лупы, т. е. отображать увеличенный фрагмент карты (рис. 3.10). Например:

```
var miniMapPositive = new YMaps.MiniMap(3);  // как обзорная карта
var miniMapNegative = new YMaps.MiniMap(-3); // как лупа
```

Переключатель типа карты

`YMaps.TypeControl` — элемент управления, позволяющий выбрать тип карты (например, "Схема", "Гибрид" или "Спутник"). Код для создания переключателя трех стандартных типов карт:

```
var typeControl = new YMaps.TypeControl();
```

При создании переключателя можно добавить в список пользовательские типы карт, а также указать, какие из типов карт должны быть скрыты в списке под ссылкой **Ещё** (рис. 3.11). Для этого необходимо передать в конструктор два необязательных параметра:

- `types` — список типов карт (по умолчанию добавляются три стандартных типа);
- `listTypeIndexes` — список индексов типов карты в списке, которые должны быть скрыты под ссылкой **Ещё**. Нумерация элементов списка начинается с нуля.

Код для создания переключателя для типов карты "Спутник" и "Гибрид" в раскрывающемся списке:

```
var typeControl = new YMaps.TypeControl([YMaps.MapType.MAP,
  YMaps.MapType.SATELLITE, YMaps.MapType.HYBRID], [1,2]);
map.addControl(typeControl);
```

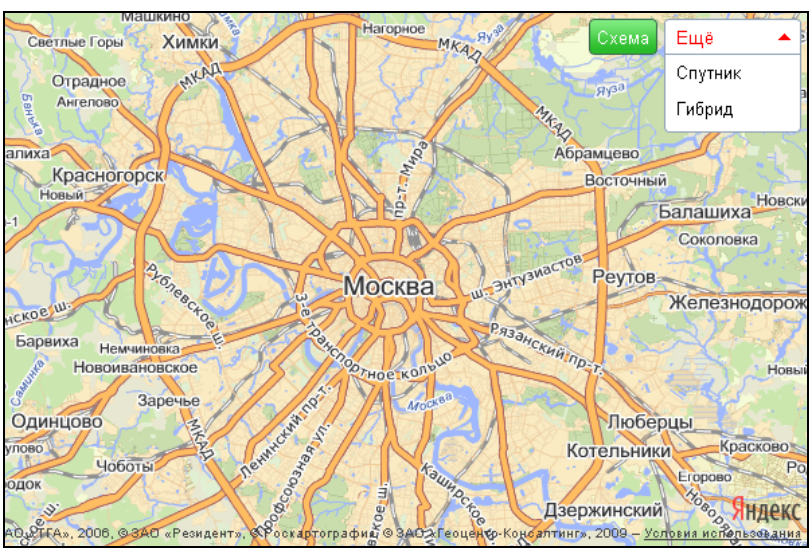


Рис. 3.11. Обзорная карта с переключателем типа карт

Масштабная линейка

`YMaps.ScaleLine` — элемент карты "масштабная линейка", позволяющий измерять расстояние между объектами, не прибегая к вычислениям (рис. 3.12). Масштабная линейка не имеет дополнительных параметров.

```
var scaleLine = new YMaps.ScaleLine();
```

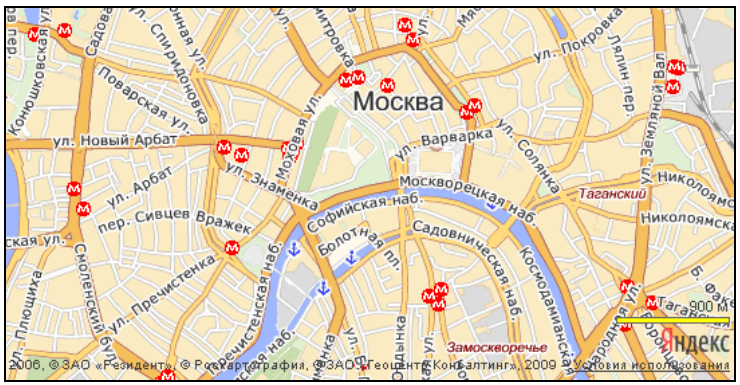


Рис. 3.12. Обзорная карта с масштабной линейкой

Поиск по карте

`YMaps.SearchControl` — элемент управления "поиск по карте", позволяющий искать на карте географические объекты по их названию или части адреса. Результаты поиска выводятся внутри самого элемента в виде списка ссылок на найденные объекты (с возможностью постраничного перелистывания) (рис. 3.13).

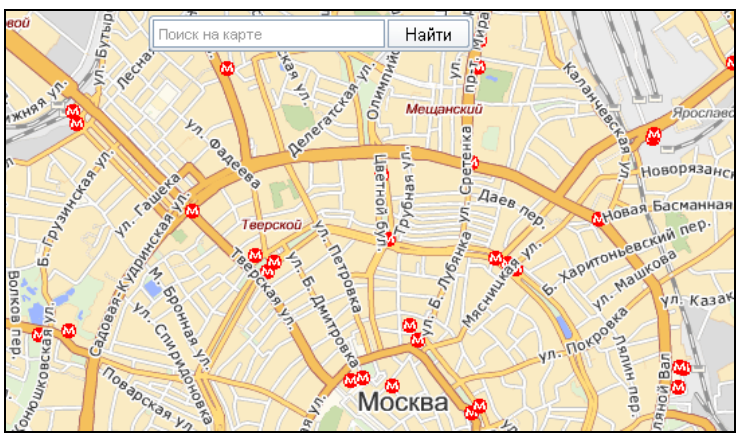


Рис. 3.13. Обзорная карта с поиском по карте

При щелчке кнопкой мыши по результату поиска на карте отображается метка с открытым пузырьком, в котором отображается адрес найденного объекта (действие по умолчанию). С помощью опций класса `YMaps.SearchControl` можно задать шири-

ну элемента управления в пикселах и количество результатов поиска на одной странице. Также возможно ограничить область поиска видимой частью карты, а также указать, требуется ли помечать меткой с открытым пузырьком найденные на карте объекты. Например:

```
var searchControl = new YMaps.SearchControl({
  resultsPerPage: 5, // Количество объектов на странице
  useMapBounds: 1    // Объекты, найденные в видимой области карты
                     // будут показаны в начале списка
});
```

3.2.4. Пример с внешними элементами управления

Создадим пример динамического управления внешними элементами на созданной карте. Пример находится на компакт-диске в папке glava_03\3-2-4. Включать/отключать элементы на карте будем без перезагрузки страницы, используя библиотеку xAjax. Посмотреть пример в действии можно по адресу **http://examples-api.bazakatalogov.ru/yandex/3-2-4/** (рис. 3.14). Содержимое файла index.php представлено в листинге 3.7.

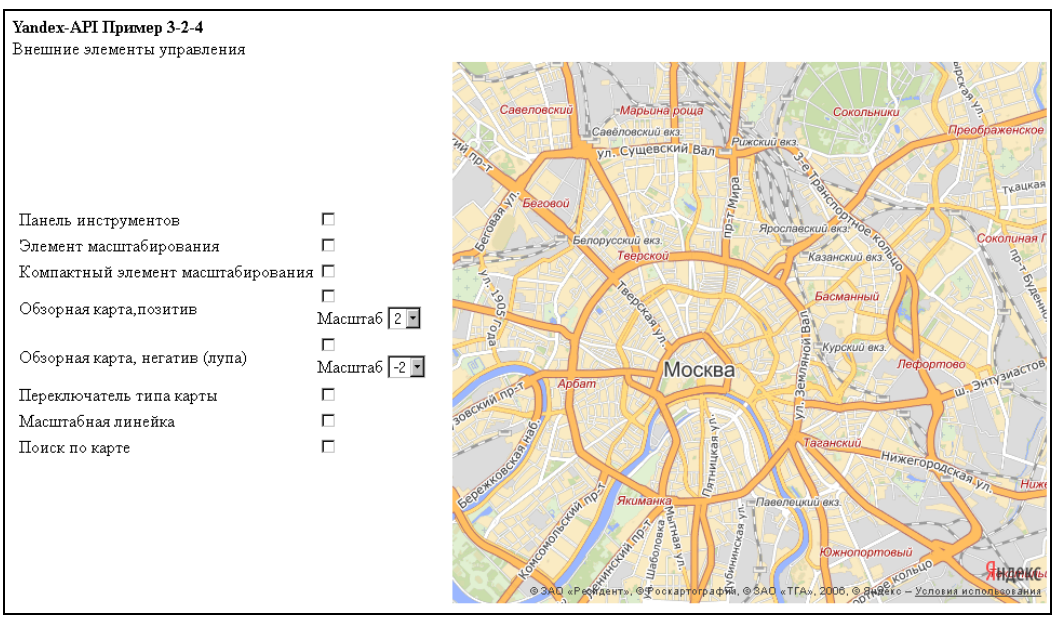


Рис. 3.14. Динамическое создание-удаление внешних элементов управления

Листинг 3.7

```
... ... ...
<!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" xml:lang="ru" lang="ru">
```

```
<head>
... ... ...
<script type="text/javascript">
var map;
// переменные глобальные для элементов управления
var toolBar;var zoomControl;var smallZoomControl;
var miniMapPositive;var miniMapNegative;var typeControl;
var scaleLine;var searchControl;
function ini()
{ // создание карты
  map = new YMaps.Map(document.getElementById("ymap"));
  map.setCenter(new YMaps.GeoPoint(37.64, 55.76), 12);
}
</script>
<?php $xajax->printJavascript(''); ?>
</head>
<body onload='ini();'>
    <div>
    <b>Yandex-API Пример 3-2-4</b><br>
    Внешние элементы управления
    </div>
    <table ><tr>
    <td>
    <div id="options" style="width:400px;">
      <form id=form1 name=form1 action='javascript:void();'>
      <table>
      <tr><td>Панель инструментов</td>
          <td><input id='elem1' name='elem1' type=checkbox
          onchange='xajax_Change_Option(1,xajax.getFormValues("form1"));'>
          </td></tr>
      <tr><td>Элемент масштабирования</td>
      <td><input name='elem2' id='elem2' type=checkbox
          onchange='xajax_Change_Option(2,xajax.getFormValues("form1"));'>
          </td></tr>
      <tr>
      <td>Компактный элемент масштабирования</td>
      <td><input name='elem3' id='elem3' type=checkbox
          onchange='xajax_Change_Option(3,xajax.getFormValues("form1"));'>
          </td></tr>
      ... ... ...
      <tr><td>Поиск по карте</td>
      <td><input name='elem9' id='elem9' type=checkbox
          onchange='xajax_Change_Option(9,xajax.getFormValues("form1"));'>
          </td></tr>
      </table>
      </form>
    </div></td>
```

```
    <td>
    <div id="ymap" style="width:550px;height:500px"></div>
    </td>
    </tr></table>
</body>
</html>
```

При изменении состояния флажка вызывается xAjax-функция `Change_Option()`, расположенная в файле change_option.php, содержимое которого приведено в листинге 3.8. Заметим также, что функция `Change_Option()` вызывается и при изменении масштаба при активных флажках обзорной карты.

Листинг 3.8

```
<?php
function Change_Option($id1,$Id2)
{ $objResponse = new xajaxResponse();
  // массив команд для включения
  $array_enable=array("",
    "toolBar=new YMaps.ToolBar();map.addControl(toolBar);",
    "zoomControl=new YMaps.Zoom();map.addControl(zoomControl);",
    "smallZoomControl=new
     YMaps.SmallZoom();map.addControl(smallZoomControl);",
    "miniMapPositive=new
     YMaps.MiniMap(%);map.addControl(miniMapPositive);",
    "miniMapNegative=new
     YMaps.MiniMap(%);map.addControl(miniMapNegative);",
    "typeControl=new YMaps.TypeControl();map.addControl(typeControl);",
    "",
    "scaleLine=new YMaps.ScaleLine();map.addControl(scaleLine);",
    "searchControl=new YMaps.SearchControl({
     resultsPerPage: 3,useMapBounds: 1 });
     map.addControl(searchControl);"
    );
  // массив команд для выключения
  $array_disable=array("",
    "map.removeControl(toolBar);",
    "map.removeControl(zoomControl);",
    "map.removeControl(smallZoomControl);",
    "map.removeControl(miniMapPositive);",
    "map.removeControl(miniMapNegative);",
    "map.removeControl(typeControl);",
    "",
    "map.removeControl(scaleLine);",
    "map.removeControl(searchControl);"
    );
```

```
  if(isset($Id2['elem'.$id1]))
  { $script=$array_enable[$id1];
    if($id1==4 || $id1==5)
      $script=str_replace("%",$Id2['elem'.$id1."1"],$script);
  }
  else  $script=$array_disable[$id1];
  $objResponse->script($script);
  $objResponse->alert("js команда:\r\n".$script);
  return $objResponse;
}
?>
```

Функция отправляет на выполнение JavaScript-код и выводит сообщение с текстом отправляемого кода (рис. 3.15).

Для использования примера на своем сайте измените константу `API_KEY` в файле my.php на значение своего ключа.

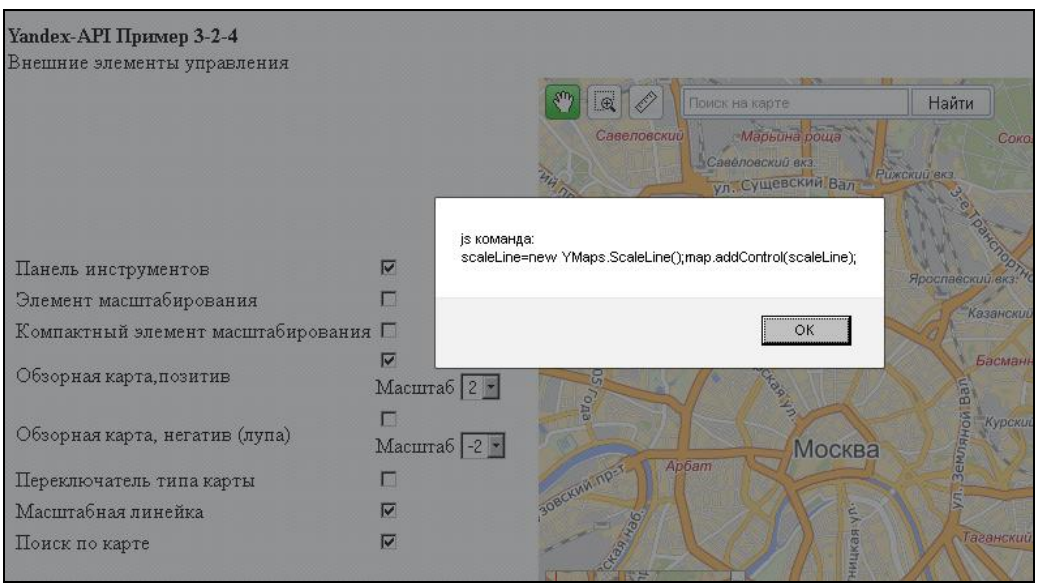


Рис. 3.15. Код JavaScript при добавлении/удалении внешних элементов управления

3.2.5. Пользовательские кнопки для панели инструментов

API Яндекс.Карт также позволяет создавать и добавлять на панель инструментов пользовательские кнопки следующих видов:

- ❒ обычная кнопка;
- ❒ переключатель;
- ❒ флажок.

Обычная кнопка

Обычная кнопка после нажатия возвращается в первоначальное состояние. Чтобы создать обычную кнопку, необходимо использовать конструктор класса `YMaps.ToolBarButton`. Пример добавления пользовательской кнопки на панель инструментов представлен в листинге 3.9.

Листинг 3.9

```
// создание панели инструментов
var toolbar = new YMaps.ToolBar();
// создание кнопки
var button = new YMaps.ToolBarButton(
  {caption: "Кнопка",hint: "Дополнительная кнопка"},
  {selected:false,enabled:true}
);
// обработчик события по нажатию кнопки
YMaps.Events.observe(button1, button1.Events.Click, function () {
  alert("Нажатие кнопки");},map);
// добавление кнопки на панель инструментов
toolbar.add(button);
// добавление панели инструментов на карту
map.addControl(toolbar);
```

Заметим, что кнопка добавляется на панель инструментов. Необходимо создать обработчик для события нажатия на кнопку. Для динамического управления кнопкой можно использовать методы объекта `YMaps.ToolBarButton` (табл. 3.1).

***Таблица 3.1.** Методы объекта `YMaps.ToolBarButton`*

Метод	Описание
`deselect()`	Устанавливает кнопку в состояние "не нажата"
`select()`	Устанавливает кнопку в состояние "нажата"
`enable()`	Устанавливает кнопку в состояние "доступна"
`disable()`	Устанавливает кнопку в состояние "недоступна"
`show()`	Показывает кнопку
`hide()`	Скрывает кнопку
`setContent(`*content*`)`	Устанавливает содержимое кнопки
`getContent()`	Возвращает содержимое кнопки
`getToolbar()`	Возвращает панель инструментов, на которую добавлена кнопка
`isSelected()`	Возвращает состояние кнопки "нажата"/"не нажата"
`isEnabled()`	Возвращает состояние кнопки "доступна"/"недоступна"
`isShown()`	Возвращает состояние видимости кнопки

Пример динамического управления кнопкой находится на компакт-диске в папке glava_03\3-2-5. Управлять параметрами пользовательской кнопки на карте будем без перезагрузки страницы, используя библиотеку xAjax. Посмотреть пример в действии можно по адресу **http://examples-api.bazakatalogov.ru/yandex/3-2-5/index1.php** (рис. 3.16). Содержимое файла index1.php представлено в листинге 3.10.

Листинг 3.10

```
... ... ...
<!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" xml:lang="ru" lang="ru">
<head>
... ... ...
<script type="text/javascript">
var map;var toolbar1;var button1;
function ini()
{ map = new YMaps.Map(document.getElementById("ymap"));
  map.setCenter(new YMaps.GeoPoint(37.64, 55.76), 10);
  // создание панели инструментов1
  toolbar1 = new YMaps.ToolBar();
  button1 = new YMaps.ToolBarButton(
    {icon:"http://api.yandex.ru/i/maps/icon-fullscreen.png",
    caption: "Кнопка", hint: "Дополнительная кнопка", width:150 },
    {selected:true, enabled:true}
  );
  // При щелчке по кнопке alert
  YMaps.Events.observe(button1, button1.Events.Click, function () {
    alert("Нажатие кнопки");},map);
  toolbar1.add(button1);
  map.addControl(toolbar1);
  }
</script>
<?php $xajax->printJavascript(''); ?>

</head>
<body onload='ini();'>
    <div>
    <b>Yandex-API Пример 3-2-5</b><br>Динамическое управление пользовательскими
       кнопками на панели инструментов (обычная кнопка)
    </div>
    <table ><tr>
    <td>
    <div id="options" style="width:400px;">
      <form id=form1 name=form1 action='javascript:void();'>
      <br><input id='elem1' name='elem1' type=checkbox checked
```

```
        onchange='xajax_Change_Option1(1,xajax.getFormValues("form1"));'>
        Доступна — Недоступна
    <br><input name='elem2' id='elem2' type=checkbox checked
        onchange='xajax_Change_Option1(2,xajax.getFormValues("form1"));'>
        Состояние Нажата — Не нажата
    <br><input name='elem3' id='elem3' type=checkbox checked
        onchange='xajax_Change_Option1(3,xajax.getFormValues("form1"));'>
        Видима — Скрыта
    <br><br>Установить контент :
    <br><input name='elem4' id='elem4' type=text
        onchange='xajax_Change_Option1(4,xajax.getFormValues("form1"));'
        value='Кнопка'>
    ... ... ...
    </form>
  </div></td>
  <td>
  <div id="ymap" style="width:550px;height:500px"></div>
  </td>
  </tr></table>
</body>
</html>
```

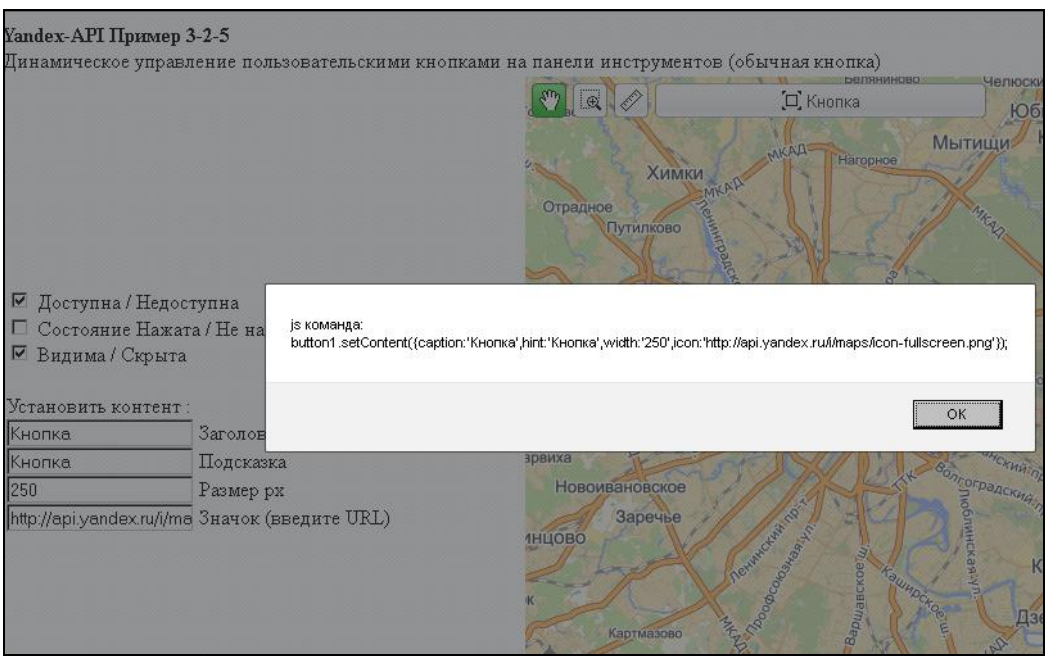


Рис. 3.16. Динамическое управление пользовательской кнопкой

При изменении состояния флажка вызывается xAjax-функция `Change_Option1()`, расположенная в файле change_option1.php, содержимое которого приведено в листинге 3.11. Обратите внимание, что кнопку в состоянии "нажата" нельзя сделать

недоступной, кнопку в состоянии "недоступна" нельзя установить в состояние "нажата", если кнопка в состоянии "скрыта", то остальные флажки не действуют.

Листинг 3.11

```
<?php
function Change_Option1($id1,$Id2)
{ $objResponse = new xajaxResponse();
  $array_enable=array("",
              "button1.enable();",
              "button1.select();",
              "button1.show();",
              "caption:'%'",
              "hint:'%'",
              "width:'%'",
              "icon:'%'");
  $array_disable=array("",
              "button1.disable();",
              "button1.deselect();",
              "button1.hide();",
              "",
              "",
              "",
              "");
  if($id1<4)
  { if(isset($Id2['elem'.$id1])) $script=$array_enable[$id1];
    else                         $script=$array_disable[$id1];
  }
  else
  { $str="";
    for($i=4;$i<8;$i++)
    { if($Id2["elem".$i]=="") $str.=$array_disable[$i];
      else
      { $str1=$array_enable[$i];
        $str1=str_replace("%",$Id2["elem".$i],$str1);
        $str.=",".$str1;
      }
    }
    $script="button1.setContent({".$str."});";
    $script=str_replace("{,","{",$script);
  }
  $objResponse->script($script);
  $objResponse->alert("js команда:\r\n".$script);
  return $objResponse;
}
?>
```

Функция отправляет на выполнение JavaScript-код и выводит сообщение с текстом отправляемого кода. Для использования примера на своем сайте измените константу `API_KEY` в файле my.php на значение своего ключа.

Переключатель

Переключатель остается нажатым до тех пор, пока не будет нажат другой переключатель из группы, в которую он входит. Переключатель всегда принадлежит какой-либо группе (но возможно создать группу из одного элемента). Группа стандартных кнопок панели инструментов имеет идентификатор, равный `YMaps.ToolBar.DEFAULT_GROUP`. Чтобы создать переключатель, необходимо использовать конструктор класса `YMaps.ToolBarRadioButton`. Пример добавления группы переключателей на панель инструментов представлен в листинге 3.12.

Листинг 3.12

```
// создание панели инструментов
var toolbar = new YMaps.ToolBar();
// создание двух кнопок одной группы id=group1
button11 = new YMaps.ToolBarRadioButton('group1', {
  caption: "Кнопка11", hint: "Кнопка11"},{selected:true,enabled:true});
button12 = new YMaps.ToolBarRadioButton('group1', {
  caption: "Кнопка12", hint:"Кнопка12"}"},{selected:false,enabled:true});
// Обработчики события по нажатию на кнопку
YMaps.Events.observe(button11, button1.Events.Select, function () {
    alert("Нажатие кнопки 11");},map);
YMaps.Events.observe(button11, button1.Events.Deselect, function () {
    alert("Deselect кнопки 11");},map);
// Добавление кнопок на панель инструментов
toolbar.add(button11);
toolbar.add(button12);
// Добавление панели инструментов на карту
map.addControl(toolbar);
```

Пример динамического управления переключателями находится на компакт-диске в папке glava_03\3-2-5. Управлять параметрами пользовательских кнопок (переключателями, входящими в две группы) на карте будем без перезагрузки страницы, используя библиотеку xajax. Посмотреть пример в действии можно по адресу **http://examples-api.bazakatalogov.ru/yandex/3-2-5/index2.php** (рис. 3.17). Содержимое файла index2.php представлено в листинге 3.13.

Листинг 3.13

```
<?php
require_once('index.common.php');
require_once('my.php');
?>
```

```
<!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" xml:lang="ru" lang="ru">
<head>
<title>Yandex-API Пример 3-2-5 </title>
<script src="http://api-maps.yandex.ru/1.1/index.xml?key=<?php echo API_KEY;
?>" type="text/javascript"></script>
<script type="text/javascript">
var map;
var toolbar1;
var button11;
var button12;
var button21;
var button22;
var button32;
function ini()
{ map = new YMaps.Map(document.getElementById("ymap"));
  map.setCenter(new YMaps.GeoPoint(37.64, 55.76), 10);
  // создание панели инструментов1
  toolbar1 = new YMaps.ToolBar();
  // группа1
  button11 = new YMaps.ToolBarRadioButton('group1', {
    caption: "Кнопка11", hint: "Кнопка11"},{selected:true});
  button12 = new YMaps.ToolBarRadioButton('group1', {
    caption: "Кнопка12", hint: "Кнопка12"});
  // группа2
  button21 = new YMaps.ToolBarRadioButton('group2', {
    icon: "http://some.url/path/img.gif", hint: "Кнопка21"},{selected:true});
  button22 = new YMaps.ToolBarRadioButton('group2', {
    icon: "http://api.yandex.ru/i/maps/icon-fullscreen.png", hint:
"Кнопка22"});
  button32 = new YMaps.ToolBarRadioButton('group2', {
    icon: "http://api-maps.yandex.ru/i/0.3/icons/icon-ruler.png", hint:
"Кнопка32"});
  //////// добавление
  // Создает разделитель на панели инструментов шириной 30 пикселов
  toolbar1.add(new YMaps.ToolBarSeparator(30));
  toolbar1.add(button11);
  toolbar1.add(button12);
  // Создает разделитель на панели инструментов шириной 20 пикселов
  toolbar1.add(new YMaps.ToolBarSeparator(20));
  toolbar1.add(button21);
  toolbar1.add(button22);
  toolbar1.add(button32);
  // добавить на карту
  map.addControl(toolbar1);
}
</script>
```

```
<?php $xajax->printJavascript(''); ?>
</head>
<body onload='ini();'>
    <div>
    <b>Yandex-API Пример 3-2-5</b><br>
    Пользовательские кнопки на панели инструментов (переключатель)
    </div>
    <table ><tr>
    <td>
    <div id="options" style="width:400px;">
      <form id=form1 name=form1 action='javascript:void();'>
      <!-- группа1 -->
      <br> <b> Кнопка11 </b>
      <br><input name='elem11' type=radio checked
          onchange='xajax_Change_Option2(1,xajax.getFormValues("form1"));'>
          Нажата/не нажата
      <input name='elem2' id='elem2' type=checkbox checked
          onchange='xajax_Change_Option2(2,xajax.getFormValues("form1"));'>
          Доступна/недоступна
      <br> <b> Кнопка12 </b>
      <br><input name='elem11' type=radio
          onchange='xajax_Change_Option2(3,xajax.getFormValues("form1"));'>
          Нажата/не нажата
      <input name='elem4' id='elem4' type=checkbox checked
          onchange='xajax_Change_Option2(4,xajax.getFormValues("form1"));'>
          Доступна/недоступна
      <!-- группа2 -->
      <br> <b> Кнопка21 </b>
      <br><input name='elem21' type=radio checked
          onchange='xajax_Change_Option2(5,xajax.getFormValues("form1"));'>
          Нажата/не нажата
      <input name='elem6' id='elem6' type=checkbox checked
          onchange='xajax_Change_Option2(6,xajax.getFormValues("form1"));'>
          Доступна/недоступна
      <br> <b> Кнопка22 </b>
      <br><input name='elem21' type=radio
          onchange='xajax_Change_Option2(7,xajax.getFormValues("form1"));'>
          Нажата/не нажата
      <input name='elem8' id='elem8' type=checkbox checked
          onchange='xajax_Change_Option2(8,xajax.getFormValues("form1"));'>
          Доступна/недоступна
      <br> <b> Кнопка32 </b>
      <br><input name='elem21' type=radio
          onchange='xajax_Change_Option2(9,xajax.getFormValues("form1"));'>
          Нажата/не нажата
      <input name='elem10' id='elem10' type=checkbox checked
          onchange='xajax_Change_Option2(10,xajax.getFormValues("form1"));'>
          Доступна/недоступна
```

```
      </form>
    </div></td>
    <td>
    <div id="ymap" style="width:550px;height:500px"></div>
    </td>
    </tr></table>
</body>
</html>
```

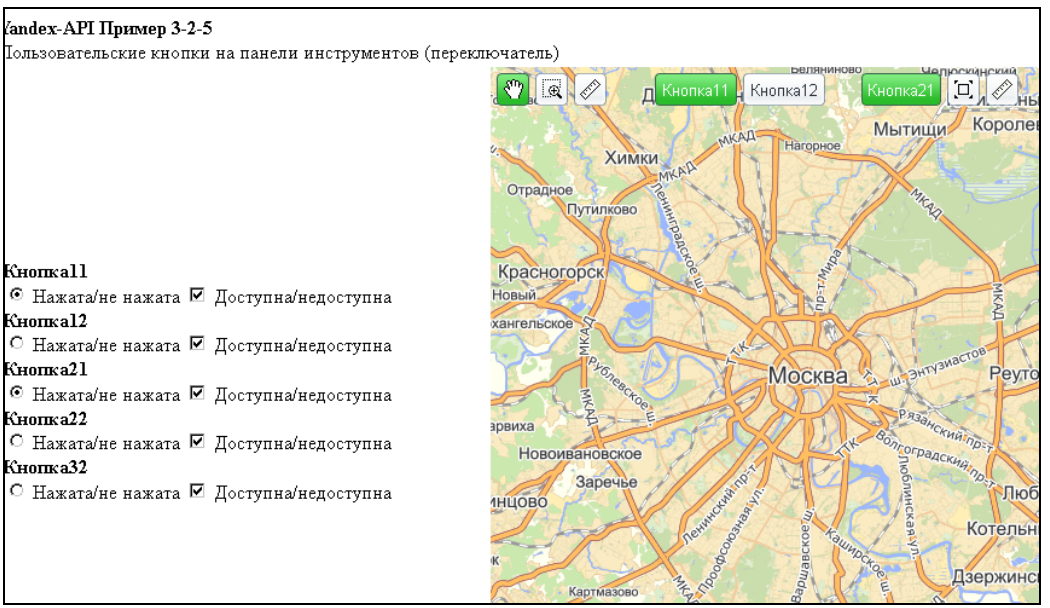


Рис. 3.17. Динамическое управление параметрами переключателей

При изменении состояния флажка или переключателя вызывается xAjax-функция `Change_Option2()`, расположенная в файле change_option2.php, содержимое которого приведено в листинге 3.14. Обратите внимание, что кнопку в состоянии "нажата" нельзя сделать недоступной, кнопку в состоянии "недоступна" нельзя установить в состояние "нажата".

Листинг 3.14

```
<?php
function Change_Option2($id1,$Id2)
{ $objResponse = new xajaxResponse();
  $array_enable=array("",
              "button11.select();",
              "button11.enable();",
              "button12.select();",
              "button12.enable();",
```

```
            "button21.select();",
            "button21.enable();;",
            "button22.select();",
            "button22.enable();",
            "button32.select();",
            "button32.enable();");
  $array_disable=array("",
            "button11.select();",
            "button11.disable();",
            "button12.select();",
            "button12.disable();",
            "button21.select();",
            "button21.disable();;",
            "button22.select();",
            "button22.disable();",
            "button32.select();",
            "button32.disable();");
  if(isset($Id2['elem'.$id1]))  $script=$array_enable[$id1];
  else                          $script=$array_disable[$id1];
  $objResponse->script($script);
  $objResponse->alert("js команда:\r\n".$script);
  return $objResponse;
}
?>
```

Функция отправляет на выполнение JavaScript-код и выводит сообщение с текстом отправляемого кода. Для использования примера на своем сайте измените константу `API_KEY` в файле my.php на значение своего ключа.

Флажок

Флажок остается установленным до тех пор, пока не будет сброшен. Чтобы создать флажок, необходимо использовать конструктор класса `YMaps.ToolBarToggleButton`. Пример добавления флажка на панель инструментов представлен в листинге 3.15.

Листинг 3.15

```
// создание панели инструментов
var toolbar = new YMaps.ToolBar();
// создание флажка
var button1 = new YMaps.ToolBarToggleButton({
  icon:" /i/maps/icon-fullscreen.png",
  caption: "Флажок",
  hint: " Кнопка-флажок "}
  {selected:true,enabled:true});
YMaps.Events.observe(button1, button1.Events.Select, function () {
  alert("Нажатие2")}, toolbar);
```

```
YMaps.Events.observe(button1, button1.Events.Deselect, function () {
  alert("Нажатие2");}, toolbar);
// Добавление на панель инструментов
toolbar.add(button11);
// добавление панели инструментов на карту
map.addControl(toolbar);
```

Разделитель на панели инструментов

Разделитель на панели инструментов создается, чтобы имитировать расстояние между двумя кнопками. Для разделителя необходимо использовать конструктор класса `YMaps.ToolBarSeparator`. Пример добавления разделителя между двумя кнопками представлен в листинге 3.16.

Листинг 3.16

```
toolbar.add(button11);
// Создает разделитель панели инструментов шириной 30 пикселов
toolBar.add(new YMaps.ToolBarSeparator(30));
toolbar.add(button12);
```

3.2.6. Создание пользовательских элементов управления

Для того чтобы создать пользовательский элемент управления, используйте класс `YMaps.IControl`. Создайте пользовательский класс, в котором будут определены два метода:

- `onAddToMap()` — вызывается при добавлении элемента управления на карту;
- `onRemoveFromMap()` — вызывается при удалении элемента управления с карты.

При добавлении элемента управления на карту в метод `onAddToMap()` передается указатель на карту и положение элемента управления (если задано). Пример создания нового элемента управления представлен в листинге 3.17.

Листинг 3.17

```
map = new YMaps.Map(document.getElementById("ymap"));
map.addControl(new NewControl ());
function NewControl ()
{ this.onAddTomMap = function (map, controlPosition)
  {  // Действия при добавлении элемента управления на карту };
  this.onRemoveFromMap = function ()
  {  // Действия при удалении элемента управления с карты  };
}
```

При добавлении элемента управления на карту в метод `onAddToMap()` передается указатель на карту и положение элемента управления (если задано). В качестве

примера создадим пользовательский элемент управления. Размещая маркеры на карте, вы можете столкнуться с тем, что на ней окажется очень много маркеров (меток) и они будут закрывать друг друга. Один из способов решения данной проблемы заключается в разбиении маркеров на группы, которые можно отображать и убирать с карты по мере необходимости. Для этого в API Яндекс.Карт есть специальный класс `YMaps.Group`. Он позволяет объединять в группы объекты любого типа. Наш пользовательский элемент будет выполнен в виде раскрывающегося списка и играть роль фильтра для отображения меток определенной группы либо всех меток. При изменении значения `select` вызывается js-функция, сначала удаляющая все маркеры, затем отображающая маркеры выбранной группы. Пример создания пользовательского элемента управления находится на компакт-диске в папке glava_03\3-2-5. Посмотреть пример в действии можно по адресу **http://examples-api.bazakatalogov.ru/yandex/3-2-5/index3.php** (рис. 3.18). Содержимое файла index3.php представлено в листинге 3.18.

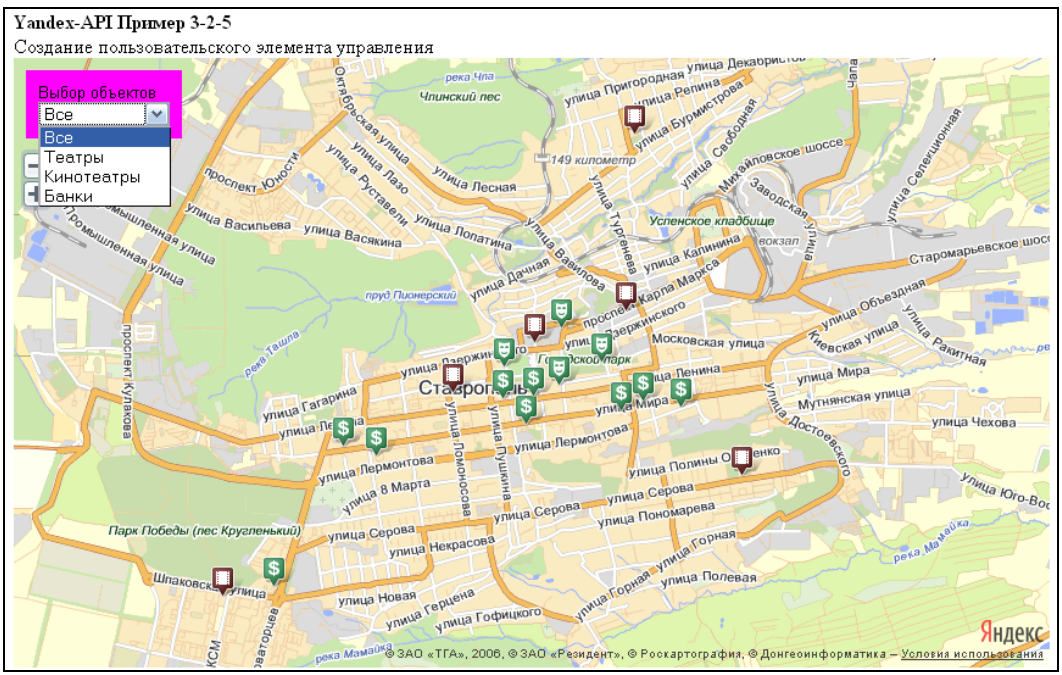


Рис. 3.18. Создание пользовательского элемента управления

Листинг 3.18

```
... ... ...
<!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" xml:lang="ru" lang="ru">
<head>
<title>Yandex-API Пример 3-2-5 </title>
```

```
<script src="http://api-maps.yandex.ru/1.1/index.xml?key=<?php echo API_KEY;
?>" type="text/javascript"></script>
<script type="text/javascript">
var map;var smallZoomControl;var group=new Array();
function ini()
{ // создание карты
  map = new YMaps.Map(document.getElementById("ymap"));
  map.setCenter(new YMaps.GeoPoint(41.96509425, 45.04268975), 12);
  // элемент масштабирование
  smallZoomControl=new YMaps.SmallZoom({smooth:false});
  map.addControl(smallZoomControl);
  ////// создание меток по группам
  // театры
  group[1] = new YMaps.GeoObjectCollection("default#theaterIcon");
  group[1].add(createPlacemark(new YMaps.GeoPoint(41.96509425, 45.04268975),
'Академический театр драмы им. Лермонтова', 'Ставрополь пл. Ленина 1'));
... ... ...
  map.addOverlay(group[1]);
  // театры
  group[2] = new YMaps.GeoObjectCollection("default#cinemaIcon");
  group[2].add(createPlacemark(new YMaps.GeoPoint(41.96940724, 45.04515414),
'Атлантис', 'Ставрополь ул. Маршала Жукова 8'));
... ... ...
  map.addOverlay(group[2]);
  // банки
  group[3] = new YMaps.GeoObjectCollection("default#bankIcon");
  group[3].add(createPlacemark(new YMaps.GeoPoint(41.92636315, 45.00475977),
'БИН банк', 'Ставрополь ул. Доваторцев 61'));
... ... ...
  map.addOverlay(group[3]);
  // Функия создания метки
  function createPlacemark (geoPoint, name, description)
  { var placemark = new YMaps.Placemark(geoPoint);
    placemark.name = name;
    placemark.description = description;
    return placemark;
  }
  // создание пользовательского элемента
  map.addControl(new Choice_Objects(group));
  // eghfdkz.obq 'ktvtyn'
  function Choice_Objects(groups)
  { // Добавление пользовательского элемента на карту  }
  ... ... ...
  function view_choice(arg)
  { map.removeAllOverlays();
    if(arg==0)
    { for(var j=1;j<=group.length;j++)
```

```
        map.addOverlay(group[j]);
    }
    else map.addOverlay(group[arg]);
  }
</script>
</head>
<body onload='ini();'>
    <div>
    <b>Yandex-API Пример 3-2-5</b><br>
    Создание пользовательского элемента управления
    </div>
    <div id="ymap" style="width:850px;height:500px"></div>
</body>
</html>
```

3.3. События

API предоставляет собственный интерфейс для обработки и создания обработчиков событий карты. Например, для карты можно добавить дополнительные пользовательские действия, которые будут срабатывать при перемещении карты, смене масштаба, типа карты и т. д. События API являются самостоятельными и отличаются от стандартных DOM-событий браузера. У многих объектов API набор событий индивидуален. Полный список событий для каждого объекта приведен в справочнике по программному интерфейсу (см. **http://api.yandex.ru/maps/jsapi/doc/ref/concepts/About.xml**).

3.3.1. Обработчики событий

Объект `YMaps.Events` позволяет подключать пользовательские обработчики событий к объектам API и вызывать пользовательские функции при возникновении определенного события (щелчок, двойной щелчок, перемещение мыши и т. д.). Каждое событие происходит в собственном контексте и позволяет оперировать только данными, доступными из этого контекста. Например, событие `MouseMove` возвращает долготу и широту географического положения той точки на карте, на которую указывает курсор мыши.

3.3.2. Подключение обработчика событий

Чтобы создать и подключить обработчик события, необходимо использовать метод `YMaps.Events.observe()`. Метод принимает на вход следующие параметры:

- объект, на котором возникает событие;
- событие, которое необходимо обрабатывать;
- callback-функцию обработки события;
- контекст выполнения callback-функции (по умолчанию совпадает с объектом, на котором возникает событие);

- флаг "начать обрабатывать событие немедленно" (по умолчанию установлен в `true`).

Пример создания обработчика события по щелчку кнопкой мыши на карте представлен в листинге 3.19.

Листинг 3.19

```
map = new YMaps.Map(document.getElementById("ymap"));
var event1=YMaps.Events.observe(map,map.Events.Click,
  function ()
  { alert("Долгота="+mEvent.getGeoPoint().getLng()+
          "Широта\r\n="+mEvent.getGeoPoint().getLat());
  }, map,true);
```

3.3.3. Удаление обработчика событий

Если обработчик события больше не требуется (например, если обрабатываемое событие должно произойти однократно), его можно удалить, вызвав метод `cleanup()`. Для использования метода необходимо, чтобы обработчики имели уникальные имена. Пример создания обработчика и удаления его после однократного действия (помещения метки) представлен в листинге 3.20.

Листинг 3.20

```
var event1=YMaps.Events.observe(map,map.Events.Click,
  function ()
  { var placemark = new YMaps.Placemark(mEvent.getGeoPoint());
    map.addOverlay(placemark);
    event1.cleanup();
  }, map);
```

3.3.4. Включение/выключение обработчика событий

Чтобы выключить обработчик, используйте метод `disable()`, чтобы включить — `enable()`.

3.3.5. Инициирование события

Чтобы инициировать обработчик события, необходимо использовать метод `YMaps.Events.notify()`. Метод принимает на вход следующие параметры:

- объект, на котором инициируется событие;
- тип события;
- данные, передаваемые функции обработчика события.

Пример инициирования события представлен в листинге 3.21.

Листинг 3.21

```
map = new YMaps.Map(document.getElementById("ymap"));
YMaps.Events.observe(map, "Event1", function (par) {
   alert(par);
});
YMaps.Events.notify(map, "Event1",par);
```

3.4. Объекты-оверлеи на карте

API Яндекс.Карт позволяет размещать на картах визуальные объекты. При использовании JavaScript API на карте можно разместить значки с текстом (метки), балун (balloon), всплывающие подсказки, линии, многоугольники, а также определить собственные визуальные объекты. В случае использования HTTP API на карту можно поместить значки с текстом, линии и многоугольники.

Визуальные объекты, размещаемые поверх карты и привязанные к координатам местности, называются *оверлеями*. Если объект размещается на карте, но к координатам не привязан, он называется *псевдооверлеем*. API Яндекс.Карт поддерживает четыре типа оверлеев: балун (balloon), метка, ломаная, многоугольник и один псевдооверлей — всплывающая подсказка.

Так как оверлеи привязаны к координатам, то при сдвиге карты они будут сдвигаться вместе с картой. При изменении масштаба карты те оверлеи, которые привязаны к нескольким координатам (ломаная и многоульник), также будут изменять свой размер. Псевдооверлей (всплывающая подсказка) не реагирует ни на смещение карты, ни на изменение ее масштаба.

3.4.1. Балун

Балун — это всплывающее окно, в котором может быть показано любое HTML-содержимое (рис. 3.19).

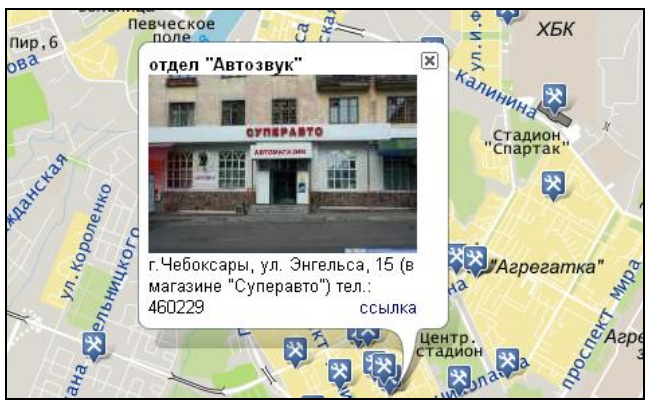


Рис. 3.19. Балун

Одновременно на карте может быть показан только один балун (экземпляр класса `YMaps.Balloon`). Для объекта `YMaps.Balloon` нет необходимости вызывать конструктор, он создается автоматически при вызове конструктора карты и существует в единственном экземпляре. На карте может быть только один балун и все, что с ним можно сделать, — это переместить его в другое место, изменить его содержимое или внешний вид. Для того чтобы показать балун, необходимо использовать метод карты `openBalloon()`, на вход которого передаются координаты точки, где должен располагаться "хвостик" балуна и задается HTML-содержимое балуна. Например, чтобы разместить балун с надписью "Это Балун" в центре карты, используем следующий код:

```
map.openBalloon(new YMaps.GeoPoint(37.64, 55.76), " Это Балун ");
```

Также балун можно показать для любых объектов-оверлеев, добавленных на карту. В листинге 3.22 представлен пример создания метки и открытия балуна для этой метки.

Листинг 3.22

```
// Создает метку и добавляет ее на карту
var placemark = new YMaps.Placemark(map.getCenter());
placemark.name = "Имя метки";
placemark.description = "Описание метки";
map.addOverlay(placemark);
// Откpыает балун
placemark.openBalloon();
```

Параметры балуна

Положение балуна на карте, его поведение при перетаскивании карты и внешний вид можно задать с помощью параметров балуна:

- `hasCloseBalloon` — флаг, указывающий, должна ли быть у балуна кнопка закрытия (по умолчанию установлен в значение `true`);
- `mapAutoPan` — флаг, указывающий, что при открытии балуна требуется автоматически сдвигать карту так, чтобы балун был виден целиком (по умолчанию установлен в значение `true`);
- `margin` — величина минимального отступа балуна от границ карты (в пикселах);
- `maxHeight` и `maxWidth` — максимальная высота и ширина балуна (в пикселах);
- `style` — стиль балуна.

В листинге 3.23 приведены примеры создания балуна с различными параметрами.

Листинг 3.23

```
// Отсутствует кнопка закрытия
map.openBalloon(map.getCenter(),text, { hasCloseBalloon: false});
```

```
// При открытии балуна сдвигать карту так, чтобы балун был виден целиком
map.openBalloon(map.getCenter(),text, { mapAutoPan: true});
// Отступ 10 пикселов от всех границ карты
map.openBalloon(map.getCenter(),text, {margin: 10});
// Отступ от верхней границы карты 10, от правой — 20, от нижней – 30,
// от левой — 40 пикселов
map.openBalloon(map.getCenter(),text, {margin: [10, 20, 30, 40]});
// Установка максимальной ширины 100 px и высоты 200 px
map.openBalloon(map.getCenter(),text, { maxHeight :200, maxWidth: 100});
```

Установка содержимого балуна

Содержимое балуна, открытого над объектом-оверлеем, форматируется в HTML с помощью шаблона. По умолчанию шаблон содержимого балуна имеет следующий вид:

```
<b>$[name]</b><div>$[description]</div>
```

Поля `$[name]` и `$[description]` заменяются соответствующими значениями из полей `name` и `description` объекта-оверлея. В листинге 3.24 представлен пример, где создается метка, у которой заполнены поля `name` и `description`.

Листинг 3.24

```
var placemark = new YMaps.Placemark(new YMaps.GeoPoint(37.64, 55.76));
placemark.name = "Имя метки";
placemark.description = "Описание метки";
map.addOverlay(placemark);
```

Если у этой метки открыть балун, то его содержимое будет таким:

```
<b>Имя метки</b><div>Описание метки</div>
```

Содержимым балуна могут быть любые HTML-конструкции. Чтобы поместить HTML-код в балун, необходимо использовать метод `setBalloonContent()`, а чтобы получить его — метод `getBalloonContent()`. В листинге 3.25 представлен пример создания содержимого балуна с помощью метода `setBalloonContent()`.

Листинг 3.25

```
var placemark = new YMaps.Placemark(map.getCenter());
// Контент для балуна
var content="<div>Изображение объекта<br><img src='img_obj.img'></div>";
// Задает содержимое балуна
placemark.setBalloonContent(content);
map.addOverlay(placemark);
// Открывает балун с надписью
placemark.balloonOpen();
```

Примечание

Содержимое, помещенное в балун с помощью метода `setBalloonContent()`, имеет приоритет перед содержимым, созданным с помощью наложения текущего стиля содержимого балуна.

Задание стиля для содержимого балуна

По умолчанию шаблон содержимого балуна имеет следующий вид:

```
<b>$[name]</b><div>$[description]</div>
```

Для того чтобы изменить стиль отображения содержимого балуна на карте, используйте класс `YMaps.BalloonContentStyle`. В листинге 3.26 представлен пример создания шаблона для содержимого балуна, связывание экземпляра класса `YMaps.BalloonContentStyle` с созданным шаблоном и применение полученного стиля к метке.

Листинг 3.26

```
var s = new YMaps.Style();
s.balloonContentStyle = new YMaps.BalloonContentStyle(
new YMaps.Template("<div style=\"color:#0A0\">$[description]</div>
<div><b>[name]</b></div>"));
var placemark = new YMaps.Placemark(map.getCenter(), {style: s} );
placemark.description = "Добро пожаловать на Яндекс.Карты!";
map.addOverlay(placemark);
placemark.openBalloon();
```

3.4.2. Метки

Метка обозначает место на карте с помощью значка. По умолчанию используется стандартный значок, однако его всегда можно заменить любым другим. В JavaScript API можно использовать изображения значков из встроенной коллекции и задавать некоторые параметры отображения метки. Кроме того, можно определять собственный способ отображения метки. Содержимое может быть обычным текстом или задаваться с помощью HTML. Метки интерактивны и реагируют на события мыши. По умолчанию при щелчке кнопкой мыши по метке открывается балун. Удерживая кнопку мыши, метку можно передвигать по карте (эту возможность необходимо включить).

Добавление метки на карту

Чтобы добавить метку на карту, необходимо передать в конструктор класса `YMaps.Placemark` координаты точки ее привязки и список параметров, а затем с помощью метода карты `addOverlay()` добавить метку на карту. Когда метка открывает балун, ее значок скрывается. Для того чтобы значок метки оставался на карте, необходимо установить опцию `hideIcon` в значение `false`. В листинге 3.27 приведен пример добавления метки на карту.

Листинг 3.27

```
// Создает метку в центре карты
var placemark = new YMaps.Placemark(map.getCenter(),{hideIcon: false});
// Устанавливает содержимое балуна
placemark.name = "Метка";
placemark.description = "Содержимое метки";
// Добавляет метку на карту
map.addOverlay(placemark);
```

Содержимое метки

В значке метки может размещаться любое HTML-содержимое. Стандартные значки меток растягиваются по ширине и высоте в зависимости от длины текста. Чтобы изменить содержимое значка метки используйте метод `setIconContent()`, а чтобы получить содержимое — метод `getIconContent()`.

```
placemark.setIconContent("Метка");
```

Чтобы очистить содержимое значка метки, вызовите метод `setIconContent()`, передав ему в качестве входного параметра занчение `null`:

```
placemark.setIconContent(null);
```

Перетаскивание метки

При перетаскивании метка последовательно генерирует три события: `DragStart`, `Drag` и `DragEnd`. Чтобы разрешить перетаскивание метки необходимо установить опцию `draggable` в значение `true`. Необходимо также создать обработчики события для метки. Пример представлен в листинге 3.28. При начале перетаскивания (событие) метки убираем балун, во время перетаскивания записываем в содержимое метки текущие координаты, по окончании перетаскивания изменяем содержимое балуна, записывая в него конечные координаты метки.

Листинг 3.28

```
// Создать метку
var placemark1 = new YMaps.Placemark(map.getCenter(),{draggable: true});
// Создать обработчики
YMaps.Events.observe(placemark1, placemark1.Events.DragStart, function (obj)
{ // закрыть балун
  obj.closeBalloon();
});
YMaps.Events.observe(placemark1, placemark1.Events.Drag, function (obj)
{ new_lng=obj.getCoordPoint().getLng();
  new_lat=obj.getCoordPoint().getLat();
  obj.setIconContent("Долгота ="+new_lng+" широта ="+new_lat);
});
```

```
YMaps.Events.observe(placemark1, placemark1.Events.DragEnd, function (obj)
{ // новое содержимое для балуна
  obj.description="Новая долгота ="+new_lng+" широта ="+new_lat;
  // открыть балун
  obj.openBalloon();
});
```

Задание стиля метки

В API предусмотрен ряд встроенных стилей для значков меток. Часть из них представлена на рис. 3.20 и 3.21. Полный список стандартных стилей для значков меток представлен в справочнике по программному интерфейсу, расположенному по адресу **http://api.yandex.ru/maps/jsapi/doc/ref/reference/styles.xml**. По умолчанию используется значок светло-голубого цвета (ключ стиля — `default#lightbluePoint`) с пустым содержимым.

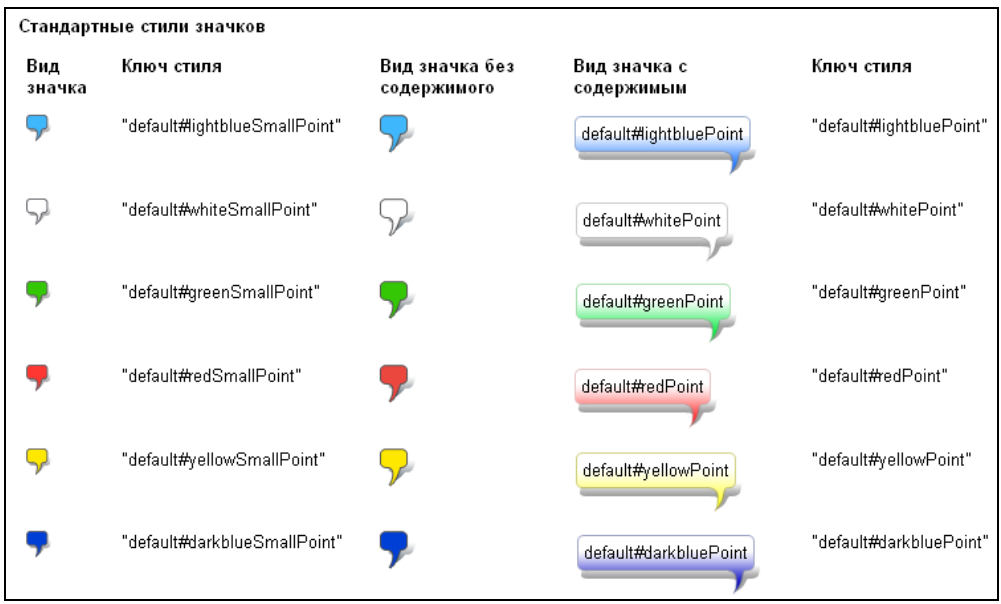


Рис. 3.20. Стандартные стили значков для меток

Чтобы применить встроенный стиль, укажите его ключ при создании метки. В листинге 3.29 представлен пример создания в видимой области карты нескольких меток со стандартными стилями.

Листинг 3.29

```
// Создание экземпляра карты и его привязка к созданному контейнеру
map = new YMaps.Map(document.getElementById("ymap"));
// Установка для карты ее центра и масштаба
map.setCenter(new YMaps.GeoPoint(41.96509425, 45.04268975), 13);
```

```
// Определение области обзора карты и связанных с ней параметров
var bounds = map.getBounds(),
pointLb = bounds.getLeftBottom(),
span = bounds.getSpan();
// Массив ключей стилей
var styleKeys = ["default#lightbluePoint", "default#whitePoint",
                 "default#greenPoint","default#redPoint",
                 "default#yellowPoint","default#darkbluePoint",
                 "default#nightPoint","default#greyPoint",
                 "default#bluePoint","default#orangePoint",
                 "default#darkorangePoint","default#violetPoint"];
// Добавление меток в видимую область карты
for (var i = 0; i <= styleKeys.length; i++)
{ var point = new YMaps.GeoPoint(pointLb.getLng() + span.x *
              Math.random(),pointLb.getLat() + span.y * Math.random());
  var placemark = new YMaps.Placemark(point, {style:styleKeys[i]});
  map.addOverlay(placemark);
}
```

Стандартные значки (пиктограммы)

Вид значка	Ключ стиля	Вид значка	Ключ стиля	Вид значка	Ключ стиля
	"default#airplaneIcon"		"default#anchorIcon"		"default#arrowDownLeftIcon"
	"default#arrowDownRightIcon"		"default#arrowLeftIcon"		"default#arrowRightIcon"
	"default#arrowUpIcon"		"default#attentionIcon"		"default#badmintonIcon"
	"default#bankIcon"		"default#barIcon"		"default#barberShopIcon"
	"default#bicycleIcon"		"default#bowlingIcon"		"default#buildingsIcon"
	"default#busIcon"		"default#cafeIcon"		"default#campingIcon"
	"default#carIcon"		"default#cellularIcon"		"default#cinemaIcon"
	"default#downhillSkiingIcon"		"default#dpsIcon"		"default#dryCleanerIcon"
	"default#electricTrainIcon"		"default#factoryIcon"		"default#fishingIcon"
	"default#gasStationIcon"		"default#gymIcon"		"default#hospitalIcon"

Рис. 3.21. Стандартные значки (пиктограммы)

Пример динамического управления свойствами метки

Создадим пример динамического управления свойствами, стилями и содержимым метки. Заодно рассмотрим в примере и создание обработчиков событий для меток. Исходные коды примера расположены на компакт-диске в папке glava_03\3-4-1. Посмотреть пример в действии можно по адресу **http://examples-api.bazakatalogov.ru/**

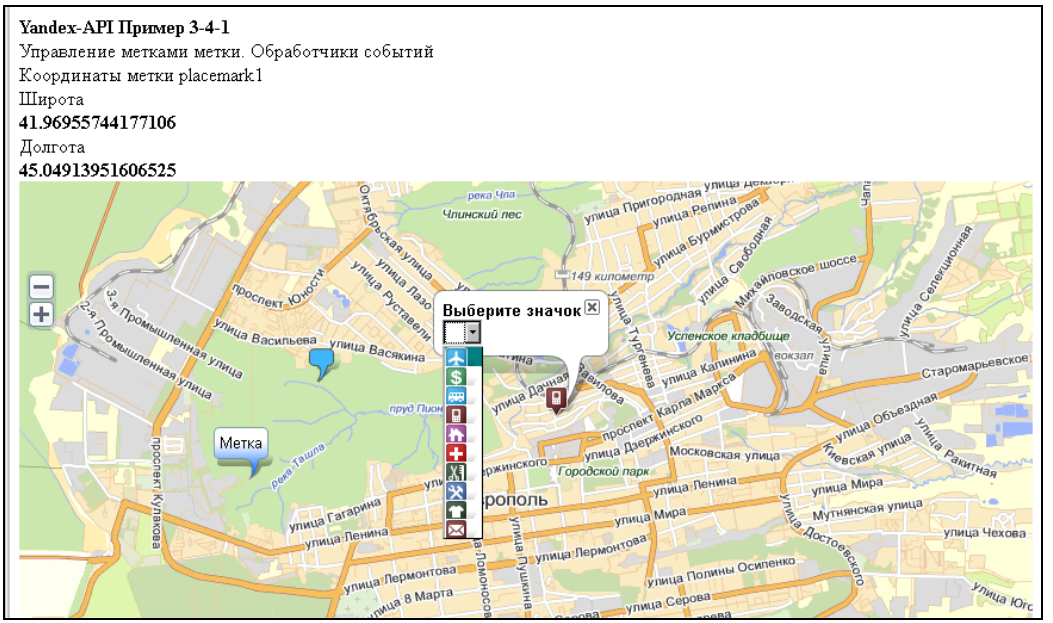


Рис. 3.22. Пример динамического управления свойствами меток

yandex/3-4-1/ (рис. 3.22). Содержимое файла index.php представлено в листинге 3.30.

Листинг 3.30

```
<?php
// API-ключ из файла настроек
require_once('my.php');
?>
<!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" xml:lang="ru" lang="ru">
<head>
<title>Yandex-API Пример 3-4-1 </title>
<script src="http://api-maps.yandex.ru/1.1/index.xml?key=<?php echo API_KEY;
?>" type="text/javascript"></script>
<script type="text/javascript">
var map;

var smallZoomControl;
var placemark1;
var placemark2;
var placemark3;
// массив стандартных иконок
var styles=new Array("default#airplaneIcon",
                     "default#bankIcon",
```

```
                    "default#busIcon",
                    "default#cellularIcon",
                    "default#houseIcon",
                    "default#hospitalIcon",
                    "default#barberShopIcon",
                    "default#workshopIcon",
                    "default#tailorShopIcon",
                    "default#mailPostIcon");
// массив картинок стандартных иконок
var images=new Array(
    "http://api-maps.yandex.ru/i/0.4/icons/airplane.png",
    "http://api-maps.yandex.ru/i/0.4/icons/bank.png",
    "http://api-maps.yandex.ru/i/0.4/icons/bus.png",
    "http://api-maps.yandex.ru/i/0.4/icons/cellular.png",
    "http://api-maps.yandex.ru/i/0.4/icons/house.png",
    "http://api-maps.yandex.ru/i/0.4/icons/hospital.png",
    "http://api-maps.yandex.ru/i/0.4/icons/barberShop.png",
    "http://api-maps.yandex.ru/i/0.4/icons/workshop.png",
    "http://api-maps.yandex.ru/i/0.4/icons/tailorShop.png",
    "http://api-maps.yandex.ru/i/0.4/icons/mailPost.png");
function ini()
{ map = new YMaps.Map(document.getElementById("ymap"));
  map.setCenter(new YMaps.GeoPoint(41.96509425, 45.04268975), 13);
  // стили метки
  var options = {name:"hospital",draggable:true,
      hideIcon:false,style:"default#hospitalIcon"};
  placemark1 = new YMaps.Placemark(new YMaps.GeoPoint(41.96509425,
      45.04268975), options);
  var select="<select onchange='new_icon(this.selectedIndex);'>";
  for(var j=0;j<styles.length;j++)
  { select=select+"<option style='background: url("+images[j]+")
    no-repeat;' value='"+j+"'>   ";
  }
  select=select+"</select>";
  placemark1.name ="Выберите иконку";
  placemark1.description =select;
  map.addOverlay(placemark1);
  // обработчик события для Drag — метки placemark1
  YMaps.Events.observe(placemark1, placemark1.Events.Drag, function (obj)
  { new_lng=obj.getCoordPoint().getLng();
    new_lat=obj.getCoordPoint().getLat();
    document.getElementById("lng").innerHTML="<b>"+new_lng+"</b>";
    document.getElementById("lat").innerHTML="<b>"+new_lat+"</b>";
  });
  // контент метки
  placemark2 = new YMaps.Placemark(new YMaps.GeoPoint(41.92509425,
    45.04268975),{draggable: true,hideIcon: false});
```

```
  placemark2.setIconContent("Метка");
  placemark2.name ="Задайте контент";
  placemark2.description ="<input value='Метка'
      onchange='placemark2.setIconContent(this.value);
      placemark2.closeBalloon();'>";
  map.addOverlay(placemark2);
  // обработчик события для DragStart — метки placemark2
  YMaps.Events.observe(placemark2, placemark2.Events.DragStart, function (obj)
  { obj.closeBalloon();
  });
  // обработчик события для DragEnd — метки placemark2
  YMaps.Events.observe(placemark2, placemark2.Events.DragEnd, function (obj)
  { new_lng=obj.getCoordPoint().getLng();
    new_lat=obj.getCoordPoint().getLat();
    alert("Новая Долгота="+new_lng+"\r\n Новая Широта="+new_lat);
  });
  // свойства метки
  placemark3 = new YMaps.Placemark(new YMaps.GeoPoint(41.93509425,
     45.05268975),{draggable: true,hideIcon: false});
  placemark3.name ="Свойства метки";
  var description ="<form id=form3 action='javascript:void();' >";
  description+="<input id=opt1 name=opt1 type=checkbox>перетаскивание";
  description+="<br><input id=opt2 name=opt2
                type=checkbox>видимость при открытом балуне";
  description+="<br><input type=button onclick='new_opt();'
                value='ok'></form>";
  placemark3.description =description;
  map.addOverlay(placemark3);
  // обработчик события для DblClick — метки placemark3
  YMaps.Events.observe(placemark3, placemark3.Events.DblClick, function (obj)
  { new_lng=obj.getCoordPoint().getLng();
    new_lat=obj.getCoordPoint().getLat();
    alert("Долгота="+new_lng+"\r\n Широта="+new_lat);
  });
  // добавить масштабирование
  smallZoomControl=new YMaps.SmallZoom({smooth:false});
  map.addControl(smallZoomControl);
}
//// доп. функции
// установить выбранную иконку для метки placemark1
function new_icon(arg)
{ var options1 = {draggable: true,hideIcon: false,style:styles[arg]};
  placemark1.setOptions(options1,true);
  placemark1.closeBalloon();
}
// установить новые свойства для метки placemark3
function new_opt()
```

```
{ if(document.forms.form3.elements.opt1.checked)
    var draggable1=true;
  else
    var draggable1=false;
  if(document.forms.form3.elements.opt2.checked)
    var hideIcon1=false;
  else
    var hideIcon1=true;
  var options3 = {draggable:draggable1 ,hideIcon: hideIcon1};
  placemark3.setOptions(options3,true);
  if(draggable1==true)
    document.forms.form3.elements.opt1.checked=true;
  if(hideIcon1==false)
    document.forms.form3.elements.opt2.checked=true;
  placemark3.closeBalloon();
}
</script>
</head>
<body onload='ini();'>
    <div>
    <b>Yandex-API Пример 3-4-1</b><br>
    Управление метками метки. Обработчики событий<br>
    Координаты метки placemark1<br>
    Широта <div id=lng><b>41.96509425</b></div>
    Долгота <div id=lat><b>45.04268975</b></div>
    </div>
    <div id="ymap" style="width:850px;height:500px"></div>
</body>
</html>
```

Для метки `placemark1` введем возможность изменения стандартного значка. В балуне для метки `placemark1` находится форма выбора значка (см. рис. 3.22). При выборе значения из выпадающего списка вызывается js-функция `new_icon()`, которая устанавливает новый стиль для метки `placemark1`.

Для метки `placemark2` введем возможность изменения содержимого. В балуне для метки `placemark1` находится `input`-поле для ввода нового содержимого метки (рис. 3.23). В поле можно ввести новое HTML-содержимое для данной метки.

Для метки `placemark3` введем возможность изменения параметров `draggable` (перетаскивание метки) и `hideIcon` (видимость метки при открытии балуна). В балуне для метки `placemark3` находятся два флажка для установки данных параметров (рис. 3.24).

Для метки `placemark1` создадим обработчик события `Drag`, который записывает текущие координаты метки при перемещении в верхнем блоке страницы.

```
YMaps.Events.observe(placemark1, placemark1.Events.Drag, function (obj)
{ new_lng=obj.getCoordPoint().getLng();
```

```
new_lat=obj.getCoordPoint().getLat();
document.getElementById("lng").innerHTML="<b>"+new_lng+"</b>";
document.getElementById("lat").innerHTML="<b>"+new_lat+"</b>";});
```

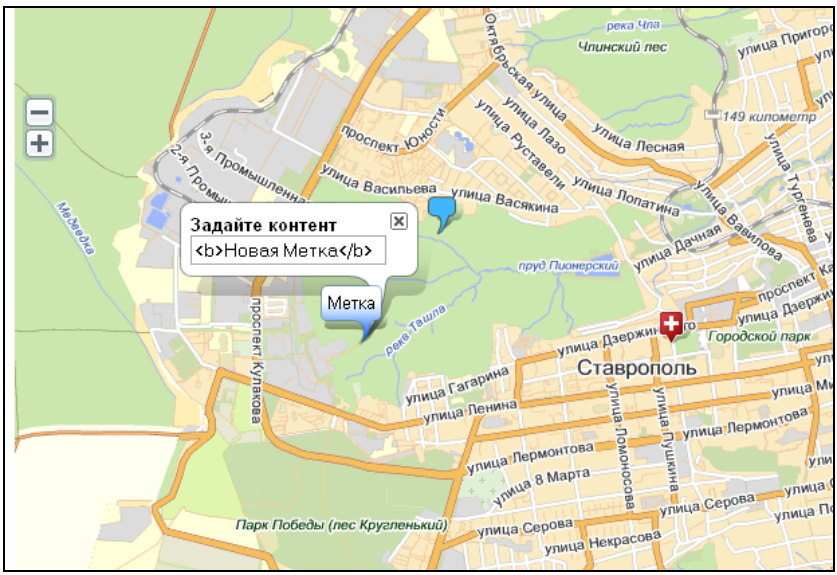


Рис. 3.23. Изменение содержимого метки `placemark2`

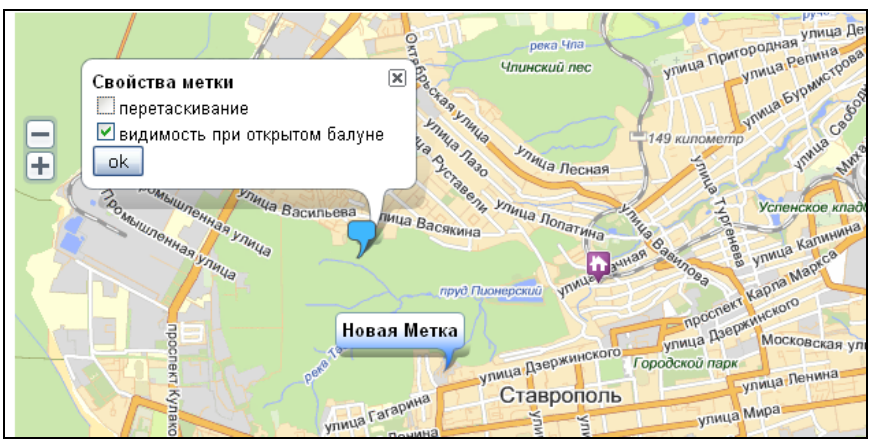


Рис. 3.24. Изменение свойств для метки `placemark3`

Для метки `placemark2` создадим обработчики события `DragStart`, который закрывает балун, и `DragEnd`, который выдает новые координаты метки через alert-сообщение.

```
YMaps.Events.observe(placemark2,placemark2.Events.DragStart,function (obj)
    {obj.closeBalloon();});
YMaps.Events.observe(placemark2,placemark2.Events.DragEnd,function (obj)
{ new_lng=obj.getCoordPoint().getLng();
```

```
  new_lat=obj.getCoordPoint().getLat();
  alert("Новая Долгота="+new_lng+"\r\n Новая Широта="+new_lat);});
```

Для метки `placemark3` создадим обработчик события `DblClick`, который выдает координаты метки `placemark3` через alert-сообщение.

```
YMaps.Events.observe(placemark3,placemark3.Events.DblClick,function (obj)
{ new_lng=obj.getCoordPoint().getLng();
  new_lat=obj.getCoordPoint().getLat();
  alert("Долгота="+new_lng+"\r\n Широта="+new_lat);});
```

Создание пользовательского значка метки

Для того чтобы в качестве значка метки использовать пользовательское изображение, необходимо воспользоваться классом `YMaps.IconStyle`:

- создать новый стиль значка — используется конструктор класса `YMaps.Style`; этот класс содержит набор данных, определяющих внешний вид объектов-оверлеев, помещаемых на карту, в том числе и стиль отображения значков меток;
- установить необходимые значения полей объекта `YMaps.IconStyle`:
  - `href` — URI графического файла значка;
  - `size` — размер значка в пикселах;
  - `offset` — сдвиг значка (в пикселах) относительно точки его позиционирования (используется для визуального выравнивания);
- создать новый стиль для тени значка (если требуется);
- передать созданный стиль либо в конструктор метки, либо в метод `setStyle()`.

В качестве примера создадим пользовательские значки меток для объектов из примера 3-4-1. Изображения для значков возьмем из значков для меток google.maps. Исходные коды примера расположены на компакт-диске в папке glava_03\3-4-2. Посмотреть пример в действии можно по адресу **http://examples-api.bazakatalogov.ru/yandex/3-4-2/** (рис. 3.25). Содержимое файла index.php представлено в листинге 3.31.

Листинг 3.31

```
... ... ...
<!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" xml:lang="ru" lang="ru">
<head>
<title>Yandex-API Пример 3-4-2 </title>
<script src="http://api-maps.yandex.ru/1.1/index.xml?key=<?php echo API_KEY;
?>" type="text/javascript"></script>
<script type="text/javascript">
var map;var group=new Array();var smallZoomControl;
```

```
function ini()
{ map = new YMaps.Map(document.getElementById("ymap"));
  map.setCenter(new YMaps.GeoPoint(41.96509425, 45.04268975), 13);
  ////// создание стилей
  // театры
  var s1 = new YMaps.Style();
  s1.iconStyle = new YMaps.IconStyle();
  s1.iconStyle.offset = new YMaps.Point(-15, -15);
  s1.iconStyle.href =
    "http://google-maps-icons.googlecode.com/files/theater.png";
  s1.iconStyle.size = new YMaps.Point(32, 37);
  YMaps.Styles.add("custom#theaterIcon", s1);
  // кинотеатры
  ... ... ...
  // банки
  ... ... ...
  //// создание групп
  // театры
  group[1] = new YMaps.GeoObjectCollection("custom#theaterIcon");
  group[1].add(createPlacemark(new YMaps.GeoPoint(41.96509425, 45.04268975),
'Академический театр драмы им. Лермонтова', 'Ставрополь пл. Ленина 1'));
  ... ... ...
  map.addOverlay(group[1]);
  // театры
  group[2] = new YMaps.GeoObjectCollection("custom#cinemaIcon");
  group[2].add(createPlacemark(new YMaps.GeoPoint(41.96940724, 45.04515414),
'Атлантис', 'Ставрополь ул. Маршала Жукова 8'));
  ... ... ...
  map.addOverlay(group[2]);
  // банки
  group[3] = new YMaps.GeoObjectCollection("custom#bankIcon");
  group[3].add(createPlacemark(new YMaps.GeoPoint(41.92636315, 45.00475977),
'БИН банк', 'Ставрополь ул. Доваторцев 61'));
  ... ... ...
  map.addOverlay(group[3]);
  // добавить масштабирование
  smallZoomControl=new YMaps.SmallZoom({smooth:false});
  map.addControl(smallZoomControl);
  // Функция создания метки
  function createPlacemark (geoPoint, name, description)
  { var placemark = new YMaps.Placemark(geoPoint);
    placemark.name = name;
    placemark.description = description;
    return placemark;
  }
}
```

```
</script>
</head>
<body onload='ini();'>
    <div>
    <b>Yandex-API Пример 3-4-2</b><br>
    Создание пользовательских значков меток<br>
    </div>
    <div id="ymap" style="width:850px;height:500px"></div>
</body>
</html>
```

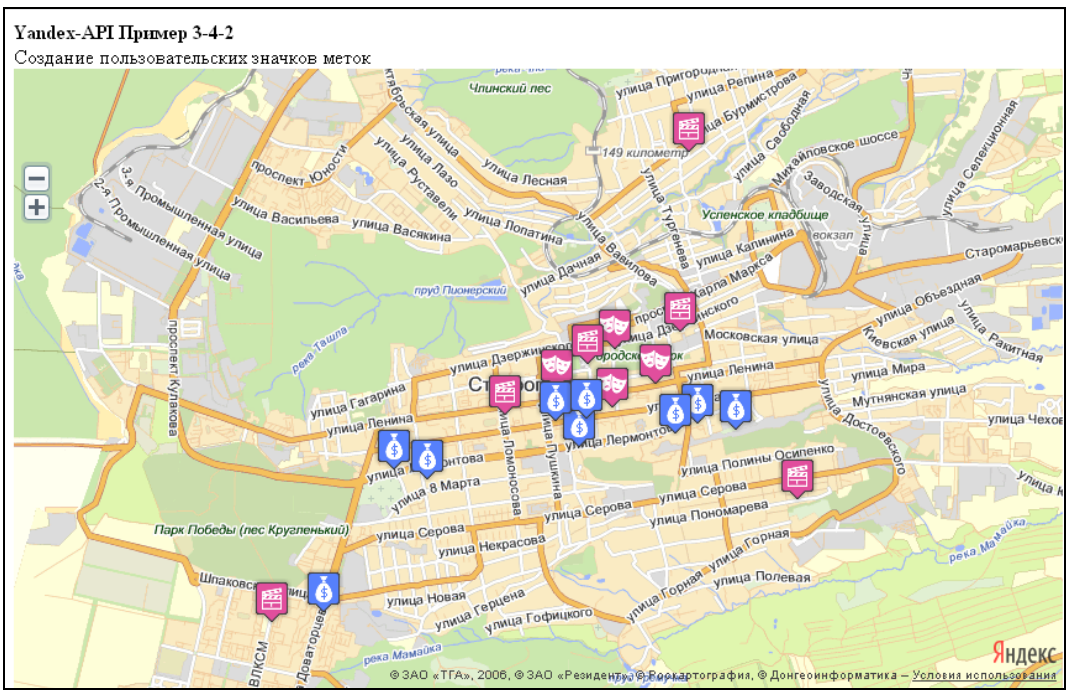


Рис. 3.25. Пользовательские значки для меток

3.4.3. Ломаная

Ломаная состоит из набора вершин, последовательно соединенных отрезками прямой. Ломаная может иметь самопересечения.

Добавление ломаной на карту

Чтобы добавить ломаную на карту, необходимо создать объект класса `YMaps.Polyline` и передать ему массив вершин ломаной, а затем с помощью метода карты `addOverlay()` добавить созданный объект на карту. Пример создания ломаной из пяти вершин представлен в листинге 3.32.

Листинг 3.32

```
var pl = new YMaps.Polyline([
  new YMaps.GeoPoint(37.7,55.7),new YMaps.GeoPoint(37.7,55.8),
  new YMaps.GeoPoint(37.8,55.8),new YMaps.GeoPoint(37.8,55.7),
  new YMaps.GeoPoint(37.7,55.7)]);
map.addOverlay(pl);
```

В режиме редактирования пользователю дается возможность изменять положение вершин ломаной, удалять имеющиеся вершины и добавлять новые.

Задание стиля ломаной

По умолчанию ломаная отображается линией красного цвета (ff0000ff) толщиной 1 пиксел, без прозрачности. Чтобы изменить стиль ломаной, задайте следующие параметры объекта `YMaps.LineStyle`:

- `strokeColor` — цвет и прозрачность линии (в системе RGBA);
- `strokeWidth` — толщину линии.

Пример определения стиля для ломаной и создания ломаной из пяти вершин представлен в листинге 3.33.

Листинг 3.33

```
var s = new YMaps.Style();
s.lineStyle = new YMaps.LineStyle();
s.lineStyle.strokeColor = '0000FF55';
s.lineStyle.strokeWidth = '5';
YMaps.Styles.add("user# Polyline1", s);
var pl = new YMaps.Polyline([new YMaps.GeoPoint(37.7,55.7),
  new YMaps.GeoPoint(37.7,55.8), new YMaps.GeoPoint(37.8,55.8),
  new YMaps.GeoPoint(37.8,55.7), new YMaps.GeoPoint(37.7,55.7)]);
pl.setStyle("user# Polyline1");
map.addOverlay(pl);
```

Примечание

Последние две цифры в `strokeColor` задают прозрачность ломаной.

Методы объекта *YMaps.Polyline*

Объект `YMaps.Polyline` имеет множество методов, позволяющих динамически управлять созданной ломаной. Методы объекта приведены в табл. 3.2.

Создадим пример динамического управления свойствами ломаной. В примере реализуем возможность перевода ломаной в режим редактирования и выход из режима редактирования, добавление, удаление, изменение порядка точек ломаной. Для нумерации точек ломаной будем создавать маркеры с номерами 0, ..., 9. Изображения

Таблица 3.2. *Методы объекта* `YMaps.ToolBarButton`

Имя	Возвращает	Описание
`addPoint(point, index)`		Добавляет одну или несколько вершин в линию. По умолчанию вершины добавляются в конец линии
`getNumPoints()`	`Integer`	Возвращает количество вершин в ломаной линии
`getPoint(index)`	`YMaps.ICoordPoint`	Возвращает координаты вершины по ее индексу
`getPoints()`	`YMaps.ICoordPoint[]`	Возвращает координаты вершин линии
`removePoint(index)`	`YMaps.ICoordPoint`	Удаляет из линии вершину с заданным индексом
`setEditingOptions(options)`		Устанавливает опции режима редактирования ломаной линии
`setPoints(points)`		Устанавливает координаты вершин линии
`startEditing()`		Включает режим редактирования ломаной линии. Должен вызываться после добавления ломаной на карту
`stopEditing()`		Отключает режим редактирования ломаной линии
`isDrawing()`	`Boolean`	Проверяет, находится ли ломаная линия в режиме рисования
`isEditing()`	Boolean	Проверяет, находится ли ломаная линия в режиме редактирования

для значков находятся в папке glava_03\3-4-3\img. Исходные коды примера расположены на диске в папке glava_03\3-4-3. Посмотреть пример в действии можно по адресу **http://examples-api.bazakatalogov.ru/yandex/3-4-3/** (рис. 3.26). Содержимое файла index.php представлено в листинге 3.34.

Листинг 3.34

```
... ... ...
<!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" xml:lang="ru" lang="ru">
<head>
... ... ...
<script type="text/javascript">
var map;
```

```
// массив маркеров
var m=new Array();
// массив стилей для маркеров
var st=new Array();var p1;var smallZoomControl;
function ini()
{ map = new YMaps.Map(document.getElementById("ymap"));
  map.setCenter(new YMaps.GeoPoint(41.96509425, 45.04268975), 13);
  // создаие стилей для маркера
  for(var i=0;i<10;i++)
  { st[i] = new YMaps.Style();
    st[i].iconStyle = new YMaps.IconStyle();
    st[i].iconStyle.offset = new YMaps.Point(-5, -5);
    st[i].iconStyle.size = new YMaps.Point(16, 16);
    st[i].iconStyle.href = "img/"+i+".png";
  }
  ////// создание стиля для полилинии
  var s = new YMaps.Style();
  s.lineStyle = new YMaps.LineStyle();
  s.lineStyle.strokeColor = '7010FF55';
  s.lineStyle.strokeWidth = '5';
  YMaps.Styles.add("user#polilyne1", s);
  // создание полилинии
  p1 = new YMaps.Polyline([
  new YMaps.GeoPoint(41.965,45.042),new YMaps.GeoPoint(41.995,45.043),
  new YMaps.GeoPoint(41.986,45.064)]);
  p1.setStyle("user#polilyne1");
  map.addOverlay(p1);
  // создание маркеров
  for(var i=0;i<p1.getNumPoints();i++)
  { m[i]=new YMaps.Placemark(new
YMaps.GeoPoint(p1.getPoint(i).getX(),p1.getPoint(i).getY() ), {style:st[i]});
    map.addOverlay(m[i]);
  }
  // обработчик события openBalloon для полилинии
  YMaps.Events.observe(p1, p1.Events.BalloonOpen,
  function ()
  { for(var j=0;j<p1.getNumPoints();j++)
    { map.removeOverlay(m[j]); }
    var description="<form id=form1  action='javascript:void();'>";
    description+="<select name=points size="+p1.getNumPoints()+">";
    for(var i=0;i<p1.getNumPoints();i++)
    { description+="<option value="+i+">"+p1.getPoint(i).getX()+
                  " "+p1.getPoint(i).getY();
    }
    description+="</select>";
    ... ... ...
    p1.setBalloonContent(description);
```

```
    if(!p1.isEditing())
    { for(var j=0;j<p1.getNumPoints();j++)
      { m[j]=new YMaps.Placemark(
               new YMaps.GeoPoint(p1.getPoint(j).getX(),
                  p1.getPoint(j).getY()), {style:st[j]});
        map.addOverlay(m[j]);
      }
    }
  },
  map);
  // добавить масштабирование карты
  ... ... ...
}
////// функции дополнительные polyline
// включение/выключение режима редактирования
// удаление точки
// изменение позиции точки
// удалить маркеры  для точек ломаной
// создать маркеры для точек ломаной
... ... ...
</script>
</head>
<body onload='ini();'>
    <div>
    <b>Yandex-API Пример 3-4-3</b><br>
    Динамическое управление параметрами ломаной. Редактирование <br>
    </div>
    <div id="ymap" style="width:850px;height:500px"></div>
</body>
</html>
```

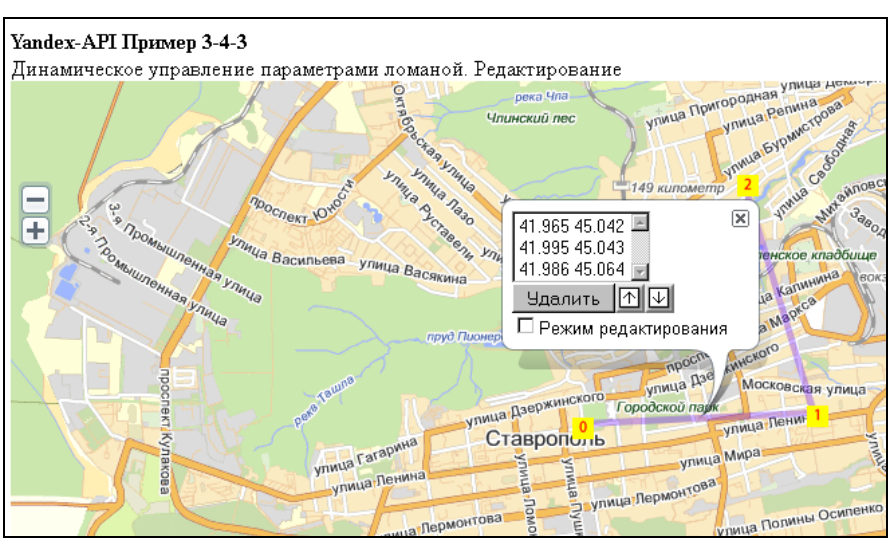


Рис. 3.26. Управление методами ломаной

При переходе в режим редактирования (установленный флажок) можно добавлять точки в ломаную (ограничение до 10) (рис. 3.27) При нажатии на кнопу **Удалить** удаляется выбранная в списке точка, при нажатии кнопок ↑ или ↓ изменяется порядок расположения точек в ломаной (рис. 3.28).

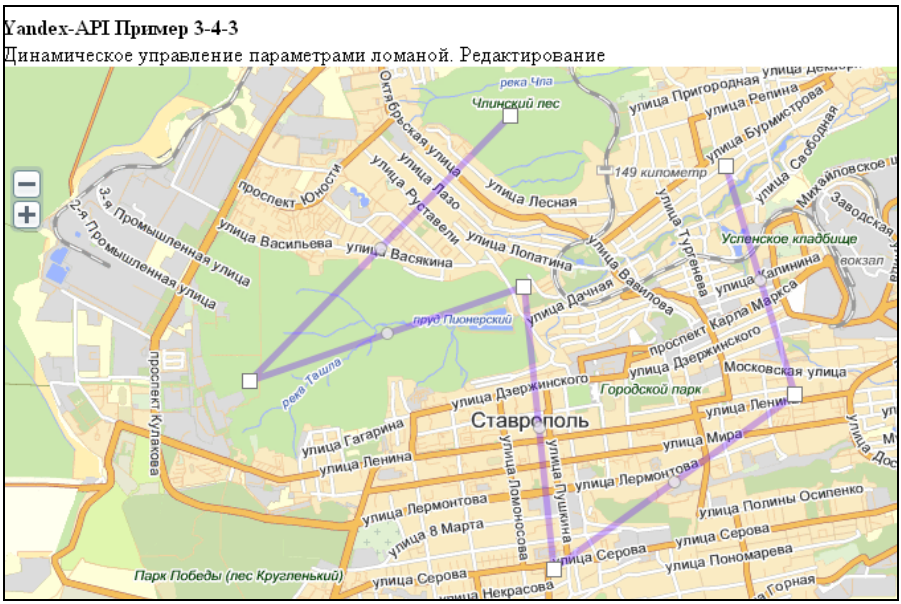


Рис. 3.27. Режим редактирования ломаной

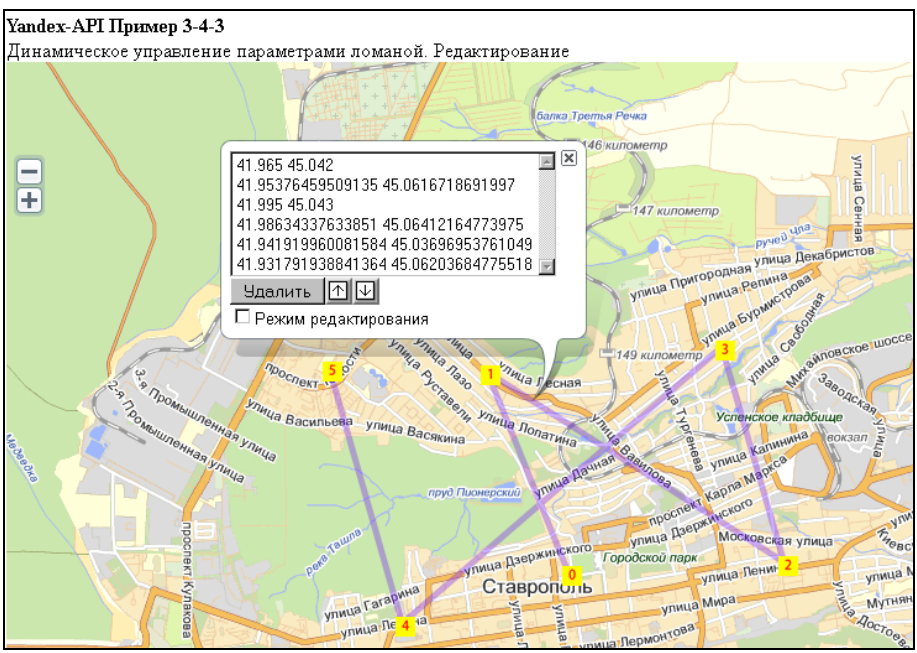


Рис. 3.28. Изменение порядка точек в ломаной

3.4.4. Многоугольник

Класс `YMaps.Polygon` позволяет рисовать многоугольники на карте. Класс во многом схож с `YMaps.Polyline`, поскольку многоугольник представляет собой замкнутую ломаную и ограниченную ею часть плоскости.

Добавление многоугольника на карту

Чтобы добавить многоугольник на карту, необходимо создать объект класса `YMaps.Polygon` и передать ему массив вершин многоугольника. Затем с помощью метода карты `addOverlay()` добавить объект на карту. Пример создания ломаной из пяти вершин представлен в листинге 3.35.

Листинг 3.35

```
var polygon = new YMaps.Polygon([ new YMaps.GeoPoint(37.7,55.7),
  new YMaps.GeoPoint(37.7,55.8), new YMaps.GeoPoint(37.8,55.8),
  new YMaps.GeoPoint(37.8,55.7), new YMaps.GeoPoint(37.7,55.7)]);
map.addOverlay(polygon);
```

В режиме редактирования пользователю дается возможность изменять положение вершин многоугольника, удалять имеющиеся вершины и добавлять новые.

Задание стиля многоугольника

По умолчанию многоугольник отображается непрозрачным, с заливкой красного цвета. Стиль отображения можно изменить, задав следующие параметры:

- `fill` — флаг, определяющий наличие заливки;
- `fillColor` — цвет заливки (в формате RGBA);
- `outline` — флаг, определяющий наличие обводки;
- `strokeColor` — цвет линии обводки (в формате RGBA);
- `strokeWidth` — толщина линии обводки.

Пример определения стиля для многоугольника и создания многоугольника из пяти вершин представлен в листинге 3.36.

Листинг 3.36

```
var style = new YMaps.Style();
style.polygonStyle = new YMaps.PolygonStyle();
style.polygonStyle.fill = true;
style.polygonStyle.outline = true;
style.polygonStyle.strokeWidth = 10;
style.polygonStyle.strokeColor = "ffff0088";
style.polygonStyle.fillColor = "ff000055";
polygon.setStyle(style);
```

Примечание

Последние две цифры в `fillColor` и `strokeColor` задают прозрачность.

Методы объекта *YMaps.Polygon*

Перечень методов объекта `YMaps.Polygon` приведен в табл. 3.3.

Таблица 3.3. *Методы объекта* `YMaps.Polygon`

Имя	Возвращает	Описание
`addPoint(`*point*`, `*index*`)`		Добавляет одну или несколько вершин в многоугольник. По умолчанию вершины добавляются в конец линии
`contains(`*coordPoint*`)`	`Boolean`	Определяет, находится ли визуально точка *coordPoint* внутри многоугольника
`getNumPoints()`	`Integer`	Возвращает количество вершин в многоугольнике
`getPoint(`*index*`)`	`YMaps.ICoordPoint`	Возвращает координаты вершины по ее индексу
`getPoints()`	`YMaps.ICoordPoint[]`	Возвращает координаты вершин многоугольника
`removePoint(`*index*`)`	`YMaps.ICoordPoint`	Удаляет из линии вершину с заданным индексом
`setEditingOptions(`*options*`)`		Устанавливает опции режима редактирования многоугольника
`setPoints(`*points*`)`		Устанавливает координаты вершин многоугольника
`startEditing()`		Включает режим редактирования многоугольника. Должен вызываться после добавления многоугольника на карту
`stopEditing()`		Отключает режим редактирования многоугольника
`isDrawing()`	`Boolean`	Проверяет, находится ли многоугольник в режиме рисования
`isEditing()`	`Boolean`	Проверяет, находится ли многоугольник в режиме редактирования

Создадим пример динамического управления свойствами многоугольника. В примере реализуем возможность перевода многоугольника в режим редактирования и выход из режима редактирования, добавление, удаление, изменение порядка вершин многоугольника. Для нумерации точек ломаной будем создавать маркеры с номерами 0, ..., 9. Изображения для значков находятся в папке glava_03\3-4-4\img. Исходные коды примера расположены на диске в папке glava_03\3-4-4. Посмот-

реть пример в действии можно по адресу **http://examples-api.bazakatalogov.ru/yandex/3-4-4/** (рис. 3.29). Содержимое файла index.php представлено в листинге 3.37.

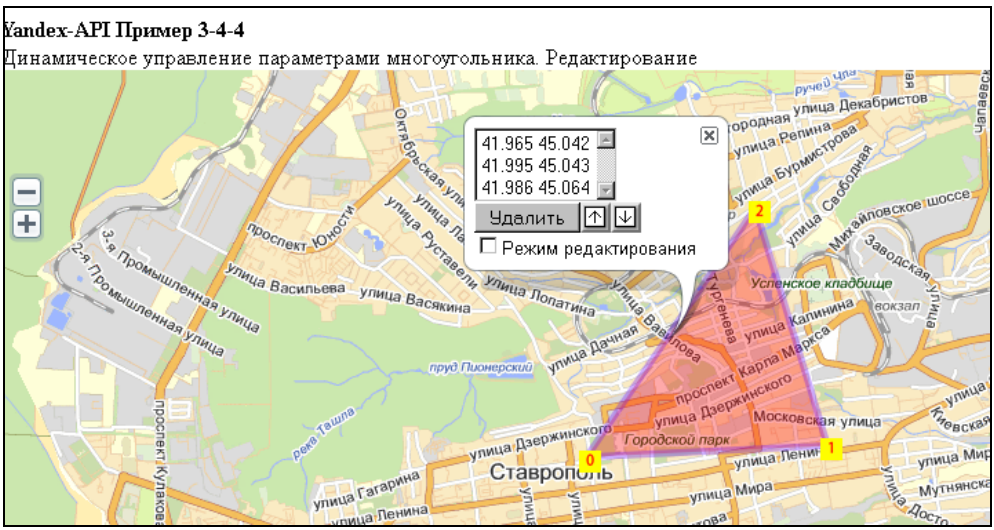


Рис. 3.29. Управление методами многоугольника

Листинг 3.37

```
<?php
// API-ключ
require_once('my.php');
?>
<!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" xml:lang="ru" lang="ru">
<head>
... ... ...
<script type="text/javascript">
var map;var m=new Array();var st=new Array();var p1;var smallZoomControl;
function ini()
{ map = new YMaps.Map(document.getElementById("ymap"));
  map.setCenter(new YMaps.GeoPoint(41.96509425, 45.04268975), 13);
  // создаие стилей для маркера
  for(var i=0;i<10;i++)
  { st[i] = new YMaps.Style();
    st[i].iconStyle = new YMaps.IconStyle();
    st[i].iconStyle.offset = new YMaps.Point(-5, -5);
    st[i].iconStyle.size = new YMaps.Point(16, 16);
    st[i].iconStyle.href = "img/"+i+".png";
  }
```

```
  ////// создание стиля для многоугольника
  var s = new YMaps.Style();
  s.polygonStyle = new YMaps.LineStyle();
  s.polygonStyle.strokeColor = '7010FF55';
  s.polygonStyle.fillColor = "ff000055";
  s.polygonStyle.strokeWidth = '5';
  YMaps.Styles.add("user#poligone1", s);
  // создание многоугольника
  p1 = new YMaps.Polygon([
  new YMaps.GeoPoint(41.965,45.042),new YMaps.GeoPoint(41.995,45.043),
  new YMaps.GeoPoint(41.986,45.064)]);
  p1.setStyle("user#poligone1");
  map.addOverlay(p1);
  ... ... ...
  // обработчик события openBalloon для многоугольника
  YMaps.Events.observe(p1, p1.Events.BalloonOpen,
  function ()
  { for(var j=0;j<p1.getNumPoints();j++)
    { map.removeOverlay(m[j]); }
    var description="<form id=form1  action='javascript:void();'>";
    description+="<select name=points size="+p1.getNumPoints()+">";
    for(var i=0;i<p1.getNumPoints();i++)
    { description+="<option value="+i+">"+p1.getPoint(i).getX()+"
                  "+p1.getPoint(i).getY(); }
    ... ... ...
    p1.setBalloonContent(description);
    if(!p1.isEditing())
    { for(var j=0;j<p1.getNumPoints();j++)
      { m[j]=new YMaps.Placemark(
               new YMaps.GeoPoint(p1.getPoint(j).getX(),
                 p1.getPoint(j).getY() ), {style:st[j]});
        map.addOverlay(m[j]);
      }
    }
  },
  map);
  // добавить масштабирование карты
  ... ... ...
}
///// функции дополнительные polygon
// включение/выключение режима редактирования
// удаление точки
// изменение позиции точки
// удалить маркеры для вершин многоугольника
// создать маркеры для вершин многоугольника
... ... ...
</script>
</head>
```

```
<body onload='ini();'>
    <div>
    <b>Yandex-API Пример 3-4-4</b><br>
    Динамическое управление параметрами многоугольника. Редактирование
    <br></div>
    <div id="ymap" style="width:850px;height:500px"></div>
</body>
</html>
```

При переходе в режим редактирования (установлен флажок) можно добавлять вершину многоугольника (ограничение до 10). При нажатии кнопки **Удалить** удаляется выбранная в списке точка, при нажатии кнопок ↑ или ↓ изменяется порядок расположения вершин многоугольника.

3.4.5. Всплывающая подсказка

Объект класса `YMaps.Hint` существует для каждой карты в единственном экземпляре и доступен в объекте карты как `YMaps.Map.hint`. Методы объекта представлены в табл. 3.4.

Таблица 3.4. *Методы объекта* `YMaps.Hint`

Метод	Описание
`hide(`*`hideTimeout`*`)`	Скрывает всплывающую подсказку либо немедленно, либо через заданный интервал времени (если указан параметр *`hideTimeout`*)
`isShown()`	Возвращает текущее состояние всплывающей подсказки: `true` — подсказка показана на карте, `false` — подсказка скрыта
`moveTo(`*`position`*`)`	Перемещает всплывающую подсказку в заданную позицию
`setContent(`*`content`*`)`	Устанавливает содержимое всплывающей подсказки
`setOptions(`*`options`*`)`	Устанавливает опции всплывающей подсказки
`show(`*`position`*`, `*`content`*`, `*`options`*`)`	Показывает всплывающую подсказку с заданными параметрами

Для того чтобы подсказка показывалась при наведении курсора на объект-оверлей (метку, ломаную или многоугольник), необходимо установить параметр `hasHint` соответствующего объекта в значение `true`. По умолчанию внутри всплывающей подсказки показывается значение поля `name`. В примере (листинг 3.38) создадим подсказку для метки, при этом зададим свой стиль для метки.

Листинг 3.38

```
var new_hint = new YMaps.Style();
new_hint.hintContentStyle = new YMaps.HintContentStyle(
new YMaps.Template(<div style="background-color:green" >
```

```
$[description]</div>"));
var placemark = new YMaps.Placemark(map.getCenter(), {hasHint: true, style:
new_hint });
placemark.description = "Подсказка для  метки";
map.addOverlay(placemark);
```

3.4.6. Группировка объектов

Однотипные объекты-оверлеи можно объединить в группу, а затем оперировать этой группой как единым целым: отображать/скрывать объекты, одновременно производить действия над всеми членами группы и т. д. Для совершения действий над группой объектов используйте методы класса `YMaps.GeoObjectCollection`. Чтобы добавить объект (или массив объектов) в группу, необходимо использовать метод `add()`, для удаления объекта из группы следует использовать метод `remove()`. В примере (листинг 3.39) показано, как создать группу меток и добавить ее на карту одной командой.

Листинг 3.39

```
// Создание коллекции
group1=new YMaps.GeoObjectCollection();
// Добавление меток в коллекцию
pl1=new YMaps.Placemark (new YMaps.GeoPoint(41.96509425, 45.04268975));
pl2=new YMaps.Placemark (new YMaps.GeoPoint(41.97310869, 45.04657263));
pl3=new YMaps.Placemark (new YMaps.GeoPoint(41.97875205, 45.04317655));
group1.add([pl1,pl2,pl3]);
// Вывод коллекции на карту
map.addOverlay(group1);
```

Чтобы узнать общее количество объектов в группе, необходимо вызвать метод `length()`.

```
var length = group.length();
```

Для доступа к отдельным объектам группы надо использовать методы `get()` (возвращает объект из группы по его индексу) и `filter()` (возвращает массив объектов, удовлетворяющих некоторому условию).

```
gCollection.get(2) // третий объект
```

Чтобы выполнить какие-либо действия над всеми элементами группы, используйте метод `forEach()`. Метод позволяет вызвать указанную пользовательскую функцию для каждого объекта группы.

Чтобы задать единый стиль всем элементам группы, передайте ключ (или указатель) стиля в качестве параметра в конструктор класса `YMaps.GeoObjectCollection`. Указанный стиль будет автоматически применен ко всем объектам группы, например:

```
var group1=new YMaps.GeoObjectCollection("default#greenPoint");
```

В примере 3-4-2, рассмотренном в *подразд. "Пример динамического управления свойствами метки" разд. 3.4.2*, показана работа по созданию групп меток, добавлению меток в группу, удаление с карты и вывод на карту групп меток.

3.5. Сервисы

Для того чтобы расширить функциональные возможности карты, API предоставляет ряд дополнительных сервисов:

- геокодирование — определение географического местоположения объектов по почтовому адресу, и наоборот;
- геотаргетинг — определение предположительного местоположения пользователя, вычисленное на основе его IP-адреса;
- маршрутизация — прокладывание маршрута по набору точек на карте;
- визуализация YMapsML — использование специализированного XML-языка для хранения сведений о географических объектах и их внешнем виде;
- карта пробок — показывает загруженность дорог в 49 городах России и ближнего зарубежья.

Рассмотрим эти сервисы подробнее.

3.5.1. Геокодирование

Класс `YMaps.Geocoder` позволяет отправлять запросы геокодеру, получать информацию о статусе и результате процесса геокодирования, а также получать результаты геокодирования в виде меток. Конструктор класса принимает два параметра:

- `request` — строка с адресом, который нужно геокодировать (для обратного геокодирования — точка с географическими координатами);
- `options` — параметры:
  - `boundedBy` — область на карте, в которой осуществляется поиск объекта;
  - `strictBounds` — флаг, указывающий, что искать следует только внутри области, заданной опцией `boundedBy`;
  - `results` — требуемое количество результатов поиска (не более);
  - `skip` — указание пропустить первые *n* результатов в ответе;
  - `prefLang` — предпочитаемый язык ответа.

Чтобы определить координаты объекта, необходимо создать экземпляр класса `YMaps.Geocoder` и передать ему в качестве параметра строку с адресом объекта. Результат поиска будет представлен в виде группы специальных меток класса `YMaps.GeocoderResult`. Эти метки отличаются от обычных меток класса `YMaps.Placemark` наличием четырех дополнительных полей, в которых передаются дополнительные сведения о найденном объекте. Класс `YMaps.GeocoderResult` реализует интерфейс `YMaps.IOverlay`, поэтому метки `YMaps.GeocoderResult` можно добавлять на карту точно так же, как и обычные.

Для обработки результатов геокодирования необходимо учитывать, что геокодеру требуется время на обработку запроса и ответ. Поэтому рекомендуется всегда использовать обработчики событий для получения информации об окончании геокодирования и о его текущем статусе. В классе `YMaps.Geocoder` предусмотрены два события: `Load` (геокодирование прошло без ошибок) и `Fault` (с ошибками). В обработчике события `Load` можно оперировать результатами геокодирования как группой. Пример, представленный в листинге 3.40, иллюстрирует запрос и обработку событий геокодера. В случае успешного геокодирования пользователю карты будет показано сообщение о количестве найденных объектов, а самый релевантный (первый) из них будет изображен на карте.

Листинг 3.40

```
YMaps.Geocoder("Ставрополь ул. Селекционная 4");
YMaps.Events.observe(geocoder, geocoder.Events.Load, function () {
  if (this.length()) {
    alert("Найдено :" + this.length());
    map.addOverlay(this.get(0));
    map.panTo(this.get(0).getGeoPoint())  }
  else
  {alert("Объект не найден");}
});
YMaps.Events.observe(geocoder, geocoder.Events.Fault, function (error) {
  alert("Произошла ошибка: " + error.message)
});
```

Объект `YMaps.Geocoder` поддерживает обратное геокодирование, с помощью которого можно преобразовывать координаты объекта в почтовый адрес. Для обратного геокодирования в параметре `request` требуется передать вместо адреса геоточку (`YMaps.GeoPoint`).

В *главе 4* мы создадим проект каталога предприятий, использующий геокодирование для отображения положения предприятия на Яндекс.Карте.

3.5.2. Геотаргетинг

С помощью API Яндекс.Карт можно определить предположительное местоположение пользователя, вычисленное на основе его IP-адреса. Информация о местоположении пользователя доступна в объекте `YMaps.location()` и имеет следующие поля:

- `longitude` — долгота;
- `latitude` — широта;
- `city` — название города;
- `region` — название региона;
- `country` — название страны.

Информация отсутствует, если определить местоположение пользователя не удалось. В листинге 3.41 показан пример установки центра карты в месте предполагаемого нахождения пользователя.

Листинг 3.41

```
// Получение информации о местоположении пользователя
if (YMaps.location)
{ center = new YMaps.GeoPoint(YMaps.location.longitude,
                              YMaps.location.latitude);
  map.openBalloon(center, "Место вашего нахождения<br/>"
  + (YMaps.location.country || "")
  + (YMaps.location.region ? ", " + YMaps.location.region : "")
  + (YMaps.location.city ? ", " + YMaps.location.city : ""))
}
else
{ center = new YMaps.GeoPoint(37.64, 55.76); }
// Установка для карты ее центра и масштаба
map.setCenter(center, 10);
```

3.5.3. Маршрутизация

Маршрутизатор — сервис автоматического прокладывания маршрутов на Яндекс.Картах. Сервис позволяет автоматически вычислять маршрут перемещения между заданными пунктами и получать различную информацию о проложенном маршруте (протяженность маршрута, время преодоления, а также данные, необходимые для построения маршрутного листа).

На момент написания книги сервис находился в режиме бета-тестирования и был доступен для построения маршрутов по Москве и Московской области и в Украине.

Точки маршрута

Для прокладывания маршрута используется класс `YMaps.Router`, которому необходимо передать точки остановки и транзитные точки маршрута. Точка остановки (`YMaps.WayPoint`) — точка, в которой предполагается или возможна остановка. Начальная и конечная точки маршрута всегда являются точками остановки. Транзитная точка (`YMaps.ViaPoint`) — точка, в которой остановка не предполагается, но через которую нужно проложить маршрут. Массив индексов транзитных точек передается вторым параметром конструктора `YMaps.Router`. Нумерация индексов начинается с нуля (см. пример далее). Точки маршрута можно задать, указав либо их координаты, либо адрес. Во втором случае координаты будут вычислены автоматически с помощью геокодера. Если через какую-то точку невозможно проложить маршрут (например, точка находится вне трассы), маршрутизатор попытается проложить маршрут максимально близко к ней. В листинге 3.42 приведены варианты для прокладывания маршрута.

Листинг 3.42

```
// Прокладывание маршрута с тремя точками останова с указанием адреса
var router1 = new YMaps.Router(['Арбатская', 'Кропоткинская',
                               'Курская', 'Третьяковская']);
// Прокладывание маршрута с двумя точками останова
// и двумя транзитными точками
var router1 = new YMaps.Router(['Арбатская', 'Кропоткинская', 'Курская'
'Третьяковская'],[1,2]);
// Точки маршрута с указанием координат
var router = new YMaps.Router([new YMaps.GeoPoint(37.658509,55.758331),
                              new YMaps.GeoPoint(37.594306,55.735429)]);
```

События построения маршрута

Во время построения маршрута объекты класса `YMaps.Router` генерируют события, которые могут быть обработаны стандартными средствами JavaScript API:

- `Fault` — событие неудачной загрузки ответа с сервера маршрутизации;
- `GeocodeError` — событие неудачного построения маршрута вследствие ошибки геокодирования;
- `Load` — событие успешного получения ответа от сервиса маршрутизации;
- `RouteError` — событие неудачного построения маршрута вследствие невозможности проложить путь к точке;
- `Success` — событие успешного построения маршрута.

События генерируются следующим образом. Вначале отправляется запрос серверу. Если ответ получен, генерируется событие `Load`, в противном случае — событие `Fault`. Эти два события показывают, может ли быть установлено соединение с сервером. Если соединение с сервером установлено, производится попытка построения маршрута. Если маршрут проложен, генерируется событие `Success`. В противном случае генерируется одно из событий: `GeocodeError` или `RouteError`, отражающих причины, по которым не удалось вычислить маршрут. При этом в обработчик события будет передан индекс точки, которую не удалось геокодировать или до которой невозможно проложить маршрут. Так как построение маршрута — асинхронная операция, оперировать объектами класса `YMaps.Router` можно только после того, как было сгенерировано событие `Success`. Желательно создать и обработчики других событий (листинг 3.43).

Листинг 3.43

```
var router = new Ymaps.Router(['Москва, Арбатская',
  'Москва, метро Китай-Город', 'Москва, метро Третьяковская']);
// Удачно
YMaps.Events.observe(router, router.Events.Success, function() {
  router.getWayPoint(0).setIconContent('Точка отправления');
```

```
  router.getWayPoint(1).setIconContent('Точка прибытия');
  map.addOverlay(router);
});
// Неудачное геокодирование
YMaps.Events.observe(router, router.Events.GeocodeError, function (number) {
  alert('Ошибка при геокодировании точки № ' + number);
})
// Недоступность точки
YMaps.Events.observe(router, router.Events.RouteError, function (number) {
  alert('Не удается проложить маршрут до точки № ' + number);
});
```

Отрезки маршрута

`YMaps.Router` задает отрезок между двумя соседними точками остановки. Таким образом, если в конструктор маршрутизатора переданы 3 точки остановки, то будут доступны два отрезка маршрута. Проложенный с помощью `YMaps.Router` маршрут представляется в виде списка объектов `YMaps.Route`. Для определения количества объектов `YMaps.Route` используется метод `getNumRoutes()`. Для доступа к конкретному объекту `YMaps.Route` по его индексу применяется метод `getRoute()`.

```
var router = new YMaps.Router(['Арбатская', 'Кропоткинская','Третьяковская']);
var route = [];
for (var i=0; i<router.getNumRoutes(); i++) {
  route[i] = route.getRoute(i);
}
```

Объекты класса `YMaps.Route` описывают траекторию перемещения с помощью ломаной линии (`YMaps.Polyline`), которая разбивается на отрезки (`YMaps.RouteSegment`). Класс `YMaps.RouteSegment` описывает логически выделенный отрезок маршрута. Достижение конца отрезка при движении по маршруту может указывать, например, на то, что производится выезд на другую улицу или на то, что в данном месте возможен поворот. Как для полного, так и для промежуточных маршрутов и их отрезков можно получить информацию о протяженности, времени передвижения по ним и ряд других параметров (см. пример в листинге 3.44). Для определения количества отрезков в маршруте используется метод `getNumRouteSegments()`. Для доступа к конкретному отрезку по его индексу — метод `getRouteSegment()`. Нумерация индексов начинается с нуля.

Листинг 3.44

```
var points=['Арбатская', 'Кропоткинская','Третьяковская'];
var router = new YMaps.Router(points,[], { viewAutoApply: true });
var route = [];var list='';
for (var i=0; i<router.getNumRoutes(); i++)
{ list+=i+points[i]+''+points[i+1]+'<br>';
  route[i] = route.getRoute(i);
```

```
  // длина маршрута
  distance=route[i].getDistance();
  // получить отрезки для маршрута
  for(var j=0; j< route[i].getNumRouteSegments();j++)
  {  var segment = route[j].getRouteSegment(j);
     list+='Едем ' + action[segment.getAction()] + " на " +
     segment.getStreet() + ', проезжаем ' + segment.getDistance() +
                          ' м. ' + '<br>';
   }
}
alert(list);
```

Отображение маршрута на карте

Класс `YMaps.Router` реализует интерфейс `YMaps.IOverlay`, т. е. проложенный маршрут может быть сразу помещен на карту. Использование опции `viewAutoApply` позволяет автоматически подбирать центр и масштаб карты таким образом, чтобы маршрут был виден на карте целиком и при этом уровень масштабирования карты был максимален (листинг 3.45).

Листинг 3.45

```
var router = new YMaps.Router(
  ['Арбатская','Кропоткинская','Третьяковская'],[],{ viewAutoApply:true});
// Добавляет на карту маршрут
map.addOverlay(router);
```

3.5.4. Визуализация YMapsML

YMapsML (Yandex Maps Markup Language) — язык описания географических данных, разработанный в компании "Яндекс". Под географическими данными подразумеваются любые данные, географически привязанные к местности. YMapsML используется как формат передачи данных между сервисами Яндекс.Карт и сторонними программными средствами. Данные в формате YMapsML могут быть отображены на Яндекс.Картах, в этом же формате возвращает информацию о местоположении объектов сервис геокодирования. YMapsML представляет собой открытый XML-формат, формальная спецификация (схема) которого доступна по адресу **http://maps.yandex.ru/schemas/ymaps/1.x/ymaps.xsd**, а описание приведено в справочнике по языку YMapsML.

Объект `YMaps.YMapsML` позволяет отобразить YMapsML-документ на карте, используя JavaScript. Чтобы отобразить документ, необходимо передать на вход конструктора `YMaps.YMapsML` URL документа, а затем добавить созданный объект-оверлей на карту. Так как загрузка YMapsML-документа — асинхронный процесс, необходимо отслеживать окончание загрузки с помощью события `Load`. Пример загрузки YMapsML документа представлен в листинге 3.46.

Листинг 3.46

```
var ml = new YMaps.YMapsML(
  "http://api.yandex.ru/maps/ymapsml/examples/xml/demonstration.xml");
map.addOverlay(ml);
YMaps.Events.observe(ml, ml.Events.Load, function ()
{alert("Данные загружены");});
```

3.5.5. Карта пробок

25 августа 2010 года компанией "Яндекс" было заявлено о возможности добавления карты пробок на любой сайт с использованием API Яндекс.Карт. Данный модуль позволяет добавить на карту слои пробок и дорожных событий, отображающие состояние автомобильного движения в городе, а также элемент управления "Пробки" (рис. 3.30).

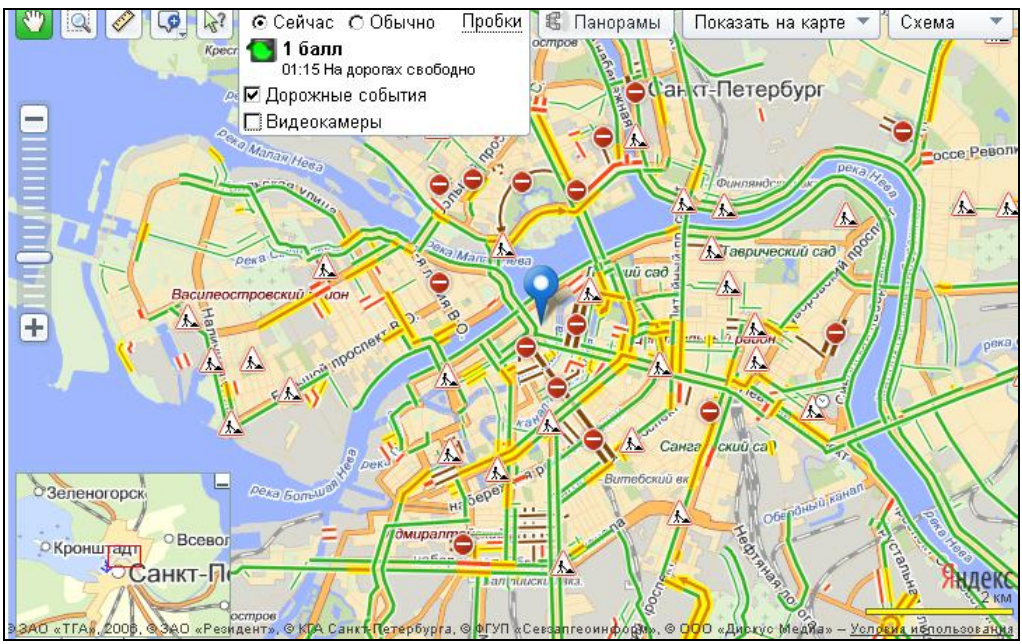


Рис. 3.30. Карта пробок и дорожных событий

На момент написания книги информация о пробках на дорогах была доступна для 38 городов России, 26 городов Украины, 2 городов Казахстана и 1 города Белоруссии. Для отображения автомобильных пробок на карте требуется загружать API вместе с модулем traffic. Модуль можно загружать при загрузке API. Для этого в строке подключения необходимо указать дополнительный параметр `modules` со значением `traffic`:

```
<script src="http://api-maps.yandex.ru/1.1/index.xml?key=<API-
ключ>&modules=traffic" type="text/javascript"></script>
```

Модуль можно загружать и по требованию. В этом случае API подключается в заголовке страницы:

```
<script src="http://api-maps.yandex.ru/1.1/index.xml?key=<API-ключ>"
type="text/javascript"></script>
```

Модуль traffic можно подключить в любой момент, вызвав метод `YMaps.load()` следующим образом:

```
YMaps.load('traffic', callback);
```

Функция callback будет вызвана после окончания загрузки модуля.

После подключения модуля traffic необходимо создать элемент управления "Пробки" и добавить его на карту:

```
var traffic = new YMaps.Traffic.Control();
map.addControl(traffic);
```

Для того чтобы отобразить пробки программно и развернуть кнопку пробок, необходимо использовать метод `show()`:

```
traffic.show();
```

Чтобы скрыть пробки и, соответственно, свернуть кнопку, необходимо использовать метод `hide()`. Чтобы узнать, показываются ли в данный момент пробки или нет, необходимо использовать метод `isShown()`. Слой пробок автоматически обновляется раз в несколько минут при наличии активности пользователя. Чтобы управлять обновлением пробок программно, необходимо использовать метод `update()`:

```
traffic.update();
```

Кроме настройки переключателя слоя дорожных событий, для слоев пробок и дорожных событий можно задать различные реакции на действия пользователя при наведении мыши на пробку или значок дорожного события:

- `cursor` — вид курсора (список курсоров см. в разделе справочника `YMaps.Cursor`);
- `hasHint` — показывать ли всплывающую подсказку;
- `hasBalloon` — показывать ли по щелчку мыши балун.

Для задания опций необходимо использовать метод `setOptions()`. Также значения опций можно передать первым параметром конструктора.

Элемент управления "Пробки" может быть свернут (при этом показываются пробки) либо развернут (пробки скрыты). Слой дорожных событий также может быть либо показан, либо скрыт. Метод `getState()` возвращает текущее состояние элемента управления:

```
traffic.getState();
```

Чтобы задать состояние элемента управления, необходимо использовать метод `setState()`:

```
traffic.setState({
  shown: true,              // элемент управления развернут, пробки показаны
  infoLayerShown: false     // слой дорожных событий скрыт
});
```

Состояние элемента управления также можно задавать в конструкторе.

С помощью модуля "Пробки" можно отображать на карте слой дорожных событий, а также показывать в панели пробок дополнительный переключатель **Дорожные события**. Переключатель позволяет пользователю карты самостоятельно включать и отключать слой дорожных событий. Добавить слой дорожных событий и переключатель можно либо в конструкторе:

```
var traffic = new YMaps.Traffic.Control({ showInfoSwitcher: true },
{ infoLayerShown: true });
```

либо с помощью методов `setOptions()` и `setState()`:

```
traffic.setOptions({ showInfoSwitcher: true });
traffic.setState({ infoLayerShown: true });
```

В примере, рассмотренном в листинге 3.47, создается карта, подключается слой пробок и включается показ пробок и дорожных событий.

Листинг 3.47

```
// создание карты
var map = new YMaps.Map(YMaps.jQuery('#YMapsID')[0]),
// создание элемента управления "Пробки"
traffic = new YMaps.Traffic.Control({
  showInfoSwitcher: true, // В кнопке переключатель "Дорожные события"
}, {
  infoLayerShown: true   // Показывать слой дорожных событий
});
// инициализация карты
map.setCenter(new YMaps.GeoPoint(37.61,55.75), 10);
// добавление элемента управления "Пробки" на карту
map.addControl(traffic);
// включение показа пробок
traffic.show();
```

3.6. Пользовательские карты

3.6.1. Создание пользовательского слоя карты

Чтобы создать новый слой с изображением карты, необходимо использовать класс `YMaps.Layer`. В конструктор класса передается источник данных для карты. Источник данных поставляет тайлы (привязанные к координатам изображения, нарезанные на фрагменты размером 256×256 пикселов) для слоя, в зависимости от текущих координат центра карты и коэффициента масштабирования. Для подготовки тайлов для источника данных рекомендуется использовать приложение API Яндекс.Карт "Подготовка слоя тайлов" *(см. разд. 3.6.2)*.

Чтобы создать источник данных необходимо использовать класс `YMaps.TileDataSource`. Конструктор класса принимает следующие параметры:

- `tileUrlTemplate` — шаблон, по которому строится URL тайла;
- `isTransparent` — флаг, указывающий, что тайлы прозрачны;
- `smoothZoomEnabled` — флаг, разрешающий плавное масштабирование тайлов.

В шаблоне tileUrlTemplate можно использовать следующие специальные конструкции:

- `%d` — заменяется числом от 1 до 4, в зависимости от номера тайла. Используется для распределения нагрузки между несколькими доменами;
- `%c` — заменяется на `x&y&z`, где `x` — номер тайла по горизонтали, `y` — номер тайла по вертикали, `z` — коэффициент масштабирования.

Рассмотрим на примере ниже поэтапное создание и размещение слоя.

- создать источник данных, например:

```
var myData = new YMaps.TileDataSource("", false, true);
```

- перекрыть метод `getTileUrl()`, чтобы задать новые правила формирования URL тайла по шаблону, в зависимости от тайловых координат и коэффициента масштабирования, например:

```
myData.getTileUrl = function (tile, zoom) {
return "http://mt.gmapuploader.com/tiles/FVSH1JsvdT/tile-" + zoom +
  "-" +(tile.y * Math.pow(2, zoom) + tile.x) + ".jpg"; }
```

- создать слой карты, который будет получать тайлы из источника данных и добавить его на карту, например:

```
var myLayer = new YMaps.Layer(myData);
map.addLayer(myLayer);
```

- с помощью метода `add()` поместите созданный слой в хранилище слоев карты YMaps.Layers, например:

```
// функция, создающая экземпляры слоя (конструктор)
var myLayer = function () { return new YMaps.Layer(myData) }
// добавить слой в хранилище
YMaps.Layers.add("test#layer", myLayer);
```

- чтобы получить слой из хранилища по ключу, необходимо использовать метод `get()` класса `YMaps.Layers`, например:

```
var myLayer = YMaps.Layers.get("test#layer");
```

3.6.2. Подготовка тайлов для пользовательского слоя карты

API Яндекс.Карт позволяет накладывать пользовательские слои поверх слоев географической карты. Пользовательские слои могут использоваться, например, для обозначения зоны покрытия сети связи, или схемы расположения зданий на местности, или при показе карты местности, не имеющей подробной схемы в сервисе Яндекс.Карт. Кроме того, API может использоваться и для отображения на сайте

карт и планов, не имеющих привязки к земной поверхности: например, карт игровых миров, фотографий, поэтажных планов сооружений или планов местности.

Для того чтобы использовать изображение в качестве пользовательского слоя Яндекс.Карт, его необходимо предварительно подготовить. Подготовка изображения включает в себя два этапа:

- привязка изображения к географическим координатам (совмещение изображения с картой) — привязка включает в себя поворот, масштабирование и морфинг изображения таким образом, чтобы на каждом масштабном уровне пиксельные координаты заданных геоточек на изображении совпадали с пискельными координатами этих же геоточек в массиве растровых данных Яндекс.Карт;
- формирование наборов тайлов — привязанное к координатам изображение требуется нарезать на фрагменты (тайлы) размером 256×256 пикселов. Каждый тайл хранится в отдельном файле (JPEG или PNG). Для каждого значения коэффициента масштабирования должен быть сформирован свой набор тайлов.

Приложение "Подготовка слоя тайлов" позволяет автоматизировать процесс подготовки произвольного изображения для показа на веб-странице с помощью API Яндекс.Карт. Скачать приложение можно, зайдя на страницу: **http://api.yandex.ru/maps/jsapi/doc/dg/concepts/ymapstiler.xml**.

Приложение позволяет полностью автоматизировать решение следующих задач:

- нарезка тайлов с привязкой по координатам — на изображение наносятся точки привязки, устанавливается соответствие пиксельных координат географическим, изображение перепроецируется под привязку и нарезается на тайлы. Полученные тайлы могут использоваться в качестве источника данных для пользовательского слоя карты (см. `YMaps.TileDataSource`);
- нарезка тайлов без привязки по координатам — изображение сразу нарезается на тайлы для отображения с помощью API (например, для просмотра фотографий и планов);
- создание кода HTML-страницы с готовой картой для размещения на сайте.

Работу в приложении мы рассмотрим при подготовке слоя карт для проекта в *разд. 4.3.1*.

ГЛАВА 4

Примеры использования в проектах API Яндекс.Карт

В предыдущей главе мы познакомились с работой API Яндекс.Карт. В этой главе создадим несколько практических проектов, использующих API Яндекс.Карт. Автор является приверженцем создания сайтов без перезагрузки страницы, поэтому при создании проектов будет использована библиотека xAjax, которая была подробно рассмотрена в *главе 2*. Создадим следующие проекты:

- полноценный каталог предприятий России с показом местонахождения предприятия на Яндекс.Карте;
- программу полного учета заказов для такси с использованием маршрутизатора — сервиса автоматического прокладывания маршрутов на Яндекс.Картах;
- создание у пользовательской карты маленького города с возможностью нанесения на карту (и редактирования) объектов с использованием API Яндекс.Карт;
- отображение карты местности с несколькими слоями пользовательских карт населенных пунктов с выбором нужного участка, отображение на них меток объектов с информацией об объекте, фотографий или swf-роликов;
- панель администратора для предыдущего проекта с возможностью создания меток, редактирования информации о метках и местоположения меток в проекте без перезагрузки страницы с использованием API Яндекс.Карт.

Приведен код и структура баз данных этих проектов, что поможет вам их использовать целиком или частично в своих проектах.

4.1. Каталог предприятий

В качестве первого примера создадим простенький сайт каталога предприятий России, структурированный по регионам и видам деятельности. В каталоге создадим форму поиска нужных предприятий по регионам и видам деятельности. Вывод предприятий будем осуществлять постранично. Для каждого предприятия будем выводить следующие данные:

- название предприятия;
- адрес предприятия;

- контактные телефоны;
- факс;
- адрес электронной почты;
- адрес сайта;
- виды деятельности.

Кроме этого, возможность посмотреть местонахождение предприятия на карте Яндекса. В сети можно посмотреть данный проект по адресу **http://yp.bazakatalogov.ru**. Вид представлен на рис. 4.1.

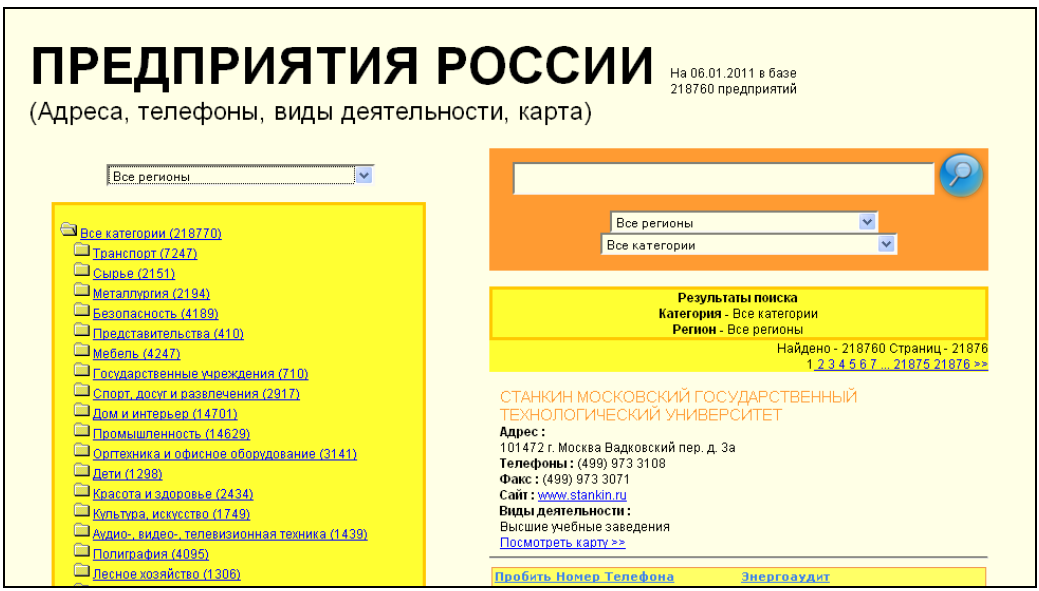


Рис. 4.1. Главная страница каталога предприятий

4.1.1. Проектирование базы данных сайта

Чтобы создать более-менее информативный сайт каталога предприятий, необходимо иметь достаточно большую базу самих предприятий. Если вы внимательно посмотрите на страницу сайта **http://yp.bazakatalogov.ru**, то увидите сообщение о количестве предприятий в базе — более 219 000. Эту базу я взял на просторах Интернета, и грех было не создать подобный сайт. База была в очень неудобном формате данных — *.mdb — множество запутывающих таблиц, пришлось писать программу многоступенчатого парсинга в mysql, поэтому получились следующие таблицы:

- `regions` — регионы России;
- `firms` — предприятия;
- `kategory` — виды деятельности;

- `group_kategory` — показывает подчиненность категорий друг другу (нужна для того, чтобы при выборе верхних категорий видов деятельности показывались все предприятия, входящие во все вложенные категории);
- `link_partners` — ссылки на сайты партнеров (создавал, т. к. брать из базы данные при формировании ссылок на сайты партнеров более удобно).

Рассмотрим подробно структуру таблиц.

Таблица `regions` содержит сведения о регионах России. Структура таблицы `regions` следующая:

- `id` — первичный ключ;
- `name` — наименование региона;
- `sort` — поле для сортировки;
- `visible` — видимость записи:
  - `yes` — видима;
  - `no` — скрыта.

Я думаю, с таблицей `regions` все понятно. Листинг 4.1 содержит дамп для создания таблицы `regions`.

> ***Примечание***
>
> На прилагаемом компакт-диске содержится дамп для создания и заполнения данными таблиц базы данных для этого примера (файл glava04\site1.sql).

Листинг 4.1

```
CREATE TABLE `regions1` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`name` varchar(255) default NULL,
`sort` int(2) NOT NULL,
`visible` set('yes', 'no') NOT NULL default 'yes',
PRIMARY KEY (`id`),
FULLTEXT KEY `keyword` (`name`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

Рассмотрим таблицу `firms` — предприятия. Структура таблицы `firms` следующая:

- `id` — первичный ключ;
- `name` — наименование предприятия;
- `info` — поле содержит практически всю информацию о предприятии в формате HTML (адрес, телефоны, факс, e-mail, адрес сайта);
- `id_kategory` — категория (вид деятельности) предприятия;
- `id_region` — регион предприятия;
- `address` — адрес предприятия;
- `phone` — телефоны;

- `email` — электронная почта;
- `url` — сайт;
- `visible` — видимость предприятия:
  - `yes` — показывать;
  - `no` — скрывать.

Листинг 4.2 содержит дамп для создания таблицы `firms`.

Листинг 4.2

```
CREATE TABLE `firms` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`name` varchar(255) default NULL,
`info` text,
`id_kategory` int(11) default '0',
`id_region` int(11) default '0',
`adress` varchar(255) default NULL,
`phone` varchar(255) default NULL,
`email` varchar(255) default NULL,
`url` varchar(255) default NULL,
`visible` set('yes', 'no') NOT NULL default 'yes',
PRIMARY KEY (`id`),
FULLTEXT KEY `title` (`name`, `info`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

Таблица `kategory` — список всех видов деятельности и подчиненность категорий друг другу. Структура таблицы `kategory` следующая:

- `id` — первичный ключ;
- `name` — наименование категории (вида деятельности);
- `id_parent` — идентификатор родительской категории;
- `sort` — для сортировки;
- `nn` — количество предприятий во всех вложенных категориях для данной категории;
- `visible` — видимость категории:
  - `yes` — видима;
  - `no` — скрыта.

Листинг 4.3 содержит дамп для создания таблицы `kategory`.

Листинг 4.3

```
CREATE TABLE `kategory` (
`id` int(5) NOT NULL AUTO_INCREMENT,
`name` varchar(255) default NULL,
```

```
`id_parent` int(5) default NULL,
`sort` int(4) default NULL,
`nn` int(8) default '0',
`visible` set('yes', 'no') NOT NULL default 'yes',
PRIMARY KEY (`id`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

Следующая таблица — `group_kategory`. При выборе категории (вида деятельности) каталога отображаются предприятия, входящие во все нижние категории. Эти данные берутся из таблицы `group_kategory` — список пар всех верхних и вложенных до любого уровня категорий. Структура таблицы `group_kategory` следующая:

- `id` — первичный ключ;
- `top_level` — идентификатор категории верхнего уровня;
- `bottom_level` — идентификатор категории нижнего уровня.

Листинг 4.4 содержит дамп для создания таблицы `group_kategory`.

Листинг 4.4

```
CREATE TABLE `group_kategory` (
`id` int(5) NOT NULL AUTO_INCREMENT,
`top_level` int(5) default NULL,
`bottom_level` int(5) default NULL,
PRIMARY KEY (`id`)
) ENGINE = MYISAM AUTO_INCREMENT = 2863DEFAULT CHARSET = cp1251
```

И последняя таблица — `link_partners` — ссылки на сайты партнеров. Структура ее такова:

- `id` — первичный ключ;
- `name` — название ссылки;
- `link` — адрес перехода по ссылке;
- `sort` — для сортировки;
- `visible` — видимость ссылки:
  - `yes` — видима;
  - `no` — скрыта.

Листинг 4.5 содержит дамп для создания таблицы `link_partners`.

Листинг 4.5

```
CREATE TABLE `link_partners` (
`id` int(5) NOT NULL AUTO_INCREMENT,
`name` varchar(50) NOT NULL,
`link` varchar(50) NOT NULL,
```

```
`sort` int(3) NOT NULL,
`visible` set('yes', 'no') NOT NULL default 'yes',
UNIQUE KEY `id` (`id`),
KEY `sort` (`sort`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

4.1.2. Программирование сайта

Приступим к программированию этого сайта. Будем использовать библиотеку xAjax, рассмотренную нами в *главе 1*, что позволит нам создать сайт без перезагрузки страницы. При обращении к странице сайта выполняется файл index.php, который выполняет следующее:

- загружает библиотеку xAjax;
- загружает API Яндекс.Карт. Так как в момент загрузки страницы с картой никаких действий производить не требуется, будем загружать API не полностью, при этом загружается специальный инициализирующий скрипт, дающий возможность подгрузить API в любой момент с помощью метода `YMaps.load()`. Для этого в URL скрипта загрузки добавим параметр `loadByRequire = 1`:

```
<script src="http://api-
maps.yandex.ru/1.1/index.xml?loadByRequire=1&key=API-ключ"
type="text/javascript"></script>
```

- формирует шапку сайта;
- создает пустые блоки `div` для будущей загрузки контента;
- вызывает xAjax-функцию для загрузки начального контента.

Содержимое файла index.php представлено в листинге 4.6.

Листинг 4.6

```
... ... ...
<html>
<head>
... ... ...
  <script type="text/javascript">
    var map, geoResult;
    function loadypmap()
    { map = new YMaps.Map(document.getElementById("ypmap"));
      map.setCenter(new YMaps.GeoPoint(37.64, 55.76), 3);
      map.addControl(new YMaps.TypeControl());}
    function showAddress (value)
    { map.removeOverlay(geoResult);
      var geocoder = new YMaps.Geocoder(value, {results: 1, boundedBy:
                     map.getBounds()});
      YMaps.Events.observe(geocoder, geocoder.Events.Load, function () {
      if (this.length())
```

```
      { geoResult = this.get(0);
        map.setBounds(geoResult.getBounds());
        var s = new YMaps.Style();
        s.balloonContentStyle = new YMaps.BalloonContentStyle(
          new YMaps.Template("<div style=\"color:#0A0;
          background-color:#ABC;width:200px;\">
          $[description]</div>")
        );
        var geoCoords = geocoder.get(0).getGeoPoint();
        geoPlacemark = new YMaps.Placemark(geoCoords,{mapAutoPan: true,
                       style: s});
        var content="<div style=\"color:#0A0;
        background-color:#ABC;width:200px;\">
        <b>"+value1+"</b><br><center><br></center> "+value2+" </div>";
        geoPlacemark.setBalloonContent(content);
        map.addOverlay(geoPlacemark);
        geoPlacemark.openBalloon();}
      else
      { alert("Ничего не найдено"); }
    });}
  </script>
... ... ...
 </head>
<body name=body1 id=body1 onload="xajax_Ini();" >
<div id=header1></div>
<div>
<!-- шапка -->
<table><tr><td><div id=header2>
  ... ... ...
  </div></td>
  <td><div align=right> ... ... ... </div>td></tr></table><br><div>
<!-- ----- рабочие страницы 2 ----------------- -->
<table width=100%><tr>
<!-- левое -->
    <td width=1% ></td><td width=40% valign=top>
     <!-- регион -->
       <div id=leftcaption1><br></div>
       <div id=left1><table></table></div>
     <!-- категории -->
       <div id=leftcaption2><br></div>
       <div id=left2><table></table></div></td><td width=5%></td>
<!-- среднее -->
    <td width=50% valign=top>
     <!-- поиск -->
      <div id=rightcaption1></div>
      <div id=right1><table></table></div>
```

```
     <!-- результаты поиска -->
      <div id=rightcaption2>Результаты поиска</div>
      <div id=right21><table></table></div>
      <div id=right22><table></table></div>
      <div id=google1></div>
      <div id=right23><table></table></div>
      <div id=google2></div><br>
      <div id=right24><table></table></div>
      <div id=right29><table></table></div>
    </td><td width=3%></td></tr></table>
<!-- карта Яндекс -->
<div id=map>... ... ...<div id="ypmapaddress"></div>
    <div id="ypmap" style="width:600px;height:400px"></div><br>
</div>
<!-- ----- всплывающие страницы -->
<div id='windowdop'>дополнительное окно</div>
<div id='windowindicator'><img src='img/zagruzka.gif'></div>
<!-- подвал -->
<div id=bottom>
... ... ...
</div>
</body>
</html>
```

Здесь требуются некоторые пояснения. Вот распределение назначения блоков:

- `left1` — форма выбора региона;
- `left2` — категории видов деятельности;
- `right1` — форма поиска;
- `rightcaption2` — информер параметров фильтра (регион, категория);
- `right21` — верхний навигатор страниц;
- `right22`, `right23`, `right24` — блоки для вывода результатов поиска (разделение существует, чтобы в результаты поиска расположить рекламу);
- `google1`,`google2` — для встраивания рекламы (Google.Adsence или на ваше усмотрение);
- `right29` — нижний навигатор страниц;
- `windowindicator` — поиск по базе может занимать значительное время, и на это время визуализируется блок с gif-картинкой индикатора загрузки;
- `map` — блок для размещения Яндекс.Карты.

Программирование дерева категорий видов деятельности

Создадим список категорий произвольной вложенности. Он будет использоваться для выбора предприятий по видам деятельности. При нажатии на значок нераскрытой категории будем выводить список вложенных категорий (xAjax-функция

`Open_Kategory()`), при щелчке по ссылке раскрытой категории будем сворачивать ее (xAjax-функция `Close_Kategory()`). Эти функции расположены в файле prgkategory\open_close_kategory.php, содержимое которого приведено в листинге 4.7.

Листинг 4.7

```
<?php
// открыть папку категории
function Open_Kategory($Id)
{ $objResponse = new xajaxResponse();
  $content=f_open_kategory($Id);
  $objResponse->assign("kategory".$Id,"innerHTML",$content);
  return $objResponse;
}
// закрыть папку категории
function Close_Kategory($Id)
{ $objResponse = new xajaxResponse();
  $content=f_close_kategory($Id);
  $objResponse->assign("kategory".$Id,"innerHTML",$content);
  return $objResponse;
}
?>
```

Функции, формирующие контент для вывода в блок `left2`, находятся в файле prgkategory\function_kategory.php (листинг 4.8).

Листинг 4.8

```
<?php
/// Выдача дерева категорий
//********Раскрытие каталога ******************
// показать вложенные подкаталоги
function f_open_kategory($Id)
{ require_once("mybaza.php");
  $text1="";
  // получение списка вложенных категорий
  $query1="SELECT name,id,nn FROM kategory WHERE id='".$Id."'
          && visible='yes' ";
  $rez1=mysql_query($query1);
  $row1=mysql_fetch_assoc($rez1);
  $text1.="<a href='javascript:void();'
           onclick='xajax_Close_Kategory(".$Id.");'>
           <img src='img/close_dir.ico'></a>
           <a href='javascript:void();'
           onclick='xajax_Close_Kategory(".$Id.");
           document.forms.FormSelectRegion.selectkategory.value=".$Id.";
           indicator();
```

```
          xajax_View_Firms_Kategory(xajax.getFormValues
          (\"FormSelectRegion\"));'>".$row1[name]."
          (".$row1[nn].")</a>";
  $query2="SELECT name,id,nn FROM kategory WHERE id_parent='".$Id."'
          && visible='yes' ";
  $rez2=mysql_query($query2);
  while($row2=mysql_fetch_assoc($rez2))
  { $query3="SELECT id FROM kategory WHERE id_parent='".$row2[id]."'
           && visible='yes' ";
    $rez3=mysql_query($query3);
    $text1.="<div class='menu' id='kategory".$row2[id]."'>
           <a href='javascript:void();'
           onclick='xajax_Open_Kategory(".$row2[id].");'>
           <img src='img/open_dir.ico'></a>
           <span><a href='javascript:void();'
           onclick='xajax_Open_Kategory(".$row2[id].");
           document.forms.FormSelectRegion.selectkategory.value=
           ".$row2[id].";
           indicator();
           xajax_View_Firms_Kategory(xajax.getFormValues
           (\"FormSelectRegion\"));'>
           ".$row2[name]." (".$row2[nn].")</a></div>";
  }
  return $text1;
}
//********Закрытие каталога ******************
function f_close_kategory($Id)
{ require_once("mybaza.php");
  $text1="";
  // получение списка вложенных категорий
  $query1="SELECT name,id,nn FROM kategory WHERE id='".$Id."'
         && visible='yes' ";
  $rez1=mysql_query($query1);
  $row1=mysql_fetch_assoc($rez1);
  $text1.="<a href='javascript:void();'
           onclick='xajax_Open_Kategory(".$Id.");'>
          <img src='img/open_dir.ico'></a>
          <a href='javascript:void();'
          onclick='xajax_Open_Kategory(".$Id.");
          document.forms.FormSelectRegion.selectkategory.value=".$Id.";
          indicator();
          xajax_View_Firms_Kategory(xajax.getFormValues
          (\"FormSelectRegion\"));'>
          ".$row1[name]." (".$row1[nn].")</a>";
  return $text1;
}
?>
```

Название категории (вида деятельности) мы выдаем в контент в виде ссылки, но при этом, в отличие от ссылки, при нажатии на значок происходит не только раскрытие категории, но и выполняются поиск и выдача в контент списка всех предприятий, входящих во вложенные категории (xAjax-функция `View_Firms_Kategory()`). Еще вызывается и JavaScript-функция `indicator()`, расположенная в заголовочной части файла index.php, визуализирующая индикатор загрузки (листинг 4.9).

Листинг 4.9

```
<script type="text/javascript" >
  function indicator()
  { document.getElementById("windowindicator").style.display="block";
    return; }
</script>
```

Вывод списка предприятий категории

На сайте представлен не только поиск предприятия из формы поиска, но и фильтр для вывода предприятий по категориям и регионам. В блоке `left1` форма выбора региона. Визуально она представлена `select`-полем для выбора региона. Кроме этого, в форме `FormSelectRegion` есть и два скрытых поля, содержащих значение выбранной категории `id=selectkategory` и страницы показа результатов `id=selectpage`. При выборе категории значение `id=selectkategory` устанавливается равным `id` выбранной категории. При выборе категории или региона вызывается xAjax-функция `View_Firms_Kategory()`, в качестве параметров передаются значения полей формы `FormSelectRegion`. Функция `View_Firms_Kategory()` расположена в файле prgfirms\view_firms_kategory.php, содержимое его приведено в листинге 4.10.

Листинг 4.10

```
<?php
// Просмотр фирм региона, категории
function View_Firms_Kategory($Id)
{ $objResponse = new xajaxResponse();
  $content=f_view_firms_kategory($Id);
  // заголовок
  $zag="<b>Результаты поиска</b>";
  $zag.="<br><b>Категория – </b>".f_string_kategory($Id[selectkategory]);
  if($Id[selectregion]>0)
    $zag.="<br><b>Регион – </b>".mysql_result(mysql_query(
    "SELECT name FROM regions WHERE id='".$Id[selectregion]."' "),0);
  else
    $zag.="<br><b>Регион – </b> Все регионы";
  $objResponse->assign("rightcaption2","innerHTML",$zag);
```

```
  // контент
  $objResponse->assign("right22","innerHTML",$content[1]);
  $objResponse->assign("right23","innerHTML",$content[2]);
  $objResponse->assign("right24","innerHTML",$content[3]);
  // страницы
  $objResponse->assign("right21","innerHTML",$content[9]);
  $objResponse->assign("right29","innerHTML",$content[9]);
  $objResponse->script(
      "document.getElementById('header1').scrollIntoView();");
  $objResponse->script(
     "document.getElementById('windowindicator').style.display='none';");
  return $objResponse;
}
?>
```

Функция, формирующая контент для вывода предприятий выбранной категории и региона, находится в файле prgfirms\function_view_firms_kategory.php. Контент создается отдельно для трех блоков, разделенных рекламными блоками, и навигатора страниц. Содержимое файла представлено в листинге 4.11.

Листинг 4.11

```
<?php
function f_view_firms_kategory($Id)
{ require_once("mybaza.php");
  $text=array();$text11="";$text12="";$text13="";
  $query0="SELECT COUNT(id) FROM firms WHERE id>0 ";
  $region=$Id[selectregion];$kategory=$Id[selectkategory];
  if($region>0)
    $query0.=" && id_region='".$region."' ";
  $query01="SELECT id FROM kategory WHERE id_parent=0 ";
  $rez01=mysql_query($query01);
  $high_kategory=mysql_result($rez01,0);
  if($kategory!=$high_kategory)
  { // список вложенных
    $in="";
    $query02="SELECT bottom_level FROM group_kategory
              WHERE top_level='".$kategory."' ";
    $rez02=mysql_query($query02);
    while($row02=mysql_fetch_row($rez02))
      $in.=$row02[0].",";
    $in=substr($in,0,strlen($in)-1);
    // добавить в запрос
    $query0.=" && id_kategory IN (".$in.") ";
  }
  $rez0=mysql_query($query0);
  $count=mysql_result($rez0,0);
```

```
  $pages=ceil($count/NN1);
  $page=min($Id[selectpage],$pages);$poz=($page-1)*NN1;
  if($count>0)
  { $query0.=" LIMIT ".$poz.", ".NN1."";
    $query1=str_replace("COUNT(id)","*",$query0);
    $rez1=mysql_query($query1); $j=0;
    while($row1=mysql_fetch_assoc($rez1))
    { $j++;
      // если есть email или www – поменять в поле info на ссылки
      if(strlen(trim($row1[email]))>0)
      { $link_email="<a href='javascript:void();' onclick='
                    xajax_Form_Email(".$row1[id].");'>
                    ".$row1[email]."</a>";
        $row1[info]=str_replace($row1[email],$link_email,$row1[info]);
      }
      if(strlen(trim($row1[url]))>0)
      { $row1[url]=str_replace("http://","",$row1[url]);
        $link_url="<a href='http://".$row1[url]."'
                   target='_blank'>".$row1[url]."</a>";
        $row1[info]=str_replace($row1[url],$link_url,$row1[info]);
      }
      // распределение по блокам (между рекламой)
      if($j>NN12)
      { $text13.="<div class='view_firm'>".$row1[name]."</div>";
        $text13.="<div class='view_firm_info'>".$row1[info]."</div>";
        $text13.="<div class='link_firm_map'><a href='javascript:void()'
                  onclick='xajax_View_Map(".$row1[id].");'>Посмотреть
                           карту >></a></div>";
        $text13.="<hr></hr>";
      }
      elseif($j>NN11)
      {
      ... ... ...
      }
      else
      {
      ... ... ...
      }
    }
    // список ссылок перехода по страницам
   ... ... ...
  // рекламные ссылки
  ... ... ...
  // возврат значений
  $text[1]=$text11; $text[2]=$text12; $text[3]=$text13;$text[9]=$text2;
  return $text;
}
?>
```

В информере параметров фильтра выводится путь от корня до текущей категории видов деятельности. Формирование строки пути осуществляет функция `f_string_kategory()`, расположенная в файле prgkategory\function_string_kategory.php, содержимое которого представлено в листинге 4.12.

Листинг 4.12

```
<?php
// Создание строки пути категории
function f_string_kategory($Id)
{ require_once("my.php");
  require_once("mybaza.php");
  $text1="";
  $query0="SELECT * FROM kategory WHERE id='".$Id."'";
  $rez0=mysql_query($query0);
  $row0=mysql_fetch_assoc($rez0);
  $text1=$text1.$row0[name];
  $id_parent=$row0[id_parent];
  while($id_parent>0)
  { $query0="SELECT * FROM kategory WHERE id='".$id_parent."'";
    $rez0=mysql_query($query0);
    $row0=mysql_fetch_assoc($rez0);
    $text1=$row0[name]."->".$text1;
    $id_parent=$row0[id_parent];
  }
  return $text1;
}
?>
```

Форма поиска предприятий

Форма поиска предприятий находится в правой верхней области страницы (см. рис. 4.1) и позволяет получить список предприятий по наименованию (строка поиска присутствует в наименовании) из выбранного региона (или всех) и выбранного вида деятельности (либо всех). Программа формирования контента для формы поиска находится в файле prgsearch\function_form_search.php, содержимое которого представлено в листинге 4.13.

Листинг 4.13

```
<?php
// Форма поиска
function f_form_search()
{ require_once("mybaza.php");
  $text1="";
  $text1.="<form id='FormSearchFirms' method='post'
                                                action='javascript:void(null);'
          onsubmit='indicator();
```

```
          xajax_View_Search_Firms(xajax.getFormValues(\"FormSearchFirms\"));'
          enctype=\"multipart/form-data\";>";
  $text1.="<a href='javascript:void(null)' onclick='
          indicator();xajax_View_Search_Firms(xajax.getFormValues
                                                  (\"FormSearchFirms\"));
          '><img src='img/search.png' align=right></a>";
  // страница
  $text1.="<input type='hidden' id='searchfirmspage'
            name='searchpage' value='1'><br>";
  // название
  $text1.="<input class=big type='text' id='searchfirmname'
            name='searchfirmname' value='' size=40 maxlength=30>";
  // регион
  $query1="SELECT id,name FROM regions WHERE visible='yes' ORDER BY sort ASC ";
  $rez1=mysql_query($query1);
  $text1.="<div style='text-align:center'><br>
           <select name=searchselectregion>";
  $text1.="<option value='0' selected>Все регионы";
  while($row1=mysql_fetch_assoc($rez1))
  { $text1.="<option value=".$row1[id].">".$row1[name]; }
  $text1.="</select>";
  // Категория
  $text1.="<div id=divsearchselectkategory>
           <select name=searchselectkategory onchange='
   xajax_Search_Select_Kategory(xajax.getFormValues(\"FormSearchFirms\"));'>";
  $query21="SELECT id,name FROM kategory WHERE id_parent='0' ";
  $rez21=mysql_query($query21);
  $row21=mysql_fetch_assoc($rez21);
  $text1.="<option value='".$row21[id]."' style=\"color:red;\"
selected>".$row21[name]."";
  $query22="SELECT id,name FROM kategory WHERE id_parent='".$row21[id]."' ";
  $rez22=mysql_query($query22);
  while($row22=mysql_fetch_assoc($rez22))
  { $text1.="<option value=".$row22[id].">".$row22[name]; }
  $text1.="</select></div></div>";
  $text1.="<div><br></div>";
  $text1.="</form>";
  return $text1;
}
?>
```

Поле `select` выбора вида деятельности при выборе категории, имеющей вложенные категории, подгружается списком вложенных категорий. Для перехода в верхний уровень имеется позиция **вверх** (рис. 4.2). Подгрузка новых опций в поле `select` происходит вызовом xAjax-функции `Search_Select_Kategory()` по событию `onchange`. xAjax-функция `Search_Select_Kategory()` расположена в файле prgsearch\search_select_kategory.php, содержимое которого представлено в листинге 4.14.

Листинг 4.14

```
<?php
function Search_Select_Kategory($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  $id_kategory=$Id[searchselectkategory];
  $text1.="<select name=searchselectkategory onchange='
  xajax_Search_Select_Kategory(xajax.getFormValues(\"FormSearchFirms\"));'>";
  $query21="SELECT id,name,id_parent FROM kategory WHERE id=".$id_kategory." ";
  $rez21=mysql_query($query21);
  $row21=mysql_fetch_assoc($rez21);
  if($row21[id_parent]>0)
  { $text1.="<option value='".$row21[id_parent].
            "' style=\"color:red;\" >вверх";
    $text1.="<option value='".$row21[id].
            "' style=\"color:red;\" selected>".$row21[name];
  }
  else
  { $text1.="<option value='".$row21[id].
            "' style=\"color:red;\" selected>".$row21[name];
    $sel="";
  }
  $query22="SELECT id,name FROM kategory WHERE id_parent='".$row21[id]."' ";
  $rez22=mysql_query($query22);
  while($row22=mysql_fetch_assoc($rez22))
  { $text1.="<option value=".$row22[id]." >".$row22[name]; }
  $text1.="</select>";
  $objResponse->assign("divsearchselectkategory","innerHTML",$text1);
  return $objResponse;
}
?>
```

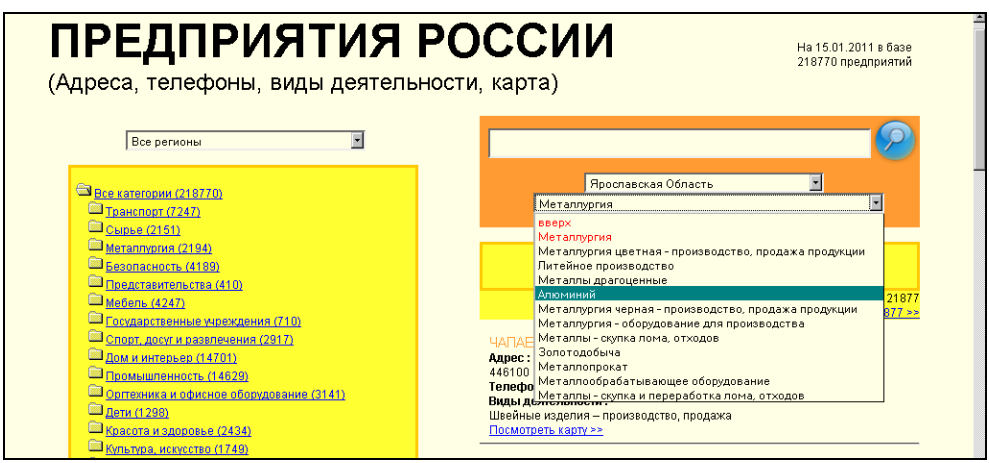


Рис. 4.2. Подгрузка в `select`-список перечня вложенных категорий

Вывод результатов поиска

Вывод результатов поиска осуществляет xAjax-функция `View_Search_Firms()`, расположенная в файле prgsearch\view_search_firms.php, содержимое которого представлено в листинге 4.15.

Листинг 4.15

```
<?php
// Просмотр фирм по поиску
function View_Search_Firms($Id)
{ $objResponse = new xajaxResponse();
  $content=f_view_search_firms($Id);
  // заголовок
  $zag="<b>Результаты поиска</b>";
  $zag.="<br><b>Категория – </b>".f_string_kategory($Id[searchselectkategory]);
  if($Id[searchselectregion]>0)
    $zag.="<br><b>Регион – </b>".mysql_result(mysql_query("SELECT name
    FROM regions WHERE id='".$Id[searchselectregion]."' "),0);
  else
    $zag.="<br><b>Регион – </b> Все регионы";
  if(strlen($Id[searchfirmname])>0)
    $zag.="<br><b>Название-</b> содержит ".utftowin($Id[searchfirmname]);
  $objResponse->assign("rightcaption2","innerHTML",$zag);
  // вывод контента
  $objResponse->assign("right22","innerHTML",$content[1]);
  $objResponse->assign("right23","innerHTML",$content[2]);
  $objResponse->assign("right24","innerHTML",$content[3]);
  // страницы навигатора страниц
  $objResponse->assign("right21","innerHTML",$content[9]);
  $objResponse->assign("right29","innerHTML",$content[9]);
  // в зону видимости
  $objResponse->script("document.getElementById(
                'header1').scrollIntoView();");
  // скрыть показ Яндекс карты
  $objResponse->script("document.getElementById(
                'windowindicator').style.display='none';");
  return $objResponse;
}
?>
```

Формирование контента осуществляет функция `f_view_search_firms()`, в которую передаются значения полей формы `FormSearchFirms`. Функция во многом похожа на функцию `f_view_firms_kategory()` (см. листинг 4.11), мы ее здесь приводить не будем. Расположена она в файле prgsearch\function_view_search_firms.php, найти вы его сможете на прилагаемом компакт-диске (см. папку glava_04\4-1 на компакдиске).

Программа начальной загрузки

При открытии главной страницы сайта не происходит загрузки контента в блоки. Загрузку контента осуществляет xAjax-функция `Ini()`, вызов которой осуществляется по событию `onload` страницы index.php. Функция `Ini()` расположена в файле ini.php, содержимое которого представлено в листинге 4.16.

Листинг 4.16

```
<?php
// Инициализация при открытии
function Ini()
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");

  //************ Выбор региона *****************
  $query11="SELECT id,name FROM regions WHERE visible='yes' ORDER BY sort ASC";
  $rez11=mysql_query($query11);
  $content="<form id='FormSelectRegion' method='post'
            action='javascript:void(null);'>";
  $content.="<select name=selectregion onchange='indicator();
          xajax_View_Firms_Kategory(xajax.getFormValues(
          \"FormSelectRegion\"));'>";
  $content.="<option value='0' selected>Все регионы";
  while($row11=mysql_fetch_assoc($rez11))
  { $content.="<option value=".$row11[id].">".$row11[name]; }
  $content.="</select>";
  $query2="SELECT id,name,nn FROM kategory WHERE id_parent=0 ";
  $rez2=mysql_query($query2);
  $row2=mysql_fetch_assoc($rez2);
  $content.="<input type=hidden name=selectkategory
            id=selectkategory value='".$row2[id]."'>";
  $content.="<input type=hidden name=selectpage
            id=selectpage value='1'>";
  $content.="</form>";
  $objResponse->assign("left1","innerHTML",$content);
//************ Категории ***************************
  //$query2="SELECT id,name,nn FROM kategory WHERE id_parent=0 ";
  //$rez2=mysql_query($query2);
  //$row2=mysql_fetch_assoc($rez2);
  $content="<div class='menu' style='margin-left:0'
          id=kategory".$row2[id].">".$row2[name]."
          (".$row2[nn].")</div>";
  $objResponse->assign("left2","innerHTML","<br>".$content."<br>");
//*************** Раскрыть категорию ******************
  $content=f_open_kategory($row2[id]);
  $objResponse->assign("kategory".$row2[id],"innerHTML",$content);
```

```
//********************** Фильтр ******************
  $content=f_form_search();
  $objResponse->assign("right1","innerHTML",$content);
//********************** Фирмы (все регионы, все рубрики)  ******************
  // заголовок
  $zag="<b>Результаты поиска</b>";
  $zag.="<br><b>Категория — </b>".f_string_kategory($row2[id]);
  $zag.="<br><b>Регион — </b> Все регионы";
  $objResponse->assign("rightcaption2","innerHTML",$zag);

  $arr=array();
  $arr[selectpage]=1;$arr[selectkategory]=$row2[id];
  $content=f_view_firms_kategory($arr);
  // контент
  $objResponse->assign("right22","innerHTML",$content[1]);
  $objResponse->assign("right23","innerHTML",$content[2]);
  $objResponse->assign("right24","innerHTML",$content[3]);
  // страницы
  $objResponse->assign("right21","innerHTML",$content[9]);
  $objResponse->assign("right29","innerHTML",$content[9]);
  // скрыть блок с картой
  $objResponse->script
  ("document.getElementById('windowindicator').style.display='block';");
  $objResponse->script("YMaps.load(loadypmap);");
  $objResponse->script("document.getElementById('header1').scrollIntoView();");
  // скрыть блок с индикатором загрузки
  $objResponse->script(
  "document.getElementById('windowindicator').style.display='none';");
  return $objResponse;
}
?>
```

Я думаю, здесь все понятно. Перейдем к выводу местонахождения предприятия, используя сервис Яндекс.Карт.

4.1.3. Использование API Яндекс.Карт

При выводе списка предприятий для каждого предприятия имеется ссылка **Посмотреть карту** (рис. 4.3). В базе данных нет информации о географических координатах предприятий. Но есть данные об адресе местонахождения данного предприятия. API Яндекс.Карт предоставляет сервис "Геокодирование" — определение географического местоположения объекта по его адресу *(см. разд. 3.5.1)*. При щелчке по ссылке **Посмотреть карту** происходит вызов xAjax-функции `View_Map()`, расположенной в файле prgfirms\view_map.php. Рассмотрим подробнее, что делает эта функция. Сначала из базы данных получаем адрес объекта. Затем скрываем блок с изображением карты (возможно, в момент нажатия ссылки была отображена карта с меткой местонахождения предыдущего объекта). Так как мы загружали API

с загрузкой по требованию *(см. разд. 3.1.2)*, подгружаем API с помощью метода `YMaps.Load()`. Вызываем js-функцию `loadmap()` (листинг 4.17). Она создает новый экземпляр класса карты и привязывает его к контейнеру (блок `ypmap`), устанавливает центр карты и размещает элементы управления.

Листинг 4.17

```
function loadypmap()
{ map = new YMaps.Map(document.getElementById("ypmap"));
    map.setCenter(new YMaps.GeoPoint(37.64, 55.76), 3);
    map.addControl(new YMaps.TypeControl());
}
```

Затем вызывается js-функция `showddress()` (листинг 4.18). Она удаляет с карты предыдущую метку и запускает сервис геокодирования. В случае удачного результата создает объект — оверлей и выводит его на карту, центр устанавливается в месте оверлея.

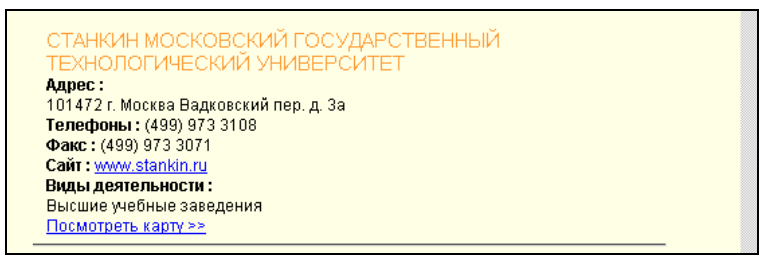


Рис. 4.3. Ссылка на просмотр карты

Листинг 4.18

```
function showAddress (value)
{ map.removeOverlay(geoResult);
  var geocoder = new YMaps.Geocoder(value,
                  {results: 1, boundedBy: map.getBounds()});
  YMaps.Events.observe(geocoder, geocoder.Events.Load, function ()
  { if (this.length())
    { geoResult = this.get(0);
      map.setBounds(geoResult.getBounds());
      var s = new YMaps.Style();
      s.balloonContentStyle = new YMaps.BalloonContentStyle(
        new YMaps.Template("<div style=\"color:#0A0;
        background-color:#ABC;width:200px;\">
        $[description]</div>"));
      var geoCoords = geocoder.get(0).getGeoPoint();
      geoPlacemark = new YMaps.Placemark(geoCoords,
                {mapAutoPan: true, style: s});
```

```
      var content="<div style=\"color:#0A0;background-color:#ABC;
               width:200px;\"><b>"+value1+"</b>
               <br><center><br></center> "+value2+" </div>";
      geoPlacemark.setBalloonContent(content);
      map.addOverlay(geoPlacemark);
      geoPlacemark.openBalloon();
    }
    else
    { alert("Ничего не найдено") }
  });
}
```

Затем блок `ymap` делается видимым (рис. 4.4). Содержимое файла prgfirms\view_map.php представлено в листинге 4.19.

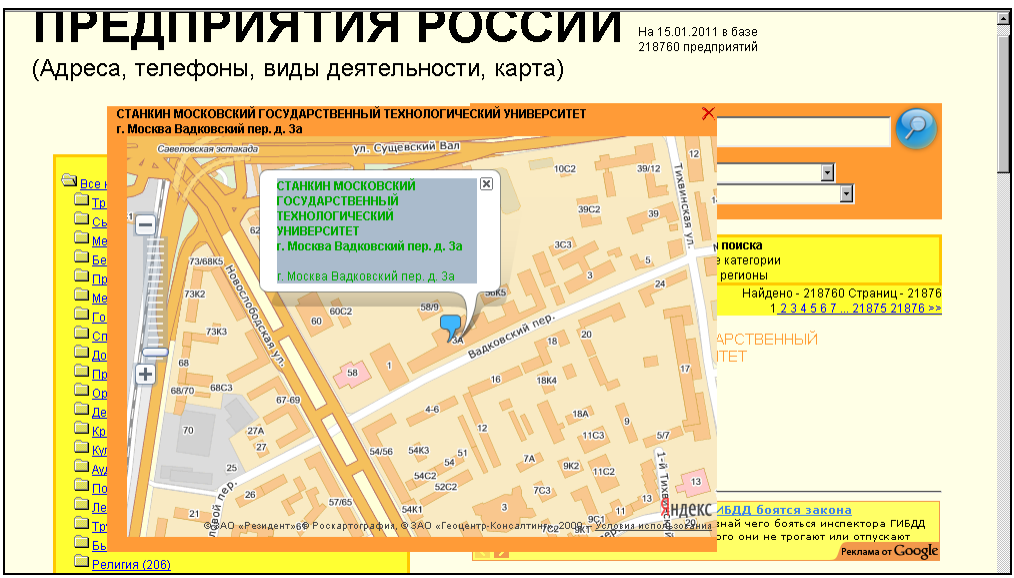


Рис. 4.4. Местоположение объекта на Яндекс.Карте

Листинг 4.19

```
<?php
//////// Посмотреть карту Яндекса ///////////////
function View_Map($Id)
{ $objResponse = new xajaxResponse();
  require_once("my.php");
  require_once("mybaza.php");
  // получить адрес выбранного объекта
  $query1="SELECT name, adress FROM firms WHERE id='".$Id."' ";
  $rez1=mysql_query($query1);
```

```
  $content=mysql_result($rez1,0,"name");
  $address.=mysql_result($rez1,0,"adress");
  $content.="<br>".$address;
  // спрятать предыдущюю карту
  $objResponse->script(
       "document.getElementById('map').style.display='block'");
  // подгрузить API
  $objResponse->script("YMaps.load(loadypmap);");
  //
  $objResponse->assign("ypmapaddress","innerHTML","<b>".$content."</b>");
  // карту в зону видимости
  $objResponse->script("document.getElementById('map').scrollIntoView();");
  // вызвать js-функцию showaddress()
  $objResponse->script("showAddress('".$address."');");
  return $objResponse;
}
?>
```

Вот такой простенький пример использования API Яндекс.Карт в реальном проекте. Данный скрипт вы можете использовать в своих проектах. Любые модификации допустимы и приветствуются. Для размещения примера на своем сайте необходимо получить для ресурса ключ и изменить значение ключа своим в файле my.php.

4.2. Сайт учета заказов такси

Теперь рассмотрим пример посложнее. Создадим сайт — программу учета заказов такси. При этом будем использовать для расчета стоимости поездки сервис автоматического прокладывания маршрутов на Яндекс.Картах. В сети вы можете посмотреть проект по адресу **http://examples-api.bazakatalogov.ru/yandex/taxi**. Вид главной страницы представлен на рис. 4.5.

4.2.1. Проектирование базы данных

Для проекта создадим в базе данных три таблицы:

- `drivers` — водители;
- `cars` — автомобили (водители могут работать на собственных авто либо на автомобилях фирмы);
- `orders` — список всех заказов с указанием маршрута, стоимости и статуса заказа.

Рассмотрим структуру и назначение таблиц.

Таблица `drivers` содержит сведения о водителях данной фирмы. Структура таблицы базы данных следующая:

- `id` — первичный ключ;
- `fio` — ФИО водителя;
- `passport` — данные паспорта;

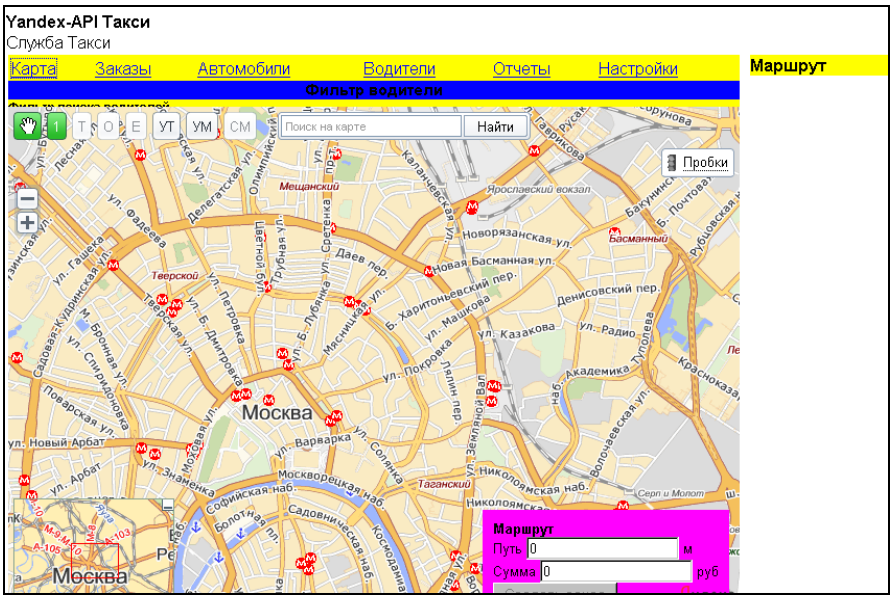


Рис. 4.5. Сайт учета заказов такси

- `address` — адрес;
- `phone` — телефон;
- `prava` — права;
- `data1` — дата поступления на работу;
- `data2` — дата увольнения;
- `status` — статус водителя:
  - `yes` — на работе;
  - `no` — не на работе;
  - `out` — уволен;
  - `firma` — фирма-владелец такси.

Включим в данную базу водителей в качестве записи и саму фирму (`status=firma`). При этом не надо составлять дополнительную базу для привязки машины к владельцу, мы в базе данных автомобилей (`cars`) введем ссылку на владельца автомобиля — на запись из таблицы `drivers` (владельцем авто может быть и сама фирма, и водитель на своей машине). Листинг 4.20 содержит дамп для создания таблицы `drivers`.

Листинг 4.20

```
CREATE TABLE `drivers` (
`id` int(9) NOT NULL AUTO_INCREMENT,
`fio` varchar(50) NOT NULL,
```

```
`passport` varchar(100) NOT NULL,
`address` varchar(100) NOT NULL,
`phone` varchar(50) NOT NULL,
`prava` varchar(30) NOT NULL,
`data1` datetime NOT NULL,
`data2` datetime NOT NULL,
`status` set('yes', 'no', 'out', 'firma') default NULL,
UNIQUE KEY `id` (`id`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

Следующая таблица — `cars`. Она содержит сведения об автомобилях. Ее структура такова:

- `id` — первичный ключ;
- `owner` — идентификатор владельца машины (из таблицы `drivers` — либо фирма, либо таксист со своим автомобилем);
- `mark` — марка автомобиля;
- `number` — номер;
- `id_driver` — идентификатор водителя (из таблицы `drivers` водитель, который выехал на этом автомобиле, если равно 0 — автомобиль свободен);
- `data1` — устанавливает текущую дату и время в момент выхода автомобиля на маршрут и в момент получения заказа от оператора;
- `data2` — устанавливает текущую дату и время в момент окончания работы;
- `status` — статус автомобиля:
  - `yeszakaz` — выполняет заказ;
  - `yes` — автомобиль с водителем, но в данный момент свободен;
  - `no` — автомобиль без водителя;
  - `out` — автомобиль выбыл (поломался, исчез, ушел в ремонт или на металлолом и т. д.).

Поле `data1` регистрирует время в момент выхода автомобиля на работу и в момент получения заказа. Это позволяет сортировать автомобили по времени получения последнего заказа, при получении оператором заказа и выборе, кому отдать заказ, вверху списка оказываются автомобили, имеющие большее время простоя. Листинг 4.21 содержит дамп для создания таблицы `cars`.

Листинг 4.21

```
CREATE TABLE `cars` (
`id` int(9) NOT NULL AUTO_INCREMENT,
`owner` int(9) NOT NULL,
`mark` varchar(30) NOT NULL,
```

```
`number` varchar(30) NOT NULL,
`id_driver` int(9) default '0',
`data1` datetime NOT NULL,
`data2` datetime NOT NULL,
`status` set('yeszakaz', 'yes', 'no', 'out') default NULL,
UNIQUE KEY `id` (`id`),
KEY `data1` (`data1`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

И наконец, основная таблица — `orders`. Содержит информацию о всех заказах в различной степени выполнения. Структура таблицы `orders` следующая:

- `id` — первичный ключ;
- `id_cars` — идентификатор автомобиля;
- `id_driver` — идентификатор водителя;
- `data1` — дата и время получения заказа оператором;
- `data2` — дата и время передачи заказа таксисту;
- `data3` — дата и время получения подачи автомобиля;
- `data4` — дата и время высадки пассажира;
- `route` — маршрут с остановками и точками проезда;
- `route_detail` — полное описание маршрута (навигатор);
- `distance` — протяженность маршрура;
- `summa` — стоимость по теоретичеки просчитанному маршруту;
- `summa_fact` — оплачено фактически;
- `status` — статус заказа;
  - `yes1` — принят оператором;
  - `yes2` — принят водителем;
  - `yes3` — автомобиль подан;
  - `yes4` — автомобиль в пути;
  - `yes5` — заказ выполнен;
  - `no1` — отказ.

Здесь нужны некоторые пояснения. Так как при прокладывании маршрута такси мы будем использовать сервис API Яндекс.Карт, это позволяет нам определять не только точки остановок (начальная, конечная, промежуточная), но и точки, через которые проходит маршрут. Кроме этого, мы сможем получать протяженность маршрута и полное описание движения по маршруту. Листинг 4.22 содержит дамп для создания таблицы `orders`.

Листинг 4.22

```
CREATE TABLE `orders` (
`id` int(9) NOT NULL AUTO_INCREMENT,
`id_cars` int(9) NOT NULL,
`id_driver` int(9) NOT NULL,
`data1` datetime NOT NULL,
`data2` datetime NOT NULL,
`data3` datetime NOT NULL,
`data4` datetime NOT NULL,
`route` text NOT NULL,
`route_detail` text NOT NULL,
`distance` float(10, 3) NOT NULL,
`summa` float(10, 2) NOT NULL,
`summa_fact` float(10, 2) NOT NULL,
`status` set('yes1', 'yes2', 'yes3', 'yes4', 'yes5', 'no1') default NULL,
UNIQUE KEY `id` (`id`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

Я думаю, структура достаточно понятна. На прилагаемом к книге компакт-диске содержится заполненная тестовая база (файл glava_04\4-2\taxi.sql). В Интернете см. **http://examples-api.bazakatalogov.ru/yandex/taxi**. Теперь приступим к программированию сайта.

4.2.2. Программирование сайта

Нам необходимо реализовать следующий функционал:

- ❒ занесение в базу, редактирование, удаление из базы водителей;
- ❒ изменение статуса водителей при выходе на работу, уходе с работы, увольнении;
- ❒ занесение в базу, редактирование, удаление из базы автомобилей;
- ❒ прикрепление водителя к автомобилю при выходе авто на маршрут;
- ❒ формирование заказа, прокладывание маршрута по карте, расчет стоимости заказа;
- ❒ передача заказа водителю и отслеживание выполнения заказа.

Программирование блока *Водители*

При выборе пункта **Водители** верхнего меню открывается блок **Водители** (рис. 4.6).

Программы блока находятся в папке drivers. Блок состоит из формы фильтра для поиска, блока вывода результатов и блока создания, просмотра, редактирования информации для выбранного водителя. Фильтр позволяет осуществлять поиск водителей по фамилии и по статусу (на работе, не на работе, в штате (на работе, не на

работе), уволен). При выходе на работу/уходе с работы оператор должен изменить статус водителя. Это делать обязательно, т. к. "усадить" за руль автомобиля программа позволит только водителя, имеющего статус "на работе". Функция создания контента формы фильтра `f_filter_drivers()` находится в файле drivers\filter_drivers.php, содержимое которого представлено в листинге 4.23.

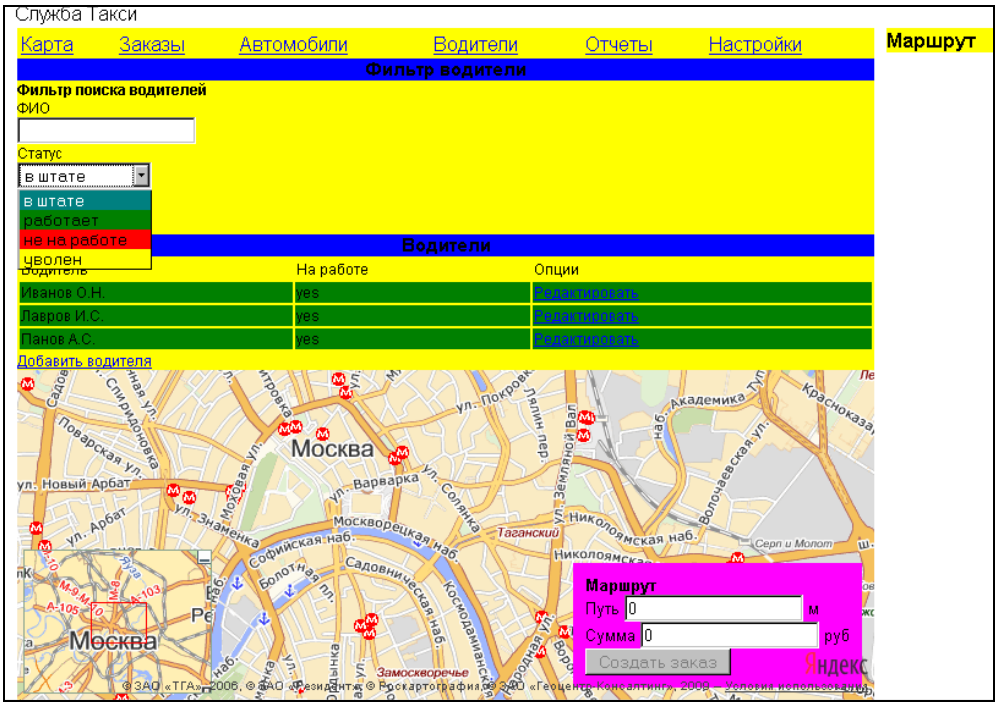


Рис. 4.6. Вид блока **Водители**

Листинг 4.23

```
<?php
// Контент для формы фильтра для выбора водителей
function f_filter_drivers()
{ $content="";
  $content.="<form id='FormFilterDrivers' action='javascript:void();'
      onsubmit='xajax.$(\"ButtonFormFilterDrivers\").disabled=true;

    xajax.$(\"ButtonFormFilterDrivers\").value=\"Подождите...\";
    xajax_View_Drivers(xajax.getFormValues(\"FormFilterDrivers\"));'
    enctype='multipart/form-data';>";
  $content.="<input type=hidden id=pagesearch name=pagesearch value=1>";
  $content.="<b>Фильтр поиска водителей </b>";
  $content.="<br>ФИО <br>
            <input type=text id=fio name=fio value=''>";
```

```
  $content.="<br>Статус <br>
            <select name=status>
            <option value='all' selected>в штате
            <option class='yes' value='yes'>работает
            <option class='no' value='no'>не на работе
            <option class='out' value='out'>уволен
            </select>";
  $content.="<br><input type=submit id='ButtonFormFilterDrivers'
            value='Показать'><br><br>";
  $content.="</form>";
  return $content;
}
?>
```

Постраничный результат выбора водителей по фильтру осуществляет xAjax-функция `View_Drivers()`. Контент формирует функция `f_view_drivers()`. Эти функции расположены в файле drivers\view_drivers.php, содержимое его представлено в листинге 4.24. Результат выводится в блоки с `id=block2` и `id=block21`. Водители с различным статусом выводятся разным цветом.

Листинг 4.24

```
<?php
// Водители по фильтру постранично
function View_Drivers($Id)
{ $objResponse = new xajaxResponse();
  // формирование контента для результата
  $content2=f_view_drivers($Id);
  // выдача контента в блоки
  $objResponse->assign("block2","innerHTML",$content2[0]);
  $objResponse->assign("block21","innerHTML",$content2[1]);
  // активация submit формы поиска
  $objResponse->assign("ButtonFormFilterDrivers","disabled",false);
  $objResponse->assign("ButtonFormFilterDrivers","value","Показать");
  return $objResponse;
}
// формирование контента
function f_view_drivers($Id)
{
  require_once('mybaza.php');
  require_once('my.php');
  $content=array();$content0.="";
  // формирование запроса
  $query0="SELECT COUNT(id) FROM ".DRIVERS." WHERE status<>'firma' ";
  if($Id[status]=='all')
    $query0.="&& (status='no' || status='yes') ";
```

```
  elseif($Id[status]=='yes')
    $query0.="&& status='yes' ";
  elseif($Id[status]=='no')
    $query0.="&& status='no' ";
  elseif($Id[status]=='out')
    $query0.="&& status='out' ";
  else
    ;
  if(strlen($Id[fio])>0)
    $query0.="&& fio LIKE '%".$Id[fio]."%' ";
  $rez0=mysql_query($query0);
  $count=mysql_result($rez0,0);
  $pages=ceil($count/NN1);
  $page=min($Id[pagesearch],$pages);$poz=($page-1)*NN1;
  if($count>0)
  { $content0.="<table width=100%><tr>";
    $content0.="<td>Водитель</td>";
    $content0.="<td>На работе</td>";
    $content0.="<td>Опции</td></tr>";
    $query0.=" LIMIT ".$poz.", ".NN1."";
    $query1=str_replace("COUNT(id)","*",$query0);
    $rez1=mysql_query($query1);
    while($row1=mysql_fetch_assoc($rez1))
    { $content0.="<tr>";
      $content0.="<td class='".$row1[status]."'>".$row1[fio]."</td>";
      $content0.="<td class='".$row1[status]."'>".$row1[status]."</td>";
      $content0.="<td class='".$row1[status]."'>
      <a href='javascript:void()' onclick='xajax_Edit_Driver(".
               $row1[id].");' >Редактировать</a></td>";
      $content0.="</tr>";
    }
    $content0.="</table>";
    // указатель страниц
  ... ... ...
  // возврат значений
  $content[0]=$content0;
  $content[1]=$content1;
  return $content;
}
?>
```

Информацию о водителе (включая статус) мы можем изменять (xAjax-функции `Edit_Driver()` и `Add_Edit_Driver()`), расположенные в файлах drivers\edit_driver.php и drivers\add_edit_driver.php соответственно. Содержимое этих файлов представлено в листинге 4.25.

Листинг 4.25

```
<?php
// Контент для формы редактирования водителя
function Edit_Driver($id)
{ $objResponse = new xajaxResponse();
  require_once('mybaza.php');
  $query1="SELECT * FROM ".DRIVERS." WHERE id='".$id."' ";
  $rez1=mysql_query($query1);
  $row1=mysql_fetch_assoc($rez1);
  $content.="<form id='FormEditDriver' action='javascript:void();'
  onsubmit='xajax.$(\"ButtonFormEditDriver\").disabled=true;
  xajax.$(\"ButtonFormEditDriver\").value=\"Подождите...\";
  xajax_Add_Edit_Driver(xajax.getFormValues(\"FormEditDriver\"));'
  enctype='multipart/form-data';>";
  $content.="<input type=hidden name=id value='".$id."'>";
  $content.="<table><tr><td>Фамилия,имя,отчество</td>";
  $content.="<td><input type=text name=fio size=50
  value='".$row1[fio]."'></td></tr>";
  ... ... ...
  if($row1[status]=='yes' || $row1[status]=='no')
    $selyes="selected";$selno="";
    $content.="<tr><td>Статус </td><td><select name=status>
              <option class='yes' value='yes'
              ".(($row1[status]==yes)?($selyes):($selno)).">
              работает<option class='no' value='no'
              ".(($row1[status]==no)?($selyes):($selno)).">
              не на работе<option class='out' value='out'
              ".(($row1[status]==out)?($selyes):($selno)).">
              уволен</select></td></tr></table>";
  $content.="<br><input type=submit id='ButtonFormEditDriver'
              value='Изменить'><br>";
  ... ... ...
  $content.="</form>";
  ... ... ...
  return $objResponse;
}
// Изменение в базе данных пользователя водителя
function Add_Edit_Driver($Id)
{ $objResponse = new xajaxResponse();
  require_once('mybaza.php');
  // sql запрос изменения информации
  $query1="UPDATE ".DRIVERS." SET
          fio='".clear($Id[fio])."',address='".clear($Id[address])."',
          phone='".clear($Id[phone])."', passport='".clear($Id[passport])."',
          prava='".clear($Id[prava])."', status='".$Id[status]."'
          WHERE id='".$Id[id]."' ";
```

```
  $rez1=mysql_query($query1);
  if(!$rez1)
  { $objResponse->alert("Ошибка редактирования");
    $objResponse->assign("ButtonFormEditDriver","disabled",false);
    $objResponse->assign("ButtonFormEditDriver","value","Создать");
    return $objResponse;
  }
  else
  { $objResponse->assign("zag3","innerHTML","<table></table>");
    $objResponse->assign("block3","innerHTML","<table></table>");
    $objResponse->alert("Данные водителя изменены");
  }
  // для обновления блока результата – эмуляция нажатия кнопки поиска
  $script="document.getElementById('ButtonFormFilterDrivers').click();";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

При щелчке по ссылке **Добавить водителя** вызывается форма создания записи нового водителя (xAjax-функция `New_Driver()`). Вид формы представлен на рис. 4.7.

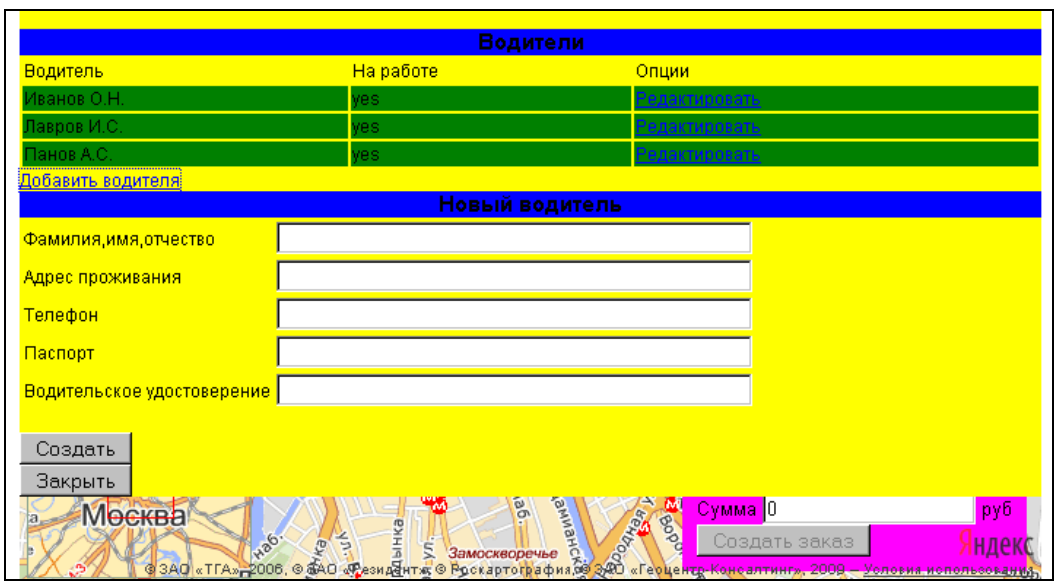


Рис. 4.7. Форма создания записи водителя такси

xAjax-функция `New_Driver()` находится в файле drivers\new_driver.php, содержимое которого представлено в листинге 4.26.

Листинг 4.26

```
<?php
// Контент для формы создания нового водителя
function New_Driver()
{ $objResponse = new xajaxResponse();
  $content="";
  $content.="<form id='FormNewDriver' action='javascript:void();'
    onsubmit='xajax.$(\"ButtonFormNewDriver\").disabled=true;
  xajax.$(\"ButtonFormNewDriver\").value=\"Подождите...\";
  xajax_Add_New_Driver(xajax.getFormValues(\"FormNewDriver\"));'
  enctype='multipart/form-data';>";
  $content.="<table>";
  $content.="<tr><td>Фамилия,имя,отчество</td>";
  $content.="<td><input type=text name=fio size=50></td></tr>";
  $content.="<tr><td>Адрес проживания</td>";
  $content.="<td><input type=text name=address size=50></td></tr>";
  $content.="<tr><td>Телефон</td>";
  $content.="<td><input type=text name=phone size=50></td></tr>";
  $content.="<tr><td>Паспорт</td>";
  $content.="<td><input type=text name=passport size=50></td></tr>";
  $content.="<tr><td>Водительское удостоверение</td>";
  $content.="<td><input type=text name=prava size=50></td></tr>";
  $content.="</table>";
  // кнопки Создать и Отменить
  $content.="<br><input type=submit id='ButtonFormNewDriver'
             value='Создать'><br>";
  $content.="<input type=button onclick='
  document.getElementById(\"zag3\").innerHTML=\"<table></table>\";
  document.getElementById(\"block3\").innerHTML=\"<table></table>\";'
  value='Закрыть'><br>";
  $content.="</form>";
  // заголовок
  $objResponse->assign("zag3","innerHTML","Новый водитель");
  $objResponse->assign("block3","innerHTML",$content);
  // блок в зону видимости
  $objResponse->script("document.getElementById('block3').scrollIntoView();");
  return $objResponse;
}
?>
```

При нажатии кнопки **Создать** вызывается xAjax-функция `Add_New_Driver()`, которая создает в базе данных запись со сведениями о новом водителе. Функция `Add_New_Driver` находится в файле drivers\add_new_driver.php, содержимое которого представлено в листинге 4.27.

Листинг 4.27

```
<?php
// Добавление в базу нового водителя
function Add_New_Driver($Id)
{ $objResponse = new xajaxResponse();
  require_once('mybaza.php');
  // добавление записи в БД
  $query1="INSERT INTO ".DRIVERS." SET
          fio='".clear($Id[fio])."',address='".clear($Id[address])."',
          phone='".clear($Id[phone])."',
          passport='".clear($Id[passport])."',
          prava='".clear($Id[prava])."',data1='".date('Y-m-d H:i:s')."',
          status='no' ";
  $rez1=mysql_query($query1);
  if(!$rez1)
  { $objResponse->alert("Ошибка добавления");
    $objResponse->assign("ButtonFormNewDriver","disabled",false);
    $objResponse->assign("ButtonFormNewDriver","value","Создать");
    return $objResponse;
  }
  else
  { $objResponse->assign("zag3","innerHTML","<table></table>");
    $objResponse->assign("block3","innerHTML","<table></table>");
    $objResponse->alert("Новый водитель добавлен");
  }
  // для обновления блока результата — эмуляция нажатия кнопки поиска
  $script="document.getElementById('ButtonFormFilterDrivers').click();";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

После изменения статуса водителей оператор может переходить к предоставлению автомобилей вышедшим на работу водителям.

Программирование блока *Автомобили*

При выборе пункта **Автомобили** верхнего меню открывается блок **Автомобили** (рис. 4.8).

Программы блока находятся в папке cars. Блок состоит из формы фильтра для поиска, блока вывода результатов и блока создания, просмотра, редактирования информации для выбранного автомобиля. Фильтр позволяет осуществлять поиск автомобилей по марке, владельцу и по статусу (**работает заказ, работает свободен, без водителя, в ремонте**). Кроме этого, в фильтре есть пункт **исправные**, который отбирает для поиска все авто, кроме статуса "в ремонте". Для вывода автомобиля на маршрут необходимо "усадить" в него водителя. Выбор водителей осуществля-

ется при редактировании данных автомобиля. При этом можно выбрать только "свободного" водителя, находящегося на работе. Программа должна автоматически формировать этот список. По окончании рабочего дня при переводе автомобиля в статус "без водителя" водитель остается свободным. Функция создания контента формы фильтра `f_filter_cars()` находится в файле cars\filter_cars.php, содержимое которого представлено в листинге 4.28.

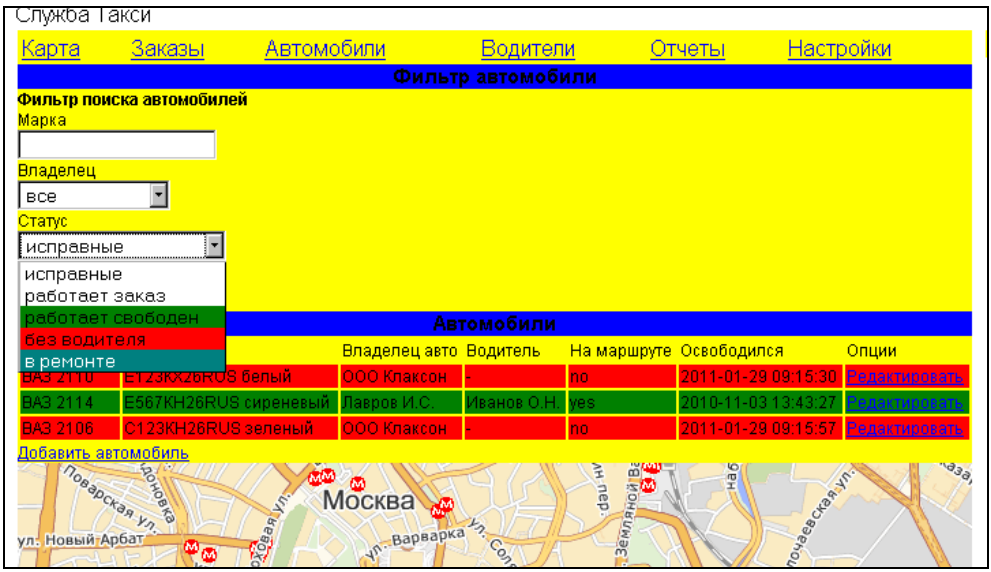


Рис. 4.8. Вид блока **Автомобили**

Листинг 4.28

```
<?php
// Контент для формы фильтра для выбора автомобилей
function f_filter_cars()
{ require_once('mybaza.php');
  $content="";
  $content.="<form id='FormFilterCars' action='javascript:void();'
     onsubmit='xajax.$(\"ButtonFormFilterCars\").disabled=true;
     xajax.$(\"ButtonFormFilterCars\").value=\"Подождите...\";
     xajax_View_Cars(xajax.getFormValues(\"FormFilterCars\"));'
     enctype='multipart/form-data';>";
  $content.="<input type=hidden id=pagesearch name=pagesearch value=1>";
  $content.="<b>Фильтр поиска автомобилей </b>";
  $content.="<br>Марка <br>
             <input type=text id=mark name=mark value=''>";
  $content.="<br>Владелец <br><select name=owner>
             <option value=0 selected>все";
  $query11="SELECT DISTINCT(owner) FROM ".CARS." ";
  $rez11=mysql_query($query11);
```

```
  while($row11=mysql_fetch_assoc($rez11))
  { $query12="SELECT fio FROM ".DRIVERS." WHERE id='".$row11[owner]."' ";
    $rez12=mysql_query($query12);
    $fio=mysql_result($rez12,0);
    $content.="<option value='".$row11[owner]."'>".$fio;
  }
  $content.="</select>";
  $content.="<br>Статус <br>
             <select name=status>
             <option value='all' selected>исправные
             <option class='yeszakaz' value='yeszakaz'>работает заказ
             <option class='yes' value='yes'>работает свободен
             <option class='no' value='no'>без водителя
             <option class='out' value='out'>в ремонте
             </select>";
  $content.="<br><input type=submit id='ButtonFormFilterCars'
             value='Показать'><br><br>";
  $content.="</form>";
  return $content;
}
?>
```

Постраничный результат выбора водителей по фильтру осуществляет xAjax-функция `View_Cars()`. Контент формирует функция `f_view_cars()`. Эти функции расположены в файле cars\view_cars.php, содержимое его представлено в листинге 4.29. Результат выводится в блоки с `id=block2` и `id=block21`. Автомобили с различным статусом выводятся разным цветом.

Листинг 4.29

```
<?php
// Водители по фильтру постранично
function View_Cars($Id)
{ $objResponse = new xajaxResponse();
  // получить контент
  $content2=f_view_cars($Id);
  // вывести результат в блоки block2, block21
  $objResponse->assign("block2","innerHTML",$content2[0]);
  $objResponse->assign("block21","innerHTML",$content2[1]);
  // восстановить активность кнопки формы поиска
  $objResponse->assign("ButtonFormFilterCars","disabled",false);
  $objResponse->assign("ButtonFormFilterCars","value","Показать");
  return $objResponse;
}
// Формирование контента
function f_view_cars($Id)
```

```
{ require_once('mybaza.php'); require_once('my.php');
  $content=array();$content0.="";
  // формирование запроса
  $query0="SELECT COUNT(id) FROM ".CARS." WHERE id>0 ";
  if($Id[status]=='all')
    $query0.="&& (status='no' || status='yes' || status='yeszakaz') ";
  elseif($Id[status]=='yeszakaz')
    $query0.="&& status='yeszakaz' ";
  elseif($Id[status]=='yes')
    $query0.="&& status='yes' ";
  elseif($Id[status]=='no')
    $query0.="&& status='no' ";
  elseif($Id[status]=='out')
    $query0.="&& status='out' ";
  else ;
  if(strlen($Id[mark])>0) $query0.="&& mark LIKE '%".$Id[mark]."%' ";
  if($Id[owner]>0)        $query0.="&& owner='".$Id[owner]."' ";
  // выполнить запрос
  $rez0=mysql_query($query0);
  $count=mysql_result($rez0,0);
  $pages=ceil($count/NN2);
  $page=min($Id[pagesearch],$pages);$poz=($page-1)*NN2;
  if($count>0)  // результат непустой
  { $content0.="<table width=100%><tr>";
    $content0.="<td>Автомобиль</td><td>Номер</td>";
    $content0.="<td>Владелец авто</td><td>Водитель</td>";
    $content0.="<td>На маршруте</td>Освободился</td>";
    $content0.="<td>Опции</td></tr>";
    $query0.=" ORDER BY data1 ASC LIMIT ".$poz.", ".NN2."";
    $query1=str_replace("COUNT(id)","*",$query0);
    $rez1=mysql_query($query1);
    // формирование данных таблицы
    while($row1=mysql_fetch_assoc($rez1))
    { $content0.="<tr>";
      $content0.="<td class='".$row1[status]."'>".$row1[mark]."</td>";
      $content0.="<td class='".$row1[status]."'>".$row1[number]."</td>";
      $query2="SELECT fio FROM drivers WHERE id='".$row1[owner]."' ";
      ... ... ...
      $content0.="<td class='".$row1[status]."'><a href='javascript:void()'
                  onclick='xajax_Edit_Car(".$row1[id].");'>
                  Редактировать</a></td>";
      $content0.="</tr>";
    }
    $content0.="</table>";
    // указатель страниц
   ... ... ...
```

```
   // ссылка на добавление автомобиля
  $content0.="<a href='javascript:void()' onclick='xajax_New_Car();'>
              Добавить автомобиль</a>";
  // возврат значений
  $content[0]=$content0;
  $content[1]=$content1;
  return $content;
}
?>
```

Информацию об автомобиле (включая статус) мы можем изменять — xAjax-функции `Edit_Car()` и `Add_Edit_Car()`, расположенные в файлах cars\edit_car.php и drivers/add_edit_car.php соответственно. Содержимое этих файлов представлено в листинге 4.30. При этом при изменении статуса автомобиля на "работает свободен" в списке водителей небходимо выбрать водителя, этот список программно формируется из свободных водителей, находящихся на работе (см. рис. 4.9). При изменении статуса автомобиля на "без водителя" водитель становится автоматически свободным.

Листинг 4.30

```
<?php
// Контент для формы редактирования автомобиля
function Edit_Car($id)
{ $objResponse = new xajaxResponse();
  require_once('mybaza.php');
  $query0="SELECT * FROM ".CARS." WHERE id='".$id."' ";
  $rez0=mysql_query($query0);
  $row0=mysql_fetch_assoc($rez0);
  $content.="<form id='FormEditCar' action='javascript:void();'
      onsubmit='xajax.$(\"ButtonFormEditCar\").disabled=true;
      xajax.$(\"ButtonFormEditCar\").value=\"Подождите...\";
      xajax_Add_Edit_Car(xajax.getFormValues(\"FormEditCar\"));'
      enctype='multipart/form-data';>";
  $content.="<input type=hidden name=id value='".$id."'>";
  $content.="<table>";
  $content.="<tr><td>Марка автомобиля</td>";
  $content.="<td><input type=text name=mark size=50
             value='".$row0[mark]."'></td></tr>";
  $content.="<tr><td>Номер, цвет</td>";
  ... ... ...
  $content.="<br><input type=submit id='ButtonFormEditCar'
             value='Изменить'><br>";
  ... ... ...
  $content.="</form>";
  // заголовок
  $objResponse->assign("zag3","innerHTML","Автомобиль".$row0[mark]);
```

```
  // вывод контента
  $objResponse->assign("block3","innerHTML",$content);
  // перевести видимость на блок
  $objResponse->script("document.getElementById('block3').scrollIntoView();");
  return $objResponse;
}
// Изменение в базе данных автомобиля
function Add_Edit_Car($Id)
{ $objResponse = new xajaxResponse();
  require_once('mybaza.php');
  if($Id[status]=='no') $id_driver=0;
  else                  $id_driver=$Id[driver];
  $query1="UPDATE ".CARS." SET
          mark='".clear($Id[mark])."',number='".clear($Id[number])."',
          owner='".$Id[owner]."',status='".$Id[status]."',
          id_driver='".$id_driver."' ";
  if($Id[status]=='yes' || $Id[status]=='yeszakaz')
    $query1.=",data1='".date('Y-m-d H:i:s')."' ";
  if($Id[status]=='no')
    $query1.=",data2='".date('Y-m-d H:i:s')."' ";
  $query1.="WHERE id='".$Id[id]."' ";
  $rez1=mysql_query($query1);
  ... ... ...
  // для обновления
  $script="document.getElementById('ButtonFormFilterCars').click();";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

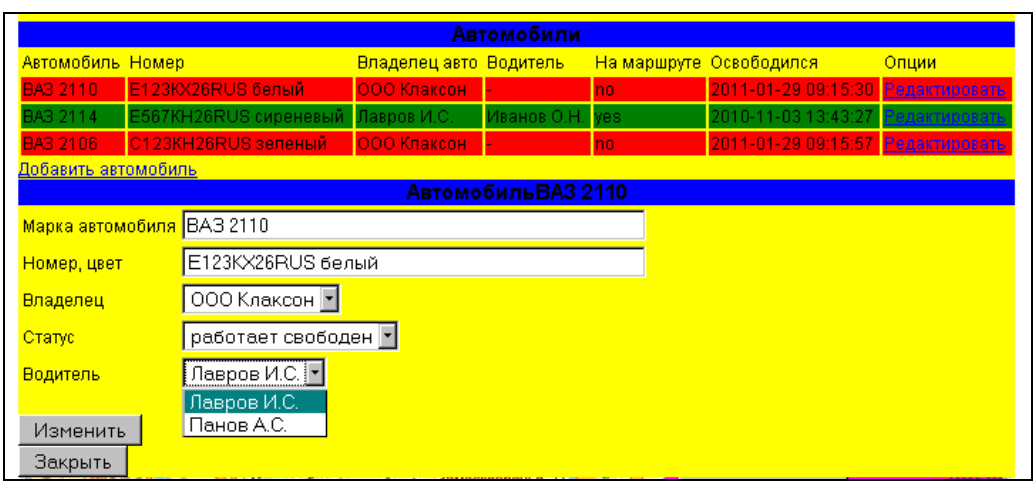


Рис. 4.9. Привязка водителя к автомобилю

При щелчке по ссылке **Добавить автомобиль** вызывается форма создания записи нового автомобиля (xAjax-функция `New_Car()`). Вид формы представлен на рис. 4.10.

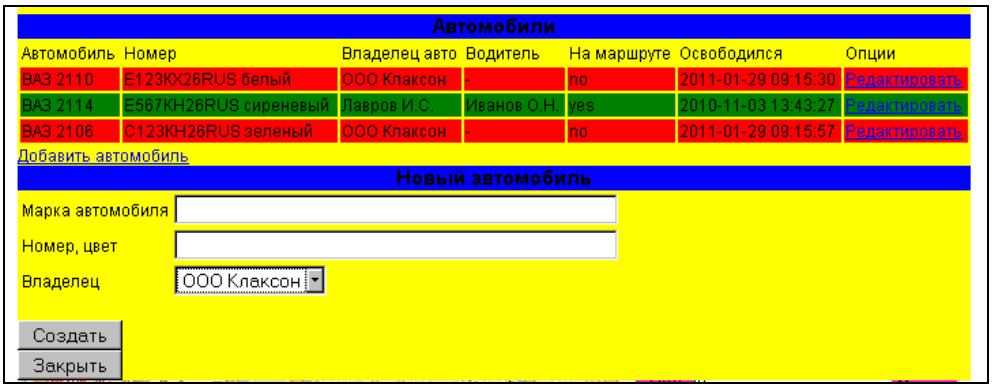


Рис. 4.10. Форма добавления нового автомобиля

xAjax-функция `New_Car()` находится в файле cars\new_car.php, содержимое которого представлено в листинге 4.31.

Листинг 4.31

```
<?php
// Контент для формы создания
// нового автомобиля
function New_Car()
{ $objResponse = new xajaxResponse();
  require_once('mybaza.php');
  $content="";
  $content.="<form id='FormNewCar' action='javascript:void();' onsubmit='
          xajax.$(\"ButtonFormNewCar\").disabled=true;
          xajax.$(\"ButtonFormNewCar\").value=\"Подождите...\";
          xajax_Add_New_Car(xajax.getFormValues(\"FormNewCar\"));'
          enctype='multipart/form-data';>";
  $content.="<table>";
  $content.="<tr><td>Марка автомобиля</td>";
  $content.="<td><input type=text name=mark size=50></td></tr>";
  $content.="<tr><td>Номер, цвет</td>";
  $content.="<td><input type=text name=number size=50></td></tr>";
  $content.="<tr><td>Владелец</td>";
  $content.="<td><select name=owner>";
  $query1="SELECT id,fio FROM ".DRIVERS." WHERE
            status='yes' || status='firma' ";
  $rez1=mysql_query($query1);
  while($row1=mysql_fetch_assoc($rez1))
  { $content.="<option value=".$row1[id].">".$row1[fio]; }
```

```
  $content.="</select></td></tr>";
  $content.="</table>";
  $content.="<br><input type=submit id='ButtonFormNewCar'
             value='Создать'><br>";
  $content.="<input type=button onclick='
             document.getElementById(\"zag3\").innerHTML=\"
             <table></table>\";
             document.getElementById(\"block3\").innerHTML=\"
             <table></table>\";'
             value='Закрыть'><br>";
  $content.="</form>";
  $objResponse->assign("zag3","innerHTML","Новый автомобиль");
  $objResponse->assign("block3","innerHTML",$content);
  $objResponse->script("document.getElementById('block3').
                 scrollIntoView();");
  return $objResponse;
}
?>
```

При нажатии кнопки **Создать** вызывается xAjax-функция `Add_New_Car()`, которая создает в базе данных запись со сведениями о новом автомобиле. Эта функция находится в файле cars\add_new_car.php, содержимое которого представлено в листинге 4.32.

Листинг 4.32

```
<?php
// Добавление в базу нового автомобиля
function Add_New_Car($Id)
{ $objResponse = new xajaxResponse();
  require_once('mybaza.php');
  $query1="INSERT INTO ".CARS." SET
          mark='".clear($Id[mark])."',number='".clear($Id[number])."',
          owner='".$Id[owner]."',status='no' ";
  $rez1=mysql_query($query1);
  if(!$rez1)
  { $objResponse->alert("Ошибка добавления");
    $objResponse->assign("ButtonFormNewCar","disabled",false);
    $objResponse->assign("ButtonFormNewCar","value","Создать");
    return $objResponse;
  }
  else
  { $objResponse->assign("zag3","innerHTML","<table></table>");
    $objResponse->assign("block3","innerHTML","<table></table>");
    $objResponse->alert("Новый автомобиль добавлен");
  }
```

```
  // для обновления – эмуляция нажатия на кнопку поиска
  $script="document.getElementById('ButtonFormFilterCars').click();";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

Теперь можно переходить к основным функциям сайта — заказам и их выполнению.

Получение заказа и создание маршрута с API Яндекс.Карт

При щелчке по ссылке **Карта** верхнего меню карта открывается полностью (рис. 4.11). При получении заказа оператор должен указать точки маршрута (начальную, конечную, промежуточные, транзитные) и запустить сервис "Маршрутизатор" API Яндекс.Карт. Из проложенного маршрута мы должны извлечь протяженность маршрута, подробное его описание, рассчитать стоимость проезда и создать заказ с начальным статусом "принят". Для управления логикой создадим на карте несколько пользовательских кнопок и пользовательский элемент управления, где будут высвечиваться: протяженность маршрута, стоимость проезда и кнопка **Создать заказ**.

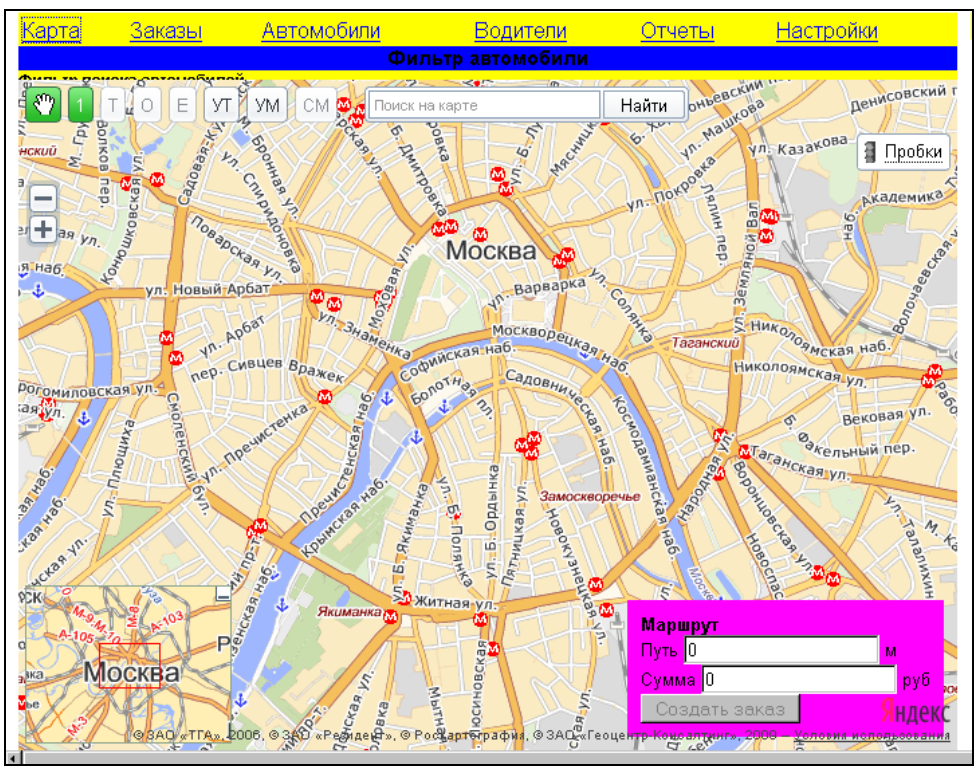


Рис. 4.11. Полный обзор карты

Размещение внешних элементов управления

Для удобства навигации по карте разместим следующие внешние элементы управления:

- компактный элемент масштабирования;
- обзорная карта;
- поиск по карте.

Код для размещения представлен в листинге 4.33.

Листинг 4.33

```
// поиск
searchControl=new YMaps.SearchControl(
       {noPlacemark:true,resultsPerPage: 3,useMapBounds: 1 });
map.addControl(searchControl);
// мини-карта
miniMapPositive=new YMaps.MiniMap(4);
map.addControl(miniMapPositive);
// масштабирование
smallZoomControl=new YMaps.SmallZoom({smooth:false});
map.addControl(smallZoomControl);
// отключить масштабирование двойным кликом
map.disableDblClickZoom();
```

Для получения маршрута с помощью сервиса "Маршрутизатор" необходимо задать точки (минимум 2 — начальную и конечную). Для логического управления процессом выбора кнопок и формированием маршрута создадим панель инструментов с кнопками. Для задания и удаления кнопок создадим переключатель на 5 кнопок:

- **1** — начальная точка маршрута;
- **Е** — конечная точка маршрута;
- **О** — остановка промежуточная;
- **Т** — транзитная точка маршрута (остановка не предполагается, но через эту точку нужно проложить маршрут);
- **УТ** — удалить точки.

Кроме этого, нам нужны две обычные кнопки — для запуска процесса построения маршрута и для очистки маршрута:

- **УМ** — удалить маршрут;
- **СМ** — создать маршрут.

В листинге 4.34 приведен код создания и размещения переключателя и кнопок запуска процесса построения маршрута.

Листинг 4.34

```
toolbar1 = new YMaps.ToolBar([new YMaps.ToolBar.MoveButton()]);
// Переключатель из кнопок
//   (начальная, конечная, промежуточные, транзитные точки)
button11 = new YMaps.ToolBarRadioButton('group1', {
    caption: "1", hint: "Начальная"},{selected:true});
button12 = new YMaps.ToolBarRadioButton('group1', {
    caption: "T", hint: "Транзитная"});
button13 = new YMaps.ToolBarRadioButton('group1', {
    caption: "O", hint: "Промежуточная остановка"});
button14 = new YMaps.ToolBarRadioButton('group1', {
    caption: "E", hint: "Конечная"});
button2 = new YMaps.ToolBarRadioButton('group1', {
    caption: "УТ", hint: "Удалить точки"});
// Начало: нужно выбрать первую точку, остальные варианты блокировать
button12.disable();button13.disable();
button14.disable();button2.enable();
// кнопки Сброс, Очистить маршрут, Создать маршрут
button3 = new YMaps.ToolBarButton({
    caption: "УМ",hint: "Очистить маршрут"});
button4 = new YMaps.ToolBarButton({
    caption: "СМ",hint: "Создать маршрут"});
button4.disable();
// добавить на панель инструментов
//toolbar1.add(new YMaps.ToolBarSeparator(10));
toolbar1.add(button11);
toolbar1.add(button12);
toolbar1.add(button13);
toolbar1.add(button14);
toolbar1.add(button2);
toolbar1.add(button3);
toolbar1.add(button4);
// добавить панель инструментов на карту
map.addControl(toolbar1);
```

Создание и размещение пользовательского элемента управления

Создадим пользовательский элемент управления — табло, на котором после построения маршрута будет выводиться информация о протяженности маршрута и стоимости поездки. Поместим на табло и кнопку **Создать заказ**, которая будет активироваться после построения маршрута. Табло разместим в правом нижнем углу карты. Как мы помним из *главы 3*, для пользовательского элемента необходимо определить два метода — `onAddToMap()` и `onRemoveFromMap()`. Код создания и размещения на карте пользовательского элемента управления "табло" приведен в листинге 4.35.

Листинг 4.35

```
// создание пользовательского элемента
map.addControl(new Res_Route());
//
function Res_Route()
{ // Добавление на карту
  this.onAddToMap = function (map, position)
  { var cont_container="<div><b>Маршрут</b><br>Путь <input id=distance
    readonly value=0> м<br>Сумма <input id=summa readonly value=0>
    руб<div id=info></div><div id=button1><input type=button id=button_ok
    onclick='create_zakaz();' value='Создать заказ' disabled></div><div>"
    this.container = YMaps.jQuery(cont_container);
    this.map = map;
    this.position = new
      YMaps.ControlPosition(YMaps.ControlPosition.BOTTOM_RIGHT,
      new YMaps.Size(10, 10));
    this.container.css({position: "absolute",
                        zIndex: YMaps.ZIndex.CONTROL,
                        background: '#f0f', listStyle: 'none',
                        padding: '10px', margin: 0});
    // Применение позиции к управляющему элементу
    this.position.apply(this.container);
    // Добавление на карту
    this.container.appendTo(this.map.getContainer());
  }
  // Удаление с карты
  this.onRemoveFromMap = function ()
  { this.container.remove();
    this.container = this.map = null;
  };
}
```

Задание точек маршрута

Задавать точки маршрута мы будем добавлением метки на карту. Добавление точки будет происходить по событию щелчка мышью по карте. Для каждого типа точек маршрута определим свой стиль метки. Задание стиля меток представлено в листинге 4.36.

Листинг 4.36

```
//** СТИЛИ ДЛЯ МЕТОК
// начальная точка
var s0 = new YMaps.Style();
s0.iconStyle = new YMaps.IconStyle();
s0.iconStyle.offset = new YMaps.Point(-15, -15);
```

```
s0.iconStyle.href = "http://google-maps-icons.googlecode.com/files/car.png";
s0.iconStyle.size = new YMaps.Point(32, 37);
YMaps.Styles.add("user#start", s0);
// конечная точка
var s1 = new YMaps.Style();
s1.iconStyle = new YMaps.IconStyle();
s1.iconStyle.offset = new YMaps.Point(-15, -15);
s1.iconStyle.href="http://google-maps-icons.googlecode.com/files/parking.png";
s1.iconStyle.size = new YMaps.Point(32, 37);
YMaps.Styles.add("user#end", s1);
// транзитная точка
var s2 = new YMaps.Style();
s2.iconStyle = new YMaps.IconStyle();
s2.iconStyle.offset = new YMaps.Point(-15, -15);
s2.iconStyle.href ="http://google-maps-icons.googlecode.com/files/right.png";
s2.iconStyle.size = new YMaps.Point(32, 37);
YMaps.Styles.add("user#tranzit", s2);
// точка-остановка
var s3 = new YMaps.Style();
s3.iconStyle = new YMaps.IconStyle();
s3.iconStyle.offset = new YMaps.Point(-15, -15);
s3.iconStyle.href ="http://google-maps-icons.googlecode.com/files/stop.png";
s3.iconStyle.size = new YMaps.Point(32, 37);
YMaps.Styles.add("user#stop", s3);
```

Выбор добавляемой метки на карту происходит согласно положению переключателя кнопок **1**, **Т**, **О**, **Е**. По событию `map.Events.Click` происходит добавление точки, соответствующей активному положению переключателя. Логика добавления точек должна предусмотреть возможность ограничения выбора текущей точки, т. е. сначала однозначно выбирается первая точка, затем кнопка **1** блокируется, возможен выбор только конечной точки, остановки, промежуточной и транзитной точки. После выбора конечной точки блокируются все кнопки, кроме кнопки **УТ**, которая становится активной. После этого добавление точек на карту становится невозможным. Становятся доступными для нажатия кнопки **УТ** (удаление всех точек) и **СМ** (создать маршрут). Координаты точки переведем в адрес (сервис "Геокодирование"). Справа от карты находится табло формируемого маршрута, куда мы будем заносить список точек маршрута (см. рис. 4.12). Одновременно для сервиса "Маршрутизатор" формируется массив точек `arrrouter` и массив транзитных точек `arrrouter1`. Код обработки события представлен в листинге 4.37.

Листинг 4.37

```
// для меток создаем группу
group1=new YMaps.GeoObjectCollection();
map.addOverlay(group1);
// при щелчке по карте — добавить метку
YMaps.Events.observe(map, map.Events.Click, function (map, mEvent)
```

```
{ // При активной Уд. Точки (уже выбрана конечная)
  if(button2.isSelected())
  { return; }
  // Создает и добавляет метку на карту
  if(button12.isSelected()) // транзитная
  { var x=metka.length;
    metka[x]=new YMaps.Placemark(mEvent.getGeoPoint(),
               {hideIcon:true,style:"user#tranzit",});
    group1.add(metka[x]);arrrouter1.push(x); img="right.png";dop="|через ";
  }
  if(button13.isSelected())  // промежуточная
  { var x=metka.length;
    metka[x]=new YMaps.Placemark(mEvent.getGeoPoint(),
               {hideIcon:true,style:"user#stop",});
    group1.add(metka[x]); img="car.png";dop="|ост. ";
  }
  if(button14.isSelected())  // конечная
  { var x=metka.length;
    metka[x]=new YMaps.Placemark(mEvent.getGeoPoint(),
               {hideIcon:true,style:"user#end",});
    group1.add(metka[x]);
    button14.deselect();button14.disable();
    button12.disable();button13.disable();
    button2.select();button4.enable();
    img="car.png";dop="|ост. ";
  }
  if(button11.isEnabled())  // начальная
  { var x=0;
    metka[x]=new YMaps.Placemark(mEvent.getGeoPoint(),
               {hideIcon:true,style:"user#start"});
    group1.add(metka[x]);
    button11.deselect();button11.disable();
    button12.enable();button13.enable();
    button14.enable();button14.select();
    img="car.png";dop="ост. ";
  }
  // преобразование координат в адрес
  geocoder1 = new YMaps.Geocoder(mEvent.getGeoPoint());
  // результат геокодирования — координаты -> адрес
  YMaps.Events.observe(geocoder1, geocoder1.Events.Load, function ()
  { if (this.length())
    { var txt=this.get(0).text;
      // в массив arrouter
      arrrouter.push(txt);
      // добавить запись точки в табло маршрута
      create_record(txt);
    }
```

```
    else {alert("не найдено");}
  });
  // ошибка геокодирования
  YMaps.Events.observe(geocoder1, geocoder1.Events.Fault, function
     (geocoder1,errorMessage)
     {alert("Произошла ошибка: " + errorMessage);});
  });
// функция добавления информации о точках маршрута в табло маршрута
function create_record(address)
{ var index=metka.length-1;
  var html="<div id=point"+index+" class='r1'><img src='"+img+"'>
           "+dop+address+"</div>";
  var old_html=document.getElementById("points").innerHTML;
  document.getElementById("points").innerHTML=old_html+html;
}
```

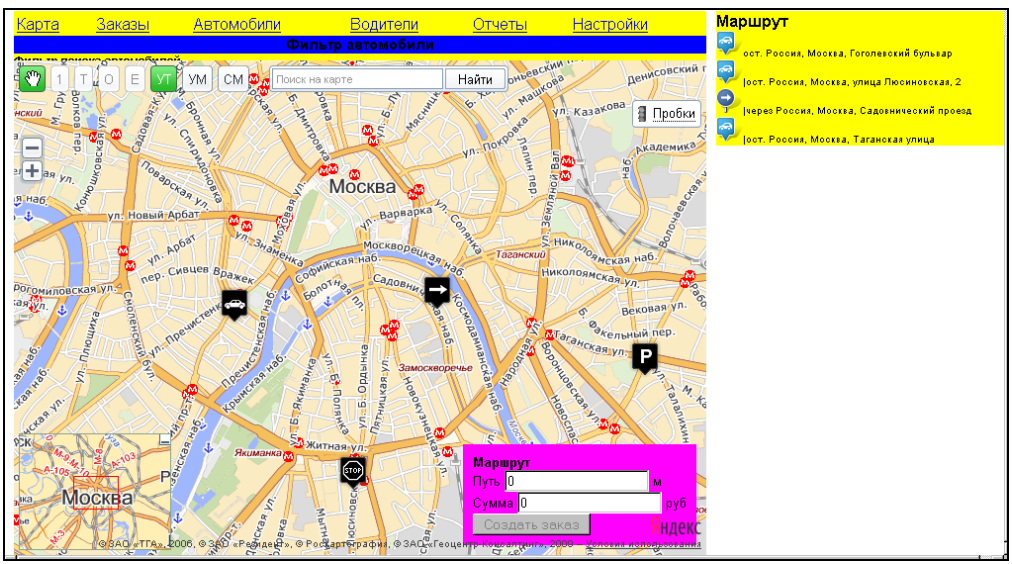


Рис. 4.12. Представление точек формируемого маршрута

Если по какой-то причине мы хотим удалить точки, необходимо нажать кнопку **УТ**. При этом удаляются все точки маршрута, и можно формировать новый маршрут сначала. Обработка события нажатия кнопки **УТ** представлена в листинге 4.38.

Листинг 4.38

```
// Сброс по нажатию кнопки УТ (удалить точки)
YMaps.Events.observe(button2, button2.Events.Click, function ()
{ button11.enable();button11.select();
  button12.disable();button13.disable(); button14.disable();
  group1.removeAll();button4.disable();
  var x=metka.length;metka.splice(0,x);  arrrouter.splice(0,x);
```

```
  var y=arrrouter1.length;arrrouter1.splice(0,y);
  map.removeOverlay(router1);
  document.getElementById("route").innerHTML="";
  document.getElementById("points").innerHTML="";
  document.getElementById("button_ok").disabled=true;
});
```

Прокладывание маршрута через массив точек

Запуск сервиса "Маршрутизация" происходит по событию нажатия кнопки **СМ** (`button4.Events.Click`). Для прокладывания маршрута используется класс `YMaps.Router`, которому передаются массивы `arrrouter` и `arrrouter1` — точки остановки и транзитные точки маршрута и следующие опции:

- `viewAutoApply=1` — установить центр и коэффициент масштабирования карты так, чтобы построенный маршрут был виден целиком;
- `avoidTrafficJams=1` — строить маршрут с учетом пробок.

После построения маршрута добавляем его на карту. Кроме этого, сервис позволяет построить простейший маршрутный лист, который мы выводим в правый блок (рис. 4.13). Подсчитываем протяженность маршрута и стоимость поездки в зависимости от протяженности и количества остановок. Становится активной кнопка **Создать заказ**. Код приведен в листинге 4.39.

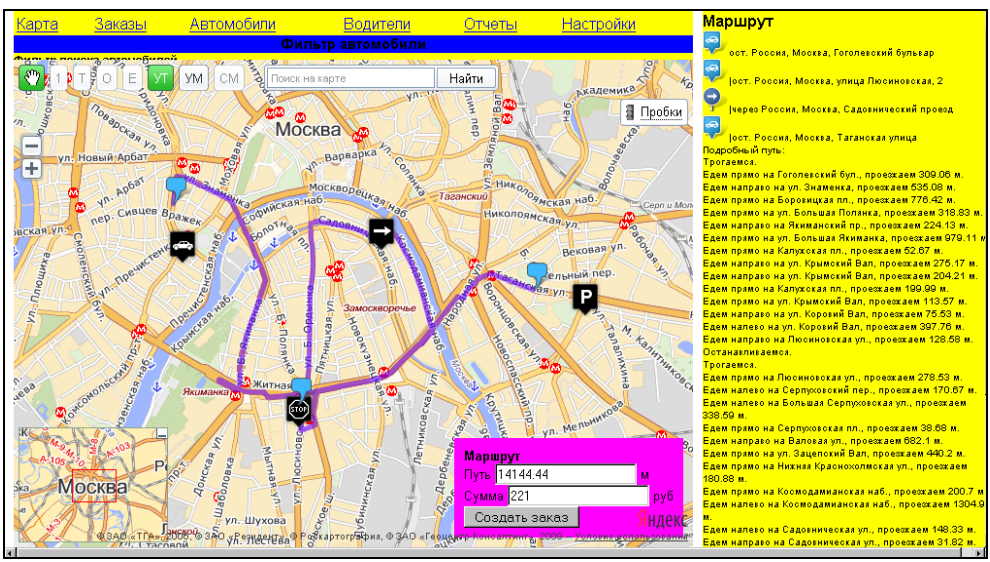


Рис. 4.13. Маршрутный лист

Листинг 4.39

```
// Прокладывание маршрута через массив точек
router1 = new YMaps.Router(arrrouter, arrrouter1,
            {viewAutoApply: 1,avoidTrafficJams:1});
```

```
// Добавляем маршрут на карту
YMaps.Events.observe(router1, router1.Events.Success, function()
{ // Добавляем маршрут на карту
  map.addOverlay(router1);
  var route;
  var segment;
  // формирование пути
  var path=["<b>Подробный путь:</b>"];
  var action = [];
  action["back"] = "назад";
  action["left"] = "налево";
  action["right"] = "направо";
  action["none"] = "прямо";
  action["slight left"] = "налево";
  action["slight right"] = "направо";
  action["hard left"] = "налево";
  action["hard right"] = "направо";
  var d=router1.getDistance();
  // стоимость проезда
  var s=calc(router1.getDistance(),router1.getNumRoutes()-1);
  document.getElementById("distance").value=d;
  document.getElementById("summa").value=s;
  for(var i1=0;i1<router1.getNumRoutes();i1++)
  { path.push("Трогаемся.");
    route=router1.getRoute(i1);
    for(var i2=0;i2<route.getNumRouteSegments();i2++)
    { segment=route.getRouteSegment(i2);
      path.push("Едем "+action[segment.getAction()]+" на "+
      segment.getStreet()+", проезжаем "+segment.getDistance()+" м.");
    }
    path.push("Останавливаемся.");
  }
  document.getElementById("route").innerHTML=path.join("<br>");
  document.getElementById("button_ok").disabled=false;
});
// обработка ошибок
YMaps.Events.observe(router1, router1.Events.GeocodeError, function(number)
{ alert('Ошибка при геокодировании точки ± ' + number);});
YMaps.Events.observe(router1, router1.Events.RouteError, function (number)
{ alert('Не удается проложить маршрут до точки ± ' + number);});
  button4.disable();
 });
}
// подсчет стоимости проезда
function calc(length_path,n_stop)
{ var summa=0;
  summa=pay_from+Math.round(pay_tarif*length_path/1000)+pay_stop*n_stop;
  return summa; }
```

Константы для расчета стоимости маршрута находятся в файле my.php (листинг 4.40).

Листинг 4.40

```
// для расчета стоимости
define(PAY_FROM,50.00);   // подача машины руб.
define(PAY_TARIF,10.00);  // руб./км
define(PAY_STOP,30.00);   // промежуточная остановка
```

Удаление маршрута с карты происходит по событию нажатия кнопки **УМ**. Обработчик события нажатия кнопки **УМ** представлен в листинге 4.41

Листинг 4.41

```
// Сброс по нажатию кнопки УМ (удалить маршрут)
YMaps.Events.observe(button3, button3.Events.Click, function ()
{ map.removeOverlay(router1);
  button4.enable();
  document.getElementById("route").innerHTML="";
  document.getElementById("button_ok").disabled=true;
});
```

При нажатии активной кнопки **Создать заказ** вызывается js-функция `create_zakaz()` (листинг 4.42), которая вызывает xAjax-функцию `Create_Zakaz()`.

Листинг 4.42

```
function create_zakaz()
{ var p=document.getElementById("points").innerHTML;
  var r=document.getElementById("route").innerHTML;
  var d=document.getElementById("distance").value;
  var s=document.getElementById("summa").value;
  xajax_Create_Order(p,r,d,s);
}
```

Программирование блока *Заказы*

При выборе пункта **Заказы** верхнего меню открывается блок **Заказы** (рис. 4.14).

Программы блока находятся в папке orders. Блок состоит из формы фильтра для поиска, блока вывода результатов и блока редактирования заказа (изменение статуса и привязка автомобиля такси к заказу). Фильтр позволяет осуществлять поиск заказов по промежутку дат, по автомобилям и по статусу заказа. Статусы заказов следующие:

- принят оператором;
- принят водителем;

- машина подана;
- в пути;
- выполнен;
- отказ клиента.

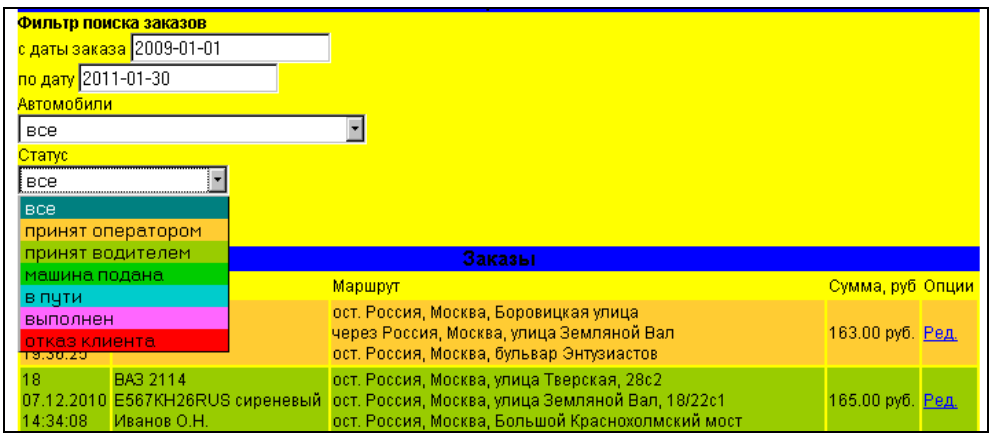


Рис. 4.14. Блок **Заказы**

Функция создания контента формы фильтра `f_filter_orders()` находится в файле orders\filter_orders.php, содержимое которого представлено в листинге 4.43. Даты выбираются из календаря (рис. 4.15). Для создания календаря используется js-библиотека epoch.

Листинг 4.43

```
<?php
// Контент для формы фильтра для выбора заказов
function f_filter_orders()
{ require_once('mybaza.php');
  $content="";
  $content.="<form id='FormFilterOrders' action='javascript:void();'
    onsubmit='xajax.$(\"ButtonFormFilterOrders\").disabled=true;
    xajax.$(\"ButtonFormFilterOrders\").value=\"Подождите...\";
    xajax_View_Orders(xajax.getFormValues(\"FormFilterOrders\"));'
    enctype='multipart/form-data';>";
  $content.="<input type=hidden id=pagesearch name=pagesearch value=1>";
  $content.="<b>Фильтр поиска заказов </b>";
  $content.="<br>с  даты заказа
     <input type=text id=data1 name=data1 value='".date('Y-m-d')."'>";
  $content.="<br>по дату <input type=text id=data2 name=data2
     value='".date('Y-m-d',strtotime('tomorrow'))."'>";
  $content.="<br> Автомобили <br><select name=car>
            <option value=0 selected>все";
```

```
  $query11="SELECT * FROM ".CARS." ";
  $rez11=mysql_query($query11);
  while($row11=mysql_fetch_assoc($rez11))
  { $content.="<option value='".$row11[id]."'>
               ".$row11[mark]." ".$row11[number];
  }
  $content.="</select>";
  $content.="<br>Статус <br>
            <select name=status>
            <option value='all' selected>все
            <option class='yes1' value='yes1'>принят оператором
            <option class='yes2' value='yes2'>принят водителем
            <option class='yes3' value='yes3'>машина подана
            <option class='yes4' value='yes4'>в пути
            <option class='yes5' value='yes5'>выполнен
            <option class='no1' value='no1'>отказ клиента
           </select>";
  $content.="<br><input type=submit id='ButtonFormFilterOrders'
            value='Показать'><br><br>";
  $content.="</form>";
  return $content;
}
?>
```

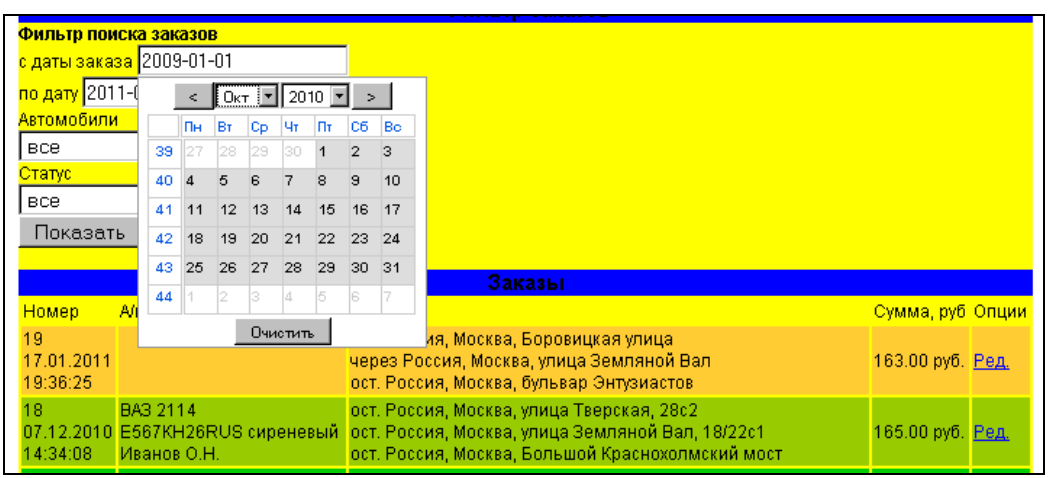


Рис. 4.15. Выбор даты

Постраничный результат выбора заказов по фильтру осуществляет xAjax-функция `View_Orders()`. Контент формирует функция `f_view_orders()`. Эти функции расположены в файле orders\view_orders.php, содержимое его представлено в листинге 4.44. Результат выводится в блоки с `id=block2` и `id=block21` (рис. 4.16). Заказы с различным статусом отображаются разным цветом.

Заказы				
Номер	А/м,Водитель	Маршрут	Сумма, руб	Опции
19 17.01.2011 19:36:25		ост. Россия, Москва, Боровицкая улица через Россия, Москва, улица Земляной Вал ост. Россия, Москва, бульвар Энтузиастов	163.00 руб.	Ред.
18 07.12.2010 14:34:08	ВАЗ 2114 E567KH26RUS сиреневый Иванов О.Н.	ост. Россия, Москва, улица Тверская, 28с2 ост. Россия, Москва, улица Земляной Вал, 18/22с1 ост. Россия, Москва, Большой Краснохолмский мост	165.00 руб.	Ред.
17 08.11.2010 13:47:07	ВАЗ 2114 E567KH26RUS сиреневый Иванов О.Н.	ост. Россия, Москва, улица Моховая, 13 ост. Россия, Москва, улица Красноказарменная, 4	150.00 руб.	Ред.
16 05.11.2010 17:05:45	ВАЗ 2110 E123KX26RUS белый Лавров И.С.	ост. Россия, Москва, улица Моховая, 15/1с1 ост. Россия, Москва, Большая Сухаревская площадь, 14/7 ост. Россия, Москва, Госпитальный переулок, 8	153.00 руб.	Ред.
15 03.11.2010 14:09:58	ВАЗ 2114 E567KH26RUS сиреневый Иванов О.Н.	ост. Россия, Москва, Страстной бульвар, 12к1 через Россия, Москва, улица Верхняя Сыромятническая, 7с2 ост. Россия, Москва, Кондрашевский тупик, 3	181.00 руб.	Ред.
14 03.11.2010 14:04:23		ост. Россия, Москва, улица Тверская, 7 ост. Россия, Москва, Золоторожская набережная, 10	120.00 руб.	Ред.
13 03.11.2010 14:01:56		ост. Россия, Москва, 1-й Колобовский переулок, 10к1 ост. Россия, Москва, Уланский переулок, 23/12	72.00 руб.	Ред.

Рис. 4.16. Вывод заказов постранично

Листинг 4.44

```
<?php
// Заказы по фильтру постранично
function View_Orders($Id)
{ $objResponse = new xajaxResponse();
  // получение контента
  $content2=f_view_orders($Id);
  // вывод в блоки
  $objResponse->assign("block2","innerHTML",$content2[0]);
  $objResponse->assign("block21","innerHTML",$content2[1]);
  // восстановить активность кнопки
  $objResponse->assign("ButtonFormFilterOrders","disabled",false);
  $objResponse->assign("ButtonFormFilterOrders","value","Показать");
  $objResponse->script("document.getElementById('block2').scrollIntoView();");
  return $objResponse;
}
// формирование контента
function f_view_orders($Id)
{ require_once('mybaza.php');
  require_once('my.php');
  $content=array();$content0.="";
  // формирование запроса
  $query0="SELECT COUNT(id) FROM ".ORDERS." WHERE id>0 ";
  $query0.="&& data1 >= '".$Id[data1]."' && data1 <='".$Id[data2]."'  ";
  if($Id[status]<>'all')
    $query0.="&& status='".$Id[status]."' ";
```

```
if($Id[car]>0)
  $query0.="&& id_cars='".$Id[car]."' ";
// получение результата и количества страниц
$rez0=mysql_query($query0);
$count=mysql_result($rez0,0);
$pages=ceil($count/NN3);
$page=min($Id[pagesearch],$pages);$poz=($page-1)*NN3;
if($count>0)
{ // заголовок таблицы
  $content0.="<table width=100%><tr>";
  $content0.="<td>Номер</td>";
  $content0.="<td>А/м,Водитель</td>";
  $content0.="<td>Маршрут</td>";
  $content0.="<td>Сумма, руб</td>";
  $content0.="<td>Опции</td>";
  $content0.="</tr>";
  $query0.=" ORDER BY data1 DESC LIMIT ".$poz.", ".NN3."";
  $query1=str_replace("COUNT(id)","*",$query0);
  $rez1=mysql_query($query1);
  while($row1=mysql_fetch_assoc($rez1))
  { $content0.="<tr>";
    // номер заказа
    $content0.="<td class='".$row1[status]."'>
    ".$row1[id]."<br>".date('d.m.Y',strtotime($row1[data1]))."
    <br>".date('H:i:s',strtotime($row1[data1]))."</td>";
    // машина водитель, даты
    $query2="SELECT mark,number FROM ".CARS." WHERE id=".$row1[id_cars]." ";
    $rez2=mysql_query($query2);
    $row2=mysql_fetch_assoc($rez2);
    $query3="SELECT fio FROM ".DRIVERS." WHERE id=".$row1[id_driver]." ";
    $rez3=mysql_query($query3);
    $row3=mysql_fetch_assoc($rez3);
    $content0.="<td class='".$row1[status]."'>".$row2[mark]."
         <br>".$row2[number]."<br>".$row3[fio]."</td>";
    $content0.="<td class='".$row1[status]."'>".$row1[route]."</td>";
    // сумма
    $content0.="<td class='".$row1[status]."'>".$row1[summa]." руб. </td>";
    $content0.="<td class='".$row1[status]."'>
          <a href='javascript:void()'
          onclick='xajax_Edit_Order(".$row1[id].");'>Ред.</a>
          </td>";
    $content0.="</tr>";
  }
  $content0.="</table>";
  // указатель страниц
... ... ...
```

```
  // возврат значений
  $content[0]=$content0;
  $content[1]=$content1;
  return $content;
}
?>
```

Информацию об автомобиле (включая статус) мы можем изменять — xAjax-функции `Edit_Order()` и `Add_Edit_Order()`, расположенные в файлах orders\edit_order.php и orders\add_edit_order.php соответственно. Содержимое этих файлов представлено в листинге 4.45. Форма редактирования заказа приведена на рис. 4.17. Для изменения статуса заказа формируются для выбора только возможные для текущего статуса изменения. Например, если статус заказа "принят оператором", то возможно изменение только на "принят водителем" и "отказ клиента". Если статус заказа "машина в пути", то возможно изменение только на "заказ выполнен" и "отказ клиента". Если заказ меняется на "принят водителем", то необходимо выбрать автомобиль из списка (рис. 4.18). Список состоит из автомобилей, имеющих статус "работает свободен" и "работает заказ", при этом вверху списка — свободные, от-

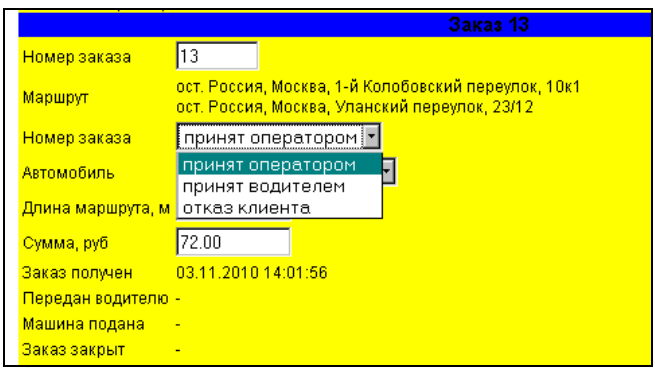


Рис. 4.17. Форма редактирования заказа

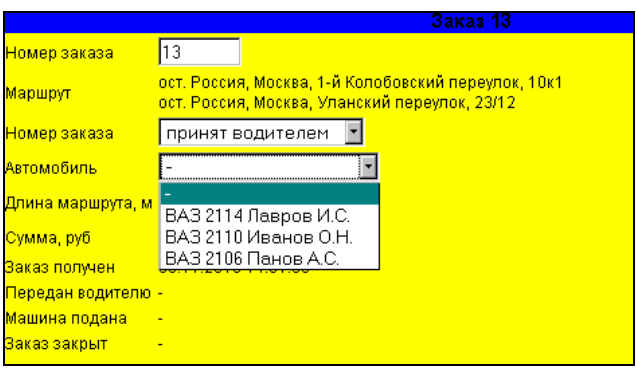


Рис. 4.18. Выбор автомобиля такси для заказа

сортированные по времени последнего выполнения заказа. Если мы меняем статус заказа на "принят водителем" и не выбираем такси, программа выдает ошибку (рис. 4.19). В заказе выводится информация о времени изменения для каждого статуса заказа (рис. 4.20).

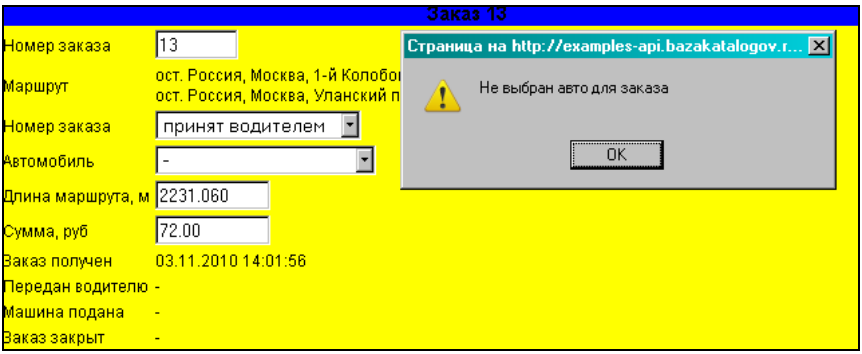


Рис. 4.19. Ошибка при изменении статуса заказа

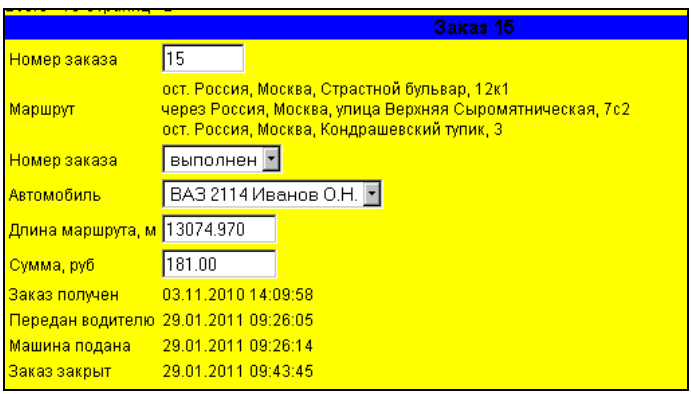


Рис. 4.20. Вывод времени изменения статуса заказа

Листинг 4.45

```
<?php
// Контент для формы редактирования заказа
function Edit_Order($id)
{ $objResponse = new xajaxResponse();
  require_once('mybaza.php');
  // запрос для получения данных о заказе
  $query0="SELECT * FROM ".ORDERS." WHERE id='".$id."' ";
  $rez0=mysql_query($query0);
  $row0=mysql_fetch_assoc($rez0);
  $content="";
  $content.="<form id='FormEditOrder' action='javascript:void();'
```

```
     onsubmit='xajax.$(\"ButtonFormEditOrder\").disabled=true;
     xajax.$(\"ButtonFormEditOrder\").value=\"Подождите...\";
     xajax_Add_Edit_Order(xajax.getFormValues(\"FormEditOrder\"));'
     enctype='multipart/form-data';>";
  $content.="<table><tr><td>Номер заказа</td>";
  $content.="<td><input type=text name=id size=6 value='".$row0[id]."'
             readonly></td></tr>";
  $content.="<tr><td>Маршрут</td>";
  $content.="<td>".$row0[route]."</td></tr>";
  // массив возможных вариантов изменения статуса
  $status=array("yes1" => array("yes1"=>"принят оператором",
                    "yes2"=>"принят водителем","no1"=>"отказ клиента"),
                "yes2" => array("yes2"=>"принят водителем",
                    "yes3"=>"машина подана","no1"=>"отказ клиента"),
                "yes3" => array("yes3"=>"машина подана",
                    "yes4"=>"в пути","no1"=>"отказ клиента"),
                "yes4" => array("yes4"=>"в пути",
                    "yes5"=>"выполнен","no1"=>"отказ клиента"),
                "yes5" => array("yes5"=>"выполнен"),
                "no1" => array("no1"=>"отказ клиента"),
   );
  // формирование контента
  $content.="<tr><td>Статус заказа</td>";
  $content.="<td><select name=status>";
  foreach($status[$row0[status]] as $st=>$val)
      $content.="<option value='".$st."'>".$val;
  $content.="</select></td></tr>";
  $content.="<tr><td>Автомобиль</td>";
  ... ... ...
  $content.="<br><input type=submit id='ButtonFormEditOrder'
             value='Изменить' ".$disabled."><br>";
  $content.="<input type=button onclick='
             document.getElementById(\"zag3\").innerHTML=\"
            <table></table>\";
             document.getElementById(\"block3\").innerHTML=\"
             <table></table>\";'
             value='Закрыть'><br>";
  $content.="</form>";
  // заголовок — номер заказа
  $objResponse->assign("zag3","innerHTML","Заказ ".$row0[id]);
  $objResponse->assign("block3","innerHTML",$content);
  $objResponse->script("document.getElementById('block3').scrollIntoView();");
  return $objResponse;
}
// Изменение в базе данных заказа
function Add_Edit_Order($Id)
{ $objResponse = new xajaxResponse();
```

```
  require_once('mybaza.php');
  $arr1=explode(";",$Id[id_cars]);
  $id_cars=$arr1[0]; $id_driver=$arr1[1];
  // проверка правильности заполнения
  ... ... ...
  // изменение
  $query1="UPDATE ".ORDERS." SET
          id_cars='".$id_cars."', id_driver='".$id_driver."',
          status='".$Id[status]."' ";
  $data=date('Y-m-d H:i:s');
  if($Id[status]=='yes2')
    $query1.=",data2='".$data."' ";
  elseif($Id[status]=='yes3')
    $query1.=",data3='".$data."' ";
  elseif($Id[status]=='yes5' || $Id[status]=='no1')
    $query1.=",data4='".$data."' ";
  else
    ;
  $query1.=" WHERE id='".$Id[id]."' ";
  $rez1=mysql_query($query1);
  ... ... ...
  // для обновления страницы просмотра заказов
  $script="document.getElementById('ButtonFormFilterOrders').click();";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

4.3. Создание карты местности с несколькими слоями пользовательских карт

Подробные карты Яндекса на момент написания книги существуют далеко не для всех даже средних городов, не говоря уже о мелких. Но имея карту какого-нибудь города, мы можем разбить ее на тайлы (т. е. фрагменты, от англ. *tile* — плитка) и, используя API, отображать ее в Яндекс.Картах на своем сайте. В данном проекте мы для одной из областей России, не имеющей подробных Яндекс.Карт, создадим несколько слоев пользовательских карт городов. На нашей карте мы будем просматривать различные объекты, обозначенные метками, и информацию об этих объектах (наименование, адрес, фото, видеоролик). Объекты будем брать из базы данных. Проект можно посмотреть в Интернете по адресу **http://examples-api.bazakatalogov.ru/yandex/4-3/map.php**. Наборы тайлов для трех городов находятся на прилагаемом компакт-диске в папке glava_04\4-3 (папки town1, town2, town3).

4.3.1. Создание пользовательских карт городов

Для начала создадим пользовательские карты нескольких городов. Желательно иметь карты в формате PNG, где пустые участки сделаны прозрачными. Для разбивки карт на тайлы воспользуемся программой Подготовка слоя тайлов из API Яндекс.Карт. Загрузим в программу нашу карту. Чтобы мы могли затем выводить нашу пользовательскую карту на нужном месте, необходимо осуществить привязку карты к реальным координатам. Требуются четыре точки (рис. 4.21).

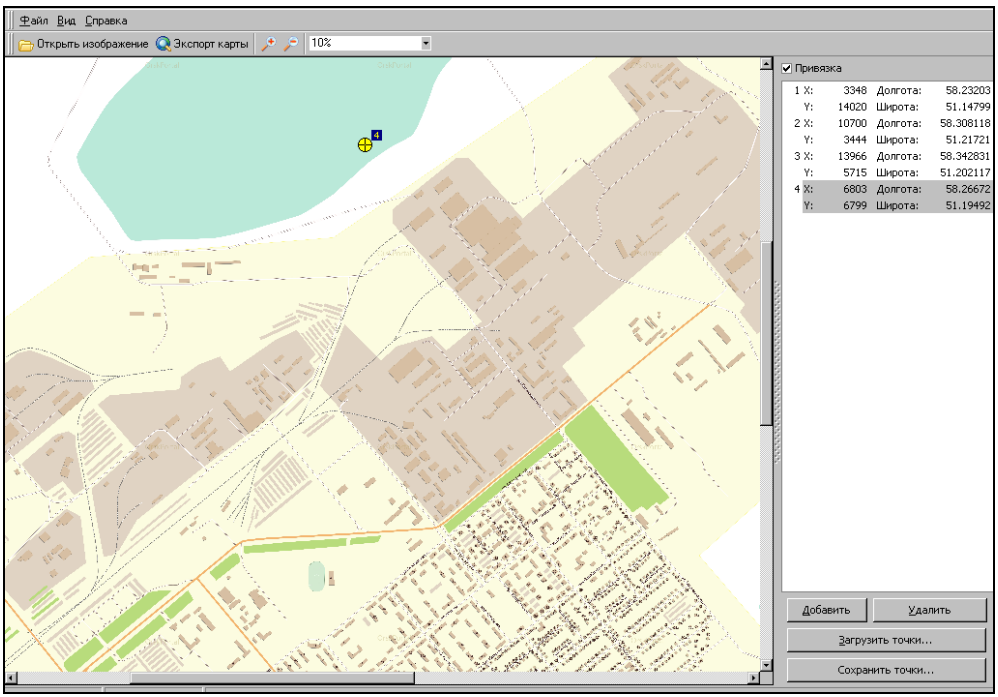


Рис. 4.21. Привязка карты к реальным координатам

Нажимаем кнопку **Экспорт карты** и в настройках выбираем 8—10 уровней, формат — PNG. Выбираем каталог для сохранения файлов тайлов и нажимаем кнопку **Готово** (рис. 4.22).

Для больших файлов процесс формирования может занять значительное время, иногда несколько часов. Запасемся терпением и получим в выбранном каталоге для хранения тайлов несколько папок файлов (для каждого масштаба своя папка с тайлами).

4.3.2. Размещение пользовательских слоев на Яндекс.Карте

Приступим к созданию слоев с изображением карты.

Сначала определим источник данных для карты. Источник данных поставляет тайлы для слоя, в зависимости от текущих координат центра карты и коэффициента

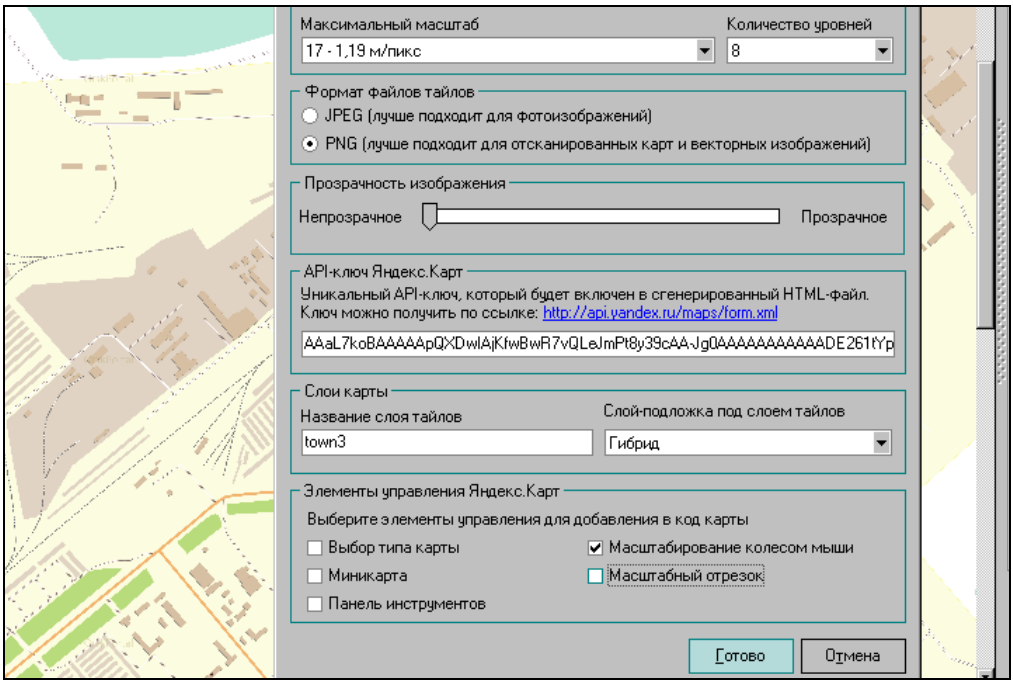


Рис. 4.22. Экспорт карты

масштабирования. Чтобы создать источник данных, используйте класс `YMaps.TileDataSource`. Конструктор класса принимает следующие параметры:

- `tileUrlTemplate` — шаблон, по которому строится URL тайла;
- `isTransparent` — флаг, указывающий, что тайлы прозрачны;
- `smoothZoomEnabled` — флаг, разрешающий плавное масштабирование тайлов.

В шаблоне `tileUrlTemplate` можно использовать следующие специальные конструкции:

- `%d` — заменяется числом от 1 до 4 в зависимости от номера тайла. Используется для распределения нагрузки между несколькими доменами;
- `%c` — заменяется `x&y&z`, где `x` — номер тайла по горизонтали, `y` — номер тайла по вертикали, `z` — коэффициент масштабирования.

Для нашего слоя это выглядит так:

```
myData1=new YMaps.TileDataSource("./town1/%z/tile-%x-%y.png", true, false);
```

Далее необходимо перекрыть метод `getTileUrl()`, чтобы задать новые правила формирования URL тайла по шаблону в зависимости от тайловых координат и коэффициента масштабирования:

```
myData1.getTileUrl = function (tile, zoom) {
  return this.getTileUrlTemplate().replace(/%x/i, tile.x).replace(/%y/i,
  tile.y).replace(/%z/i, zoom);}
```

Затем надо создать слой карты, который будет получать тайлы из источника данных, и добавить его на карту:

```
var myLayer = new YMaps.Layer(myData);
map.addLayer(myLayer);
```

С помощью метода `add()` помещаем созданный слой в хранилище слоев карты `YMaps.Layers`:

```
// Функция, создающая экземпляры слоя (конструктор)
var myLayer = function () {
  return new YMaps.Layer(myData)}
// Добавляет слой в хранилище
YMaps.Layers.add("test#layer", myLayer);
```

Чтобы получить слой из хранилища по ключу, необходимо использовать метод `get()` класса `YMaps.Layers`:

```
var myLayer = YMaps.Layers.get("test#layer");
```

Для нашего примера мы будем создавать слои для трех городов, для каждого города два слоя — обычная карта и карта дорог с названиями улиц и нумерацией домов. Код для создания одного слоя приведен в листинге 4.46.

Листинг 4.46

```
options1 = {tileUrlTemplate: "./town1/%z/tile-%x-%y.png",
           controls: {typeControl: false,
                      miniMap: false,
                      toolBar: false,
                      scaleLine: false },
           scrollZoomEnabled: true,
           mapCenter: new YMaps.GeoPoint(58.5324744883062, 51.2298678572725),
           backgroundMapType: YMaps.MapType.HYBRID,
           mapZoom: 10,
           isTransparent: true,
           smoothZooming: false,
           layerKey: "my#layer1",
           mapType: { name: "town1",
                      textColor: "#000000" },
           copyright: ""
           },
map = new YMaps.Map(document.getElementById("YMapsID")),
myData1 = new YMaps.TileDataSource(options1.tileUrlTemplate,
options1.isTransparent, options1.smoothZooming);
myData1 = new YMaps.TileDataSource("./town1/%z/tile-%x-%y.png", true, false);
myData1.getTileUrl = function (tile, zoom)
{ return this.getTileUrlTemplate().replace(/%x/i, tile.x).replace(/%y/i,
tile.y).replace(/%z/i, zoom);
}
```

```
MyLayer1 = function ()
{ return new YMaps.Layer(myData1); }
YMaps.Layers.add(options1.layerKey, MyLayer1);
mapLayers1 = options1.backgroundMapType ?
options1.backgroundMapType.getLayers() : [],
myMapType1 = new YMaps.MapType(YMaps.jQuery.merge(mapLayers1, [
options1.layerKey ]), options1.mapType.name, { textColor:
options1.mapType.textColor });
```

В качестве подложки для слоев выбираем HYBRID, это связано с тем, что для территорий, где Яндекс не создал свои карты, установление подложки типа MAP будет ограничивать масштаб показа значением 14—15 и не позволит показать более подробную пользовательскую карту. При размещении нескольких слоев видим только один из них. Для показа всех слоев и перехода между ними создадим пользовательский элемент выбора конкретного города.

4.3.3. Создание переключателя выбора городов

Пользовательский элемент создадим в виде блока, на котором расположен элемент выбора нужного объекта из раскрывающегося списка (рис. 4.23). Код для создания и помещения на карту пользовательского элемента представлен в листинге 4.47.

Рис. 4.23. Пользовательский элемент выбора города

Листинг 4.47

```
// создание пользовательского элемента
map.addControl(new Choice_Objects());
function Choice_Objects()
{ // Добавление на карту
  this.onAddToMap = function (map, position)
  { var cont_container="<span>Выбор города<br>
      <select onchange='view_choice(this.selectedIndex)'>
      <option value=0 disabled>Выбрать
```

```
      <option value=1 selected>Город1
      <option value=2>Город2
      <option value=3>Город3</select><span>"
    this.container = YMaps.jQuery(cont_container);
    this.map = map;
    this.position = new YMaps.ControlPosition(YMaps.ControlPosition.LEFT_RIGHT,
       new YMaps.Size(10, 10));
    this.container.css({ position: "absolute",
                         zIndex: YMaps.ZIndex.CONTROL,
                         background: '#f0f', listStyle: 'none',
                         padding: '10px', margin: 0 });
    // Применение позиции к управляющему элементу
    this.position.apply(this.container);
    // Добавление на карту
    this.container.appendTo(this.map.getContainer());
  }
  // Удаление с карты
  this.onRemoveFromMap = function ()
  { this.container.remove();
    this.container = this.map = null;
  };
}
```

Для выбора типа карты для одного города (схема города или схема дорог) создадим переключатель **Схема/Спутник**. Для него будем использовать внешний элемент управления Яндекс.Карт — переключатель на панели инструментов, код создания и размещения на карте которого представлен в листинге 4.48.

Листинг 4.48

```
toolbar1 = new YMaps.ToolBar([]);
// радиокнопка
button11 = new YMaps.ToolBarRadioButton('group1', {
    caption: "Схема", hint: "Схема"},{selected:true});
button12 = new YMaps.ToolBarRadioButton('group1', {
    caption: "Спутник", hint: "Спутник"});
// добавить на toolbar
toolbar1.add(button11);
toolbar1.add(button12);
// добавить панель инструментов на карту
map.addControl(toolbar1,
               new YMaps.ControlPosition(YMaps.ControlPosition.TOP_RIGHT,
                                         new YMaps.Point(10, 30)));
//
YMaps.Events.observe(button11, button11.Events.Select, function () {
  view_choice(flagmap);
}, toolbar1);
```

```
//
YMaps.Events.observe(button11, button11.Events.Deselect, function () {
  view_choice(flagmap);
}, toolbar1);
```

При выборе города либо переключении **Схема/Спутник** вызывается js-функция `view_choice()`. Она выбирает нужный слой из хранилища и плавно переходт к центру выбранной карты. Содержимое функции представлено в листинге 4.49.

Листинг 4.49

```
function view_choice(arg)
{ flagmap=arg;
  if(button11.isSelected())
  { if(arg==1)
    { YMaps.Layers.get("my#layer1");
      map.setCenter(map.getCenter(), 13, myMapType1);
      map.panTo(options1.mapCenter, {flying:1});
      Ymaps.setMaxZoom(17);
    }
    if(arg==2)
    { YMaps.Layers.get("my#layer2");
      map.setCenter(map.getCenter(), 13, myMapType2);
      map.panTo(options2.mapCenter, {flying:1});
      map.setMaxZoom(17);
    }
    if(arg==3)
    { YMaps.Layers.get("my#layer3");
      map.setCenter(map.getCenter(), 13, myMapType3);
      map.panTo(options3.mapCenter, {flying:1});
    }
  }
  else
  { if(arg==1)
    { YMaps.Layers.get("my#layer101");
      map.setCenter(map.getCenter(), 13, myMapType101);
      map.panTo(options101.mapCenter, {flying:1});
    }
    if(arg==2)
    { YMaps.Layers.get("my#layer102");
      map.setCenter(map.getCenter(), 13, myMapType102);
      map.panTo(options102.mapCenter, {flying:1});
    }
    if(arg==3)
    { YMaps.Layers.get("my#layer103");
      map.setCenter(map.getCenter(), 13, myMapType103);
```

```
      map.panTo(options103.mapCenter, {flying:1});
    }
  }
}
```

4.3.4. Размещение на картах меток

Создание и редактирование меток мы рассмотрим в следующей главе. Здесь нам необходимо вывести из базы данных уже существующие метки для предприятий и организаций городов нашей территории. Код для вывода меток из базы данных представлен в листинге 4.50.

Листинг 4.50

```
// Вывести все метки из базы
<?php
  $text01="";
  $query01="SELECT * FROM info_metka WHERE visible='yes' ORDER BY id ASC ";
  $rez01=mysql_query($query01);$i=0;
  while($row01=mysql_fetch_assoc($rez01))
  { $query02="SELECT name FROM style_metka WHERE id=".$row01[id_metka]."";
    $rez02=mysql_query($query02);
    $row02=mysql_fetch_assoc($rez02);
    $text01.="placemark[".$i."] = new YMaps.Placemark(new
    YMaps.GeoPoint(".$row01[lng].",
    ".$row01[lat]."),{hideIcon: false,style: '".$row02[name]."'});";
    $text01.="placemark[".$i."].name = '".$row01[name]."';";
    // если есть картинка
    if($row01[img]<>"")
    { $row01[description]="<img src=\"resize_200.php?pic="
      .$row01[img]."\"><br>".$row01[description];
    }
    $text01.="placemark[".$i."].description = '".$row01[description]."';";
    $text01.="map.addOverlay(placemark[".$i."]);";
    $text01.="YMaps.Events.observe(placemark[".$i."],
              placemark[".$i."].Events.Click, function (obj)
    { xajax_Info_Marker(".$row01[id].");
    });";
    $i++;
  }
  echo $text01;
?>
```

В балун для каждой метки заносится информация о каждой записи (название организации, дополнительная информация, если есть фото, уменьшенное до 200 пикселов). Для каждой метки формируется обработчик события `Events.Click`,

который кроме открытия балуна подгружает в блок, расположенный под картами, фото объекта или ролик формата SWF для данного объекта (рис. 4.24). В случае отсутствия для данной метки фото или ролика (и при начальной загрузке) подгружаем стандартную картинку (рис. 4.25). Подгрузку осуществляет xAjax-функция `Info_Marker()`, расположенная в файле info_marker.php, содержимое которого представлено в листинге 4.51. Для подгрузки flash-ролика используется js-библиотека swfobject, для подключения которой необходимо в заголовочную часть включить следующую строку:

```
<script type="text/javascript" src="js/swfobject.js"></script>
```

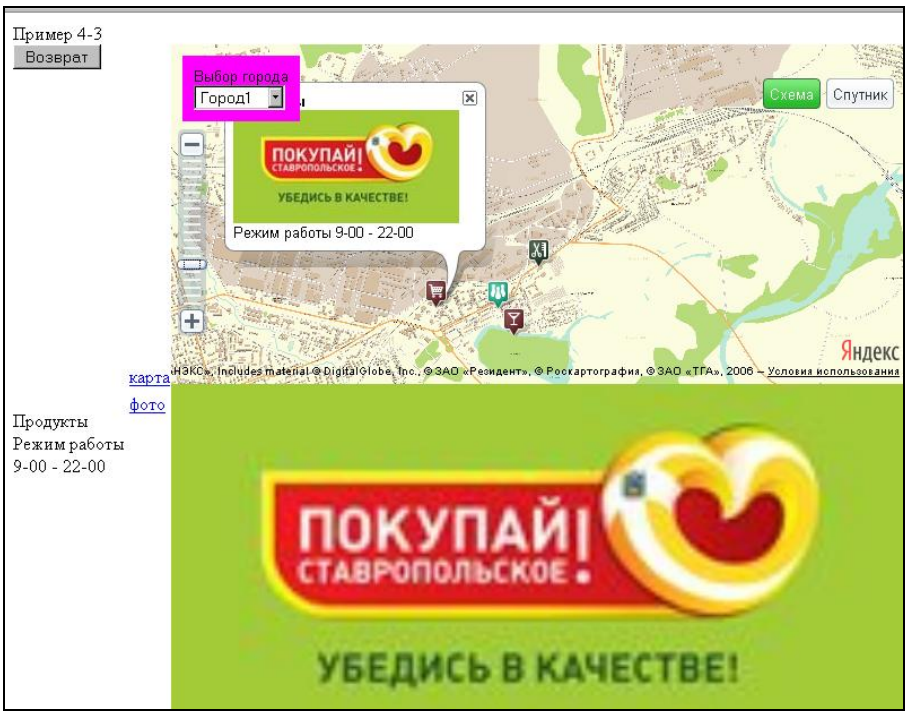


Рис. 4.24. При выборе метки

Листинг 4.51

```
<?php
function Info_Marker($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  require_once("my.php");
  // получение данных метки
  $query1="SELECT * FROM info_metka WHERE id=".$Id." ";
  $rez1=mysql_query($query1);
  $row1=mysql_fetch_assoc($rez1);
```

```
  // вывод инфо в левый блок
  $text1="<div id=leftbottom1>".$row1[name]."</div>";
  $text1.="<div id=leftbottom2>".$row1[description]."</div>";
  $text1.="<div id=leftbottom3>".$row1[info]."</div>";
  // вывод картинки или ролика
  if(substr_count($row1[img],"swf")>0)       // ролик
  { $text2="flash1 = new  SWFObject('".$row1[img]."',
               'mymovie','650','300','7','#336699');flash1.write('photo');";
    $objResponse->script($text2);
  }
  elseif($row1[img]<>"")    // фото
  { $text2="<img src='".$row1[img]."' width=650px>";
    $objResponse->assign("photo","innerHTML",$text2);
    $objResponse->assign("photoinfo","innerHTML","");
  }
  else  // фотозаглушка
  { $text2="<img src='no.jpg' width=650px>";
    $objResponse->assign("photo","innerHTML",$text2);
    $objResponse->assign("photoinfo","innerHTML","");
  }
  $objResponse->script("document.getElementById('rightbottom')
               .scrollIntoView();");
  return $objResponse;
}
?>
```

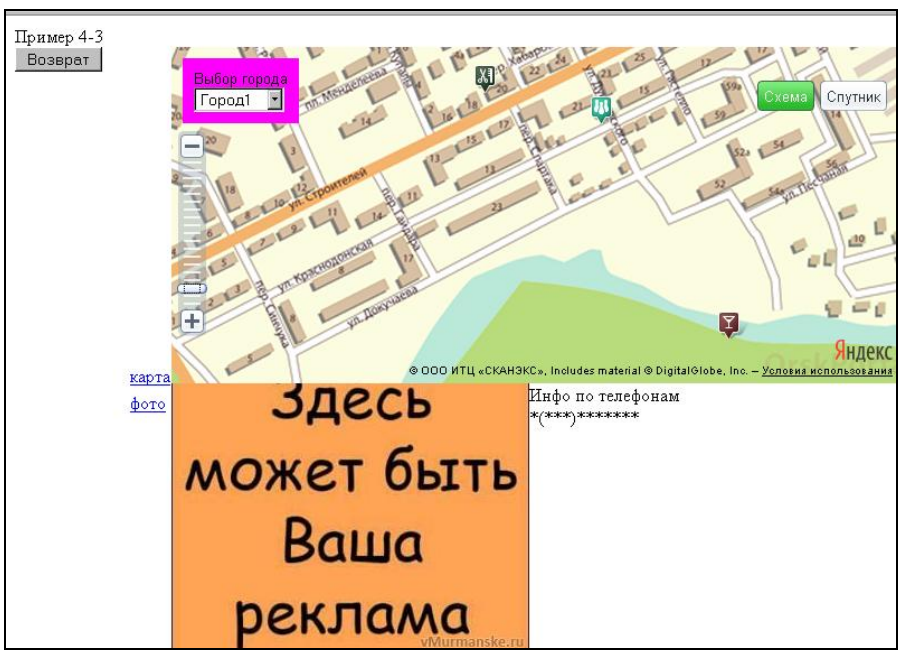


Рис. 4.25. Вывод стандартной картинки

В скрипте реализована возможность размещения карты, либо фото (или ролика) на всю высоту блока. При щелчке по ссылке **карта** или **фото** вызывается xAjax-функция `Full()`, которая перераспределяет место под карту или фото в зависимости от входящего аргумента:

- `1` — скрыть фото, карта на весь экран (рис. 4.26);
- `2` — скрыть карту, фото (или ролик) на весь экран;
- `3` — карта на пол-экрана, фото на пол-экрана.

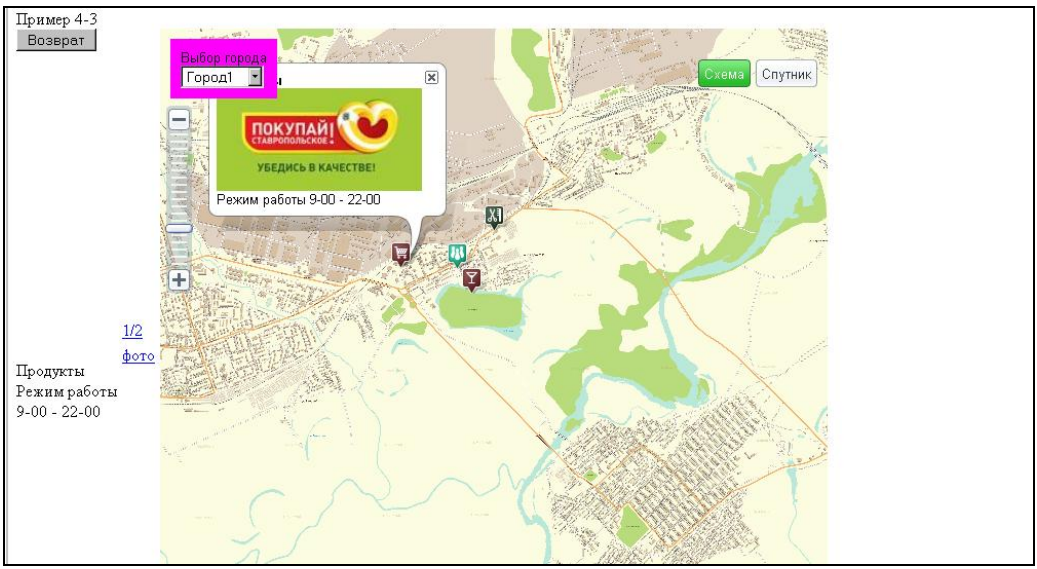


Рис. 4.26. Вывод карты на весь экран

Функция `Full()` расположена в файле full.php, содержимое которого представлено в листинге 4.52

Листинг 4.52

```
<?php
function Full($Id)
{ $objResponse = new xajaxResponse();
  switch($Id)
  { case 1:
        $objResponse->assign("YMapsID","visibility","visible");
        $objResponse->assign("YMapsID","style.height","650px");
        $objResponse->assign("rightbottom","style.height","0px");
        $middle2='<table>';
        $middle2.='<tr><td><a href="javascript:void();"
              onclick="xajax_Full(3)">1/2</a></td></tr>';
        $middle2.='<tr><td><a href="javascript:void();"
              onclick="xajax_Full(2)">фото</a></td></tr>';
```

```
        $middle2.='</table>';
        $objResponse->assign("middle2","innerHTML",$middle2);
        break;
    case 2:
        $objResponse->assign("YMapsID","visible","hidden");
        $objResponse->assign("YMapsID","style.height","0px");
        $objResponse->assign("rightbottom","style.height","600px");
        $objResponse->assign("photo","style.width","650px");
        $objResponse->assign("photoinfo","innerHTML","");
        $middle2='<table>';
        $middle2.='<tr><td><a href="javascript:void();"
               onclick="xajax_Full(1)">карта</a></td></tr>';
        $middle2.='<tr><td><a href="javascript:void();"
               onclick="xajax_Full(3)">1/2</a></td></tr>';
        $middle2.='</table>';
        $objResponse->assign("middle2","innerHTML",$middle2);
        break;
    case 3:
        $objResponse->assign("YMapsID","visibility","visible");
        $objResponse->assign("YMapsID","style.height","300px");
        $objResponse->assign("rightbottom","style.height","300px");
        $objResponse->assign("photo","style.width","650px");
        $objResponse->assign("photoinfo","innerHTML","");
        $middle2='<table>';
        $middle2.='<tr><td><a href="javascript:void();"
               onclick="xajax_Full(1)">карта</a></td></tr>';
        $middle2.='<tr><td><a href="javascript:void();"
               onclick="xajax_Full(2)">фото</a></td></tr>';
        $middle2.='</table>';
        $objResponse->assign("middle2","innerHTML",$middle2);
        break;
    default:
        break;
  }
  return $objResponse;
}
?>
```

4.3.5. Скрытие/показ меток при изменении масштаба

При уменьшении масштаба метки начинают сливаться, поэтому предусмотрим возможность удалять метки при уменьшении масштаба до определенного значения и заново создавать при увеличении. При событии изменения масштаба и центра карты проверяем значение текущего масштаба и при необходимости вызываем функции `del_all_metka()` и `create_all_metka()` (листинг 4.53).

Листинг 4.53

```
// событие при изменении масштаба и центра карты
YMaps.Events.observe(map, map.Events.Update, function ()
{ zoom_new = map.getZoom();
  if(zoom_new!=zoom_old)
  { if(zoom_old>=zoom_gr && zoom_new<zoom_gr)
       del_all_metka();
    if(zoom_old<zoom_gr && zoom_new>=zoom_gr)
       create_all_metka();
  }
  zoom_old=zoom_new;
});
function del_all_metka()
{ for(var i=0;i<placemark.length;i++)
  { map.removeOverlay(placemark[i]); }
}
function create_all_metka()
{ <?php
  $text01="";
  $query01="SELECT * FROM info_metka WHERE visible='yes' ORDER BY id ASC
            LIMIT 0, 10 ";
  $rez01=mysql_query($query01);$i=0;
  while($row01=mysql_fetch_assoc($rez01))
  { $query02="SELECT name FROM style_metka WHERE id=".$row01[id_metka]." ";
    $rez02=mysql_query($query02);
    $row02=mysql_fetch_assoc($rez02);
    $text01.="placemark[".$i."] = new YMaps.Placemark(new
      YMaps.GeoPoint(".$row01[lng]."
     ,".$row01[lat]."),{hideIcon: false,style: '".$row02[name]."'});";
    $text01.="placemark[".$i."].name = '".$row01[name]."';";
    // если есть фото
    if($row01[img]<>"")
    { $row01[description]="<img src=\"resize_200.php?
      pic=".$row01[img]."\"><br>".$row01[description];
    }
    $text01.="placemark[".$i."].description = '".$row01[description]."';";
    $text01.="map.addOverlay(placemark[".$i."]);";
    $text01.="YMaps.Events.observe(placemark[".$i."],
            placemark[".$i."].Events.Click, function (obj)
    { new_lng=obj.getCoordPoint().getLng();
      new_lat=obj.getCoordPoint().getLat();
      xajax_Info_Marker(".$row01[id].");
    });";
    $i++;
  }
  echo $text01;
  ?>
}
```

4.3.6. Передача параметров в скрипт и возврат значений из скрипта

Посмотрим, как можно использовать скрипт для связи с другими скриптами. Допустим, нам необходимо показать месторасположение предприятия из формы создания (редактирования) предприятия для сайта каталога предприятий. Или указать расположение объекта недвижимости для сайта недвижимости. Для городов, имеющих подробные карты Яндекса, это не проблема. Там можно использовать сервис "Геокодирование" API Яндекс.Карт. Для пользовательских карт такого механизма нет. Но мы можем в форме, допустим, создания (редактирования) предприятия создать фрейм, поместить в него наш скрипт, передать параметры и получить координаты метки, которую мы создадим в нашем скрипте. Эти координаты поместим в запись базы данных для данного предприятия и потом при переходе по ссылке **Показать на карте** передавать эти координаты в карту для отображения метки. Допустим, мы имеем форму, как показано на рис. 4.27. В сети вы можете посмотреть данный проект по адресу **http://examples-api.bazakatalogov.ru/yandex/4-3/index2.php**.

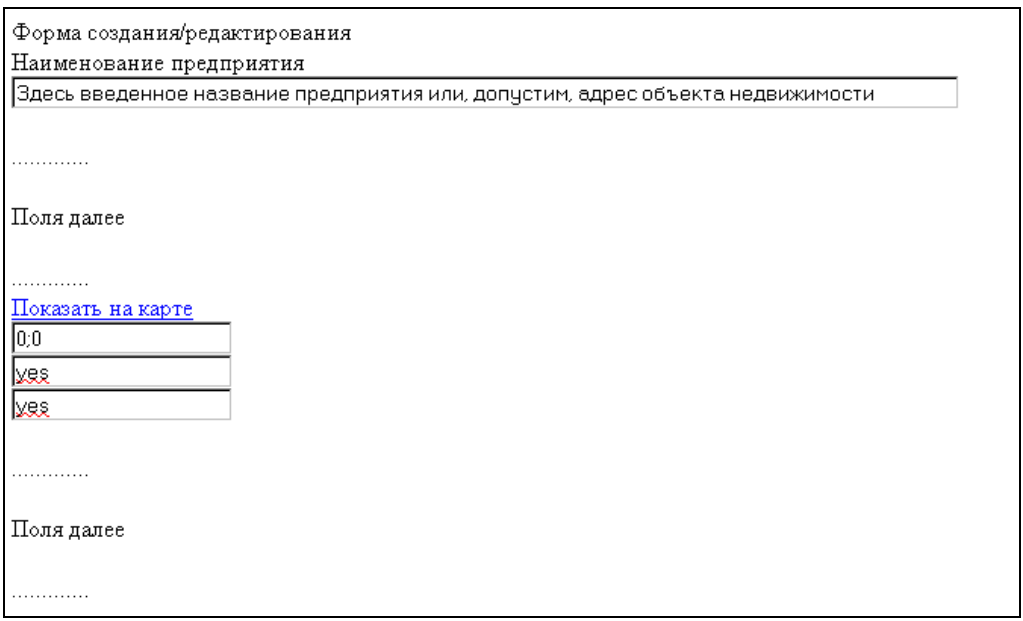


Рис. 4.27. Форма создания редактирования

Поля `input` для наглядности имеют тип `text`, к тому же их надо скрывать. Их назначение:

- `maps` — координаты метки (`lat;lng`), если нет метки "0;0";
- `edit_maps` — флаг редактирования:
  - `yes` — возможно редактирование метки;
  - `no` — просмотр метки;

- `type_maps` — тип формы:
  - `yes` — объявления недвижимости;
  - `no` — предприятия.

При щелчке по ссылке **Показать на карте** открывается фрейм (рис. 4.28), URL фрейма формируется из данных полей `maps`, `edit_maps`, `type_maps`. Эти данные наш скрипт получает во фрейме методом `get()`. Содержимое файла index2.php представлено в листинге 4.54.

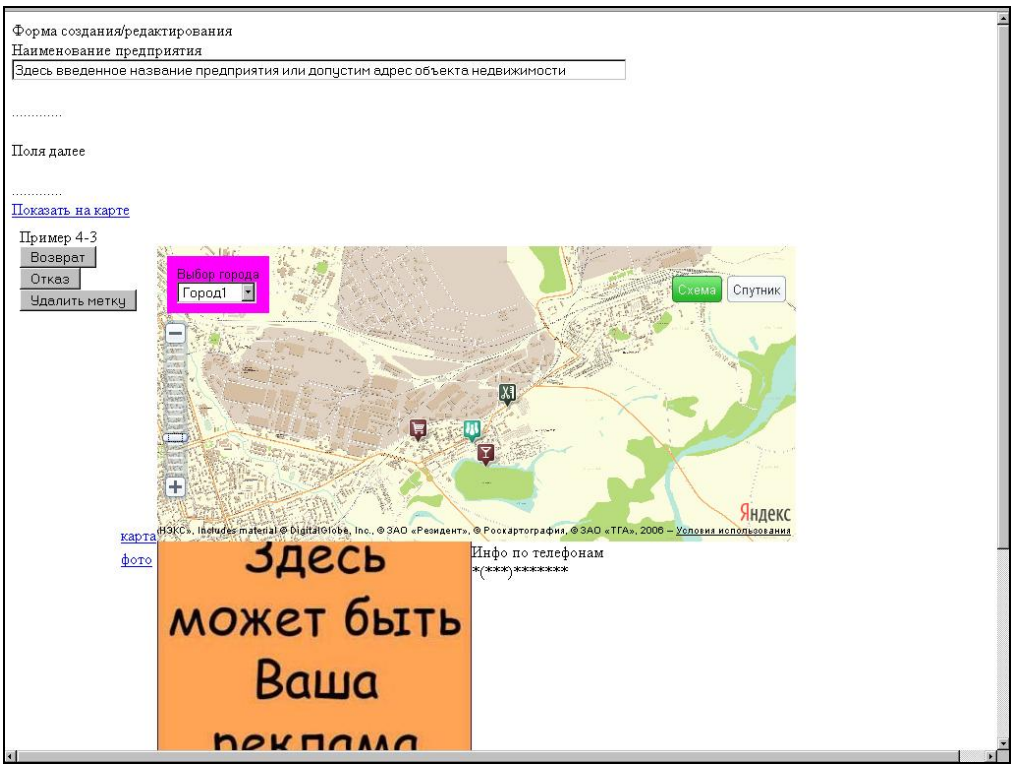


Рис. 4.28. Открытие скрипта map.php из формы во фрейме

Листинг 4.54

```
<html>
<head>
  <script>
    function show_map()
    { document.getElementById("fr1").style.height=600;
      document.getElementById("fr1").src="http://maps.orskportal.ru/map.php";
    }
  </script>
</head>
```

```
<body>
  Форма создания/редактирования<br>
  <form name=form1>
  Наименование предприятия<br>
  <input type=text id=name_org name=name_org size=100
    value="Здесь введенное название предприятия или, допустим, адрес объекта
недвижимости"><br>
<br>... ... ........<br>
<br>Поля далее<br>
<br>... ... ........<br>
<!-- Начало вставки -->
<a href="javascript:void();"
   onclick='var url="map.php?koords="+document.forms.form1.elements.maps.value;
url=url+"&name="+document.forms.form1.elements.name_org.value;
url=url+"&edit="+document.forms.form1.elements.edit_maps.value;
url=url+"&type_maps="+document.forms.form1.elements.type_maps.value;
document.getElementById("fr1").style.height=600;
document.getElementById("fr1").src=url;'>
            Показать на карте</a>
<!-- form1 — name формы  -->
<!-- мне нужно название форм, например  -->
<!-- form1 — создания объявления  -->
<!-- form2 — редактирования объявления  -->
<!-- form3 — создания организации  -->
<div id=maps>
 <iframe id="fr1" src="" height=0 width=1024 frameborder=no></iframe>
</div>
<input type=text id='maps' name='maps' value="0;0"><br>
<!--type=hidden 1-lat,2-lng  -->
<input type=text id='edit_maps' name='edit_maps' value="yes"><br>
<!--type=hidden yes-edit,no-view  -->
<input type=text id='type_maps' name='type_maps' value="yes"><br>
<!--type=hidden: yes — объявления, no — организации  -->
<!-- Конец вставки  -->
<br>... ... ........<br>
<br>Поля далее<br>
<br>... ... ........<br>
</form>
</body>
</html>
```

Если на карте нет метки, ее можно создать, щелкнув по нужному месту карты, затем ее можно переместить или удалить (рис. 4.29). При этом в скрытое поле заносятся координаты метки. Если при открытии фрейма передаются ненулевые координаты, метка помещается на карту. Функции, реализующие эти возможности, представлены в листинге 4.55.

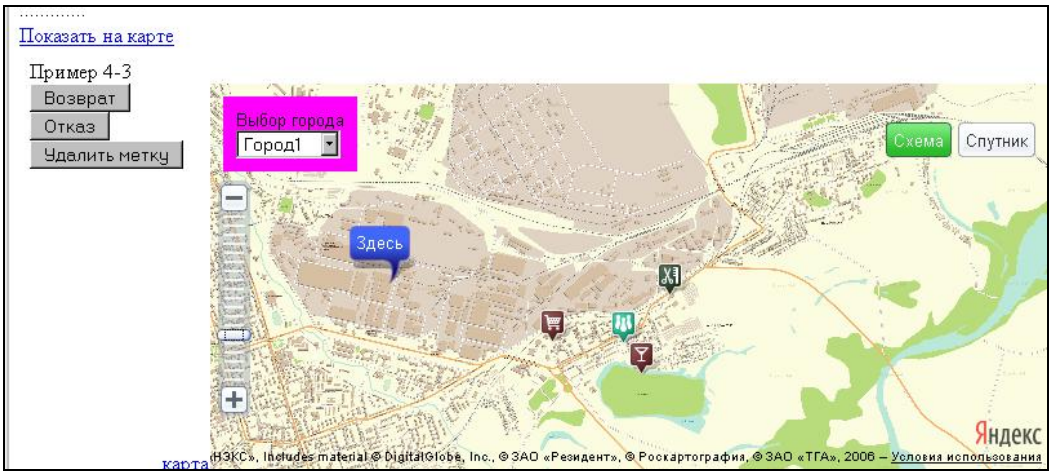


Рис. 4.29. Создание метки

Листинг 4.55

```
// если метка есть — добавить на карту
<?php
  if($metka_yes==1)
  { $txt='metka = new YMaps.Placemark(new YMaps.GeoPoint('.$lng.','.$lat.'),
        {hideIcon: false,draggable:
        '.(($edit=="yes")?("true"):("false")).',style:"default#nightPoint"});
    metka.name = metka_name;
    metka.setIconContent(icon_text);
    map.addOverlay(metka);';
    if($edit=='yes')
    { $txt.='YMaps.Events.observe(metka, metka.Events.DragEnd, function (obj)
      { document.forms.yandex1.elements.koords.value=
            obj.getGeoPoint().getLng()+";"+obj.getGeoPoint().getLat();
      });';
    }
    echo $txt;
  }
?>
// событие по клику — добавление метки
<?php
  if($edit=='yes')
    echo 'YMaps.Events.observe(map, map.Events.Click, function (map,mEvent)
    { if(metka_yes<1)
      { metka = new YMaps.Placemark(mEvent.getGeoPoint(),{hideIcon:
         false,draggable: true,style:"default#nightPoint"});
        metka_yes=1;
        document.forms.yandex1.elements.koords.value=
          mEvent.getGeoPoint().getLng()+";"+mEvent.getGeoPoint().getLat();
```

```
        metka.setIconContent(icon_text);
        metka.name = metka_name;
        map.addOverlay(metka);
        //metka.openBalloon();
        YMaps.Events.observe(metka, metka.Events.DragEnd, function (obj)
        { document.forms.yandex1.elements.koords.value=
            obj.getGeoPoint().getLng()+";"+obj.getGeoPoint().getLat();
        });
    }
  }, this);';
?>
```

При нажатии кнопки **Возврат** в родительский фрейм возвращаются координаты созданной метки (листинг 4.56).

Листинг 4.56

```
function back_save()
{ var p1=window.parent.document;
  var el=p1.getElementById('fr1');
  el.style.height=0;
  el.src="";
  if(type_maps=="yes")
  { var el1=p1.forms.form1.elements.maps;
    el1.value=document.forms.yandex1.elements.koords.value;
  }
  if(type_maps=="no")
  { var el1=p1.forms.form1.elements.maps;
    el1.value=document.forms.yandex1.elements.koords.value;
  }
}
```

Полное содержимое файла map.php представлено на прилагаемом компакт-диске. Этот файл находится в папке glava_04\4-3.

4.4. Создание, редактирование меток для карты местности с несколькими слоями пользовательских карт

Для создания полнофункциональной карты необходимо иметь админ-профиль, где можно создавать новые метки, редактировать содержимое и местоположение существующих меток, удалять ненужные метки. Необходим в админ-панели и поиск нужных меток по фильтру. Создавать админ-панель будем с помощью технологии Ajax, чтобы весь процесс работы происходил без перезагрузки страницы. Использовать будем библиотеку xAjax и встроенную в API Яндекс.Карт библиотеку

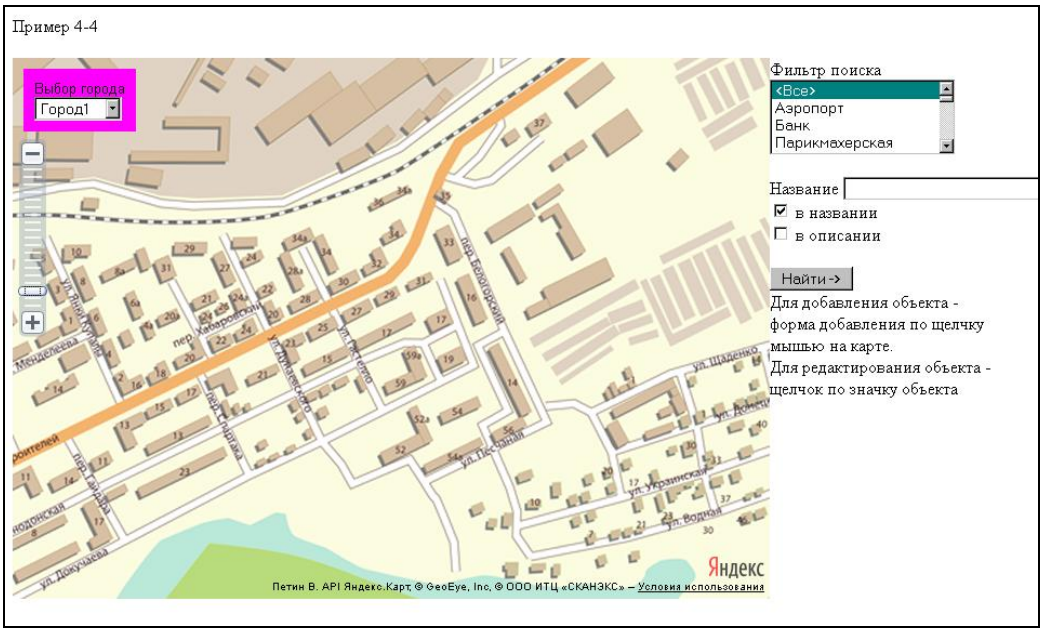


Рис. 4.30. Панель администратора

jQuery. В Интернете данный проект представлен по адресу **http://examples-api.bazakatalogov.ru/yandex/4-3/map_admin.php**. При заходе по данному адресу открывается форма авторизации. Введите пароль 12345. Попадете в страницу админ-панели (рис. 4.30).

Теперь приступим к программированию админ-панели.

4.4.1. Проектирование базы данных

Для реализации данного проекта нам потребуются две таблицы базы данных:

- `style_metka` — информация о стилях используемых меток;
- `info_metka` — информация об объектах, нанесенных на карту.

Таблица `style_metka` содержит информацию о значке для метки. В API предусмотрен ряд встроенных стилей для значков меток. Их и будем использовать. Большой перечень значков позволит охватить очень многие городские объекты. Значки предусмотрены для отображения аэропортов, вокзалов, станций метро, магазинов, больниц и пр. Полный перечень значков можно посмотреть по адресу **http://api.yandex.ru/maps/jsapi/doc/ref/reference/styles.xml**.

Структура таблицы `style_metka` следующая:

- `id` — первичный ключ;
- `name` — имя стиля метки;
- `info` — название значка;

- `type` — тип значка:
  - `1` — встроенный в Яндекс;
  - `2` — пользовательский;
- `visible` — возможность использования значка:
  - `yes` — доступен;
  - `no` — скрыт;
- `sort` — поле для сортировки значков в форме выбора.

Дамп для создания таблицы `style_metka` приведен в листинге 4.57.

Листинг 4.57

```
CREATE TABLE `style_metka` (
`id` int(9) NOT NULL AUTO_INCREMENT,
`name` varchar(30) NOT NULL,
`info` varchar(30) NOT NULL,
`type_metka` set('1', '2') NOT NULL default '1',
`visible` set('yes', 'no') NOT NULL default 'yes',
`sort` int(11) NOT NULL,
UNIQUE KEY `id` (`id`)
) ENGINE = MYISAM DEFAULT CHARSET = cp1251;
```

В таблице `info_metka` содержится информация о нанесенных на карту объектах. Структура таблицы следующая:

- `id` — первичный ключ;
- `id_metka` — идентификатор значка для данного объекта;
- `lng` — географическая долгота;
- `lat` — географическая широта;
- `name` — наименование объекта;
- `description` — информация об объекте;
- `info` — дополнительная информация об объекте;
- `img` — фото или ролик об объекте;
- `visible` — видимость на карте;
  - `yes` — видим;
  - `no` — скрыт.

Дамп для создания таблицы `info_metka` приведен в листинге 4.58.

Листинг 4.58

```
CREATE TABLE `info_metka` (
`id` int(9) NOT NULL AUTO_INCREMENT,
`id_metka` int(9) NOT NULL,
```

```
`lng` float(18, 14) NOT NULL,
`lat` float(18, 14) NOT NULL,
`name` varchar(30) NOT NULL,
`description` text NOT NULL,
`info` tinytext NOT NULL,
`img` varchar(20) NOT NULL,
`visible` set('yes', 'no') NOT NULL default 'yes',
UNIQUE KEY `id` (`id`)
) ENGINE = MYISAM DEFAULT CHARSET = cp1251;
```

4.4.2. Авторизация администратора

При входе в систему администрирования появляется форма (рис. 4.31). При программировании будем использовать библиотеку xAjax, чтобы все изменения на странице и в базе данных происходили без перезагрузки страницы. Для авторизации мы закрываем карту. Для этого создаем пользовательский элемент по размеру карты, на который наносим форму авторизации. При нажатии кнопки **OK** вызывается xAjax-функция `Auth()`, которая сверяет введенный пароль с паролем администратора (пароль хранится в файле настроек my.php) и в случае совпадения удаляет пользовательский элемент и показывает карту. Код создания пользовательского элемента с формой авторизации представлен в листинге 4.59.

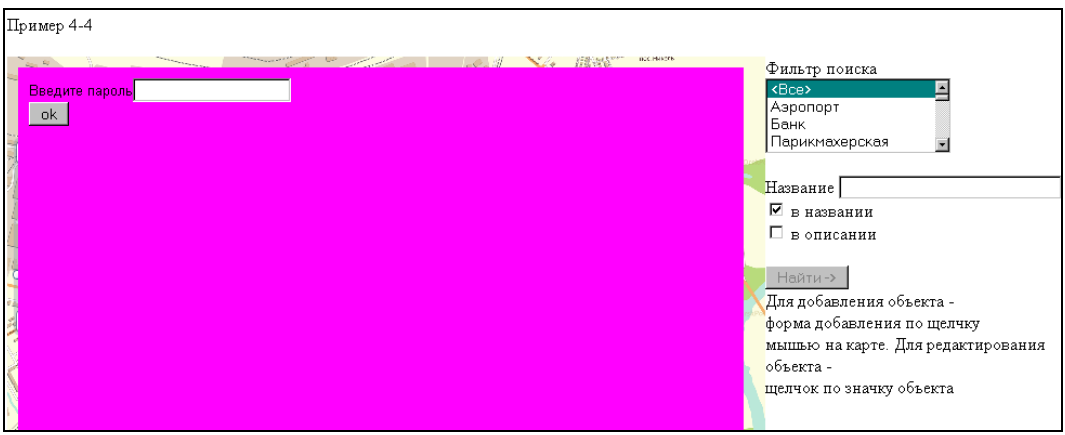


Рис. 4.31. Форма авторизации

Листинг 4.59

```
// создание пользовательского элемента авторизации
auth=new Form_Auth();
map.addControl(auth);
function Form_Auth()
{ // Добавление на карту
  this.onAddToMap = function(map, position)
```

```
  { var cont_container="<form id=form_auth name=form_auth
      action='javascript:void()' onsubmit='xajax_Auth(
      xajax.getFormValues(\"form_auth\"));'>Введите пароль
     <input name=my_password><br><input type=submit value='ok'></form>"
    this.container = YMaps.jQuery(cont_container);
    this.map = map;
    this.position = new YMaps.ControlPosition(
            YMaps.ControlPosition.LEFT_RIGHT, new YMaps.Size(10, 10));

    this.container.css({  position: "absolute",
                          zIndex: YMaps.ZIndex.CONTROL,
                          background: '#f0f', listStyle: 'none',
                          padding: '10px',    margin: 0,
                          width: '650px',     height: '450px',  });
    // Применение позиции к управляющему элементу
    this.position.apply(this.container);
    // Добавление на карту
    this.container.appendTo(this.map.getContainer());
  }
  // Удаление с карты
  this.onRemoveFromMap = function ()
  { this.container.remove();
    this.container = this.map = null;
  };
}
```

Функция `Auth()` находится в файле auth.php, содержимое которого представлено в листинге 4.60.

Листинг 4.60

```
<?php
function Auth($Id)
{ $objResponse = new xajaxResponse();
  require_once("my.php");
  if($Id[my_password]!=MY_PASSWORD)
  { $objResponse->alert("Неверный пароль !!!");
    return $objResponse;
  }
  $script="auth.onRemoveFromMap();";
  $objResponse->script($script);
  $objResponse->alert("Авторизация успешно !!!");
  $script="document.forms.FormSearch.ButtonFormSearch.disabled=false;";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

4.4.3. Вывод карты

Вывод карты аналогичен выводу карты в примере 4-3. Единственное отличие — убираем за ненадобностью панель инструментов с кнопками-переключателями **Схема**/**Спутник**. Последовательность та же:

1. Создание и размещение на карте пользовательских слоев.
2. Создание пользовательского элемента — переключателя выбора городов.
3. Вывод на карту меток из базы данных.

Все мы это подробно рассмотрели в *разд. 4.3*.

Что нам нужно добавить:

- ❒ создание новой метки;
- ❒ редактирование содержимого метки;
- ❒ изменение координат метки;
- ❒ поиск меток по фильтру.

Этим мы и займемся далее.

4.4.4. Добавление новой метки

Для вызова формы создания новой метки необходимо щелкнуть мышью по нужному месту на карте. При этом в месте щелчка создаем невидимую метку и открываем балун, содержимое которого и есть форма создания новой метки (рис. 4.32).

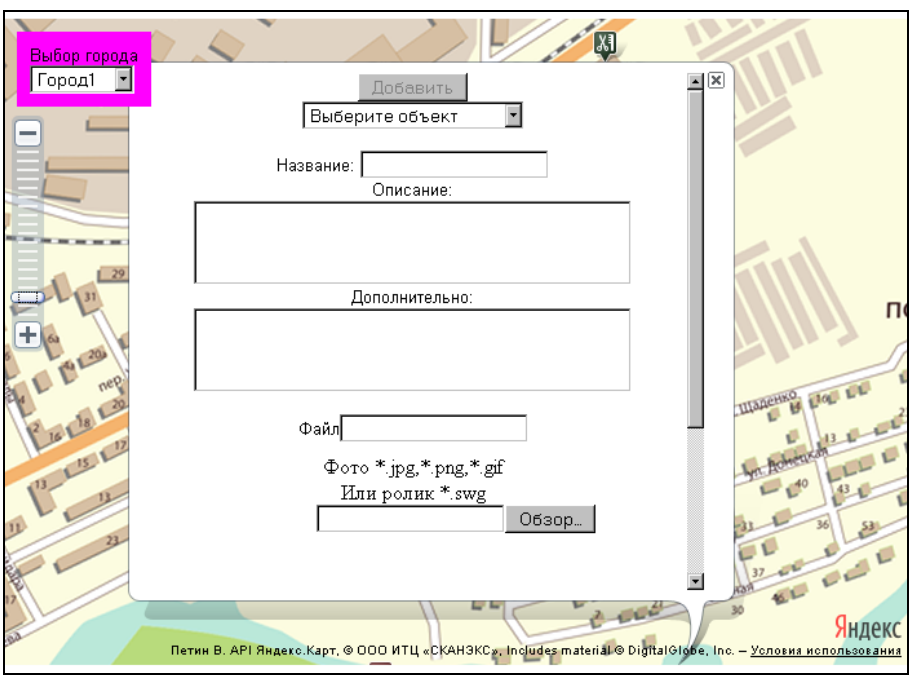


Рис. 4.32. Форма создания метки

Код создания формы представлен в листинге 4.61.

Листинг 4.61

```
// При щелчке по карте добавить форму создания
YMaps.Events.observe(map, map.Events.Click, function (map, mEvent)
{ // всплывающая форма для выбора метки
  var Htmlselect ="<div id=imagePreview style='float:left; width:30px;'></div>
<select name=imageMarker id=imageMarker
onchange='previewImages(this.form.imageMarker.options);'>
<option value=''>Выберите объект";
    <?php
    // вывод названий значков
    $text1="";
    $query1="SELECT * FROM style_metka WHERE visible='yes' ORDER BY id ASC ";
    $rez1=mysql_query($query1);
    while($row1=mysql_fetch_assoc($rez1))
      {$text1.="<option value=".$row1[name]." >".$row1[info];}
    ?>
    Htmlselect=Htmlselect+'<?php echo $text1; ?>';
    Htmlselect=Htmlselect+"</select>";
    // форма
    var myHtml ='<center>
    <form id="formadd_point" name="formadd_point" method="post"
    action="javascript:void(null);"
    onsubmit="xajax_Add_Marker(xajax.getFormValues
    (\'formadd_point\'));">
    <input name="subpoint" id="subpoint" type="submit"
    value="Добавить"><br>'+Htmlselect+'<br>
    <br>Название: <input name="namepoint" id="namepoint"
    type="text" size=20 maxlength=50 />
    ... ... ...
    </form></center>';
  // открыть балун
  map.openBalloon(mEvent.getGeoPoint(), myHtml);
  // обработка события при изменении типа метки
  YMaps.jQuery("#imageMarker").change(function()
  { YMaps.jQuery("#imagePreview").empty();
    if (YMaps.jQuery("#imageMarker").val()!="")
    { YMaps.jQuery("#imagePreview").append("<img src=\"" +
      "<?php echo DIR_IMG; ?>"
      +escape(YMaps.jQuery("#imageMarker").val())
      + ".png\" />");
       YMaps.jQuery("#subpoint").attr("disabled",false); }
     else
     { YMaps.jQuery("#imagePreview").append("");
```

```
        YMaps.jQuery("#subpoint").attr("disabled",true); }
  });
  YMaps.jQuery("#subpoint").attr("disabled",true);
})
```

Получаемые по событию `map.Events.Click` координаты заносятся в скрытое поле. Выбор значка осуществляем из списка `select`. Подгрузку фотографии (или ролика) для объекта осуществляем без перезагрузки страницы, помещая форму выбора во фрейм и возвращая название загруженного файла в поле нашей формы из фрейма. Процедуру, реализующую загрузку файлов на сервер без перезагрузки страницы, мы рассмотрим в *разд. 4.4.8*. Заполенная форма имеет вид, представленный на рис. 4.33.

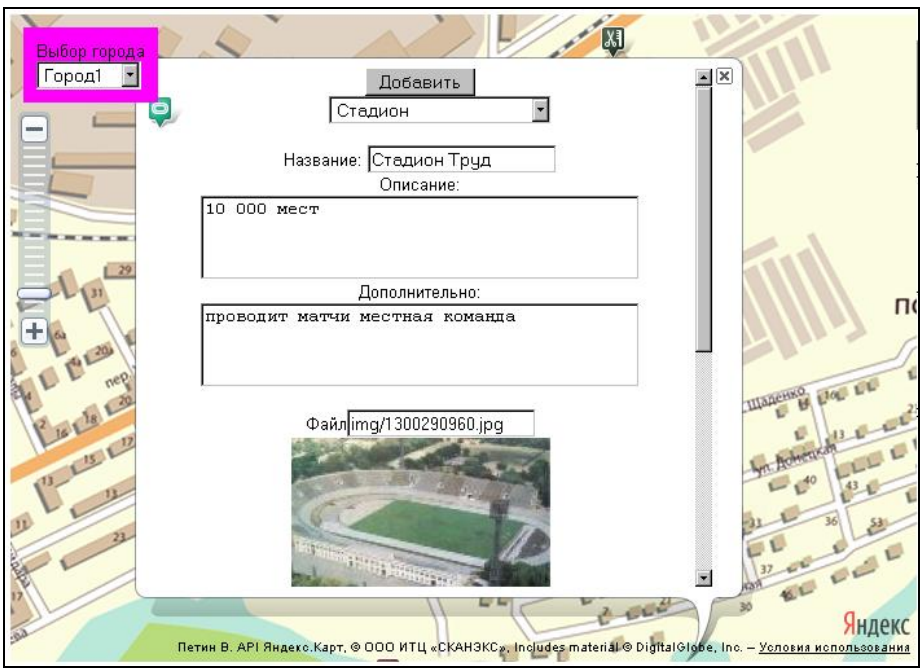


Рис. 4.33. Вид заполненной формы создания метки

При нажатии кнопки **Добавить** вызывается xAjax-функция `Add_Marker()`, в которую передаются данные формы. Эта функция выполняет следующие действия:

- заносит данные об объекте в базу данных;
- создает метку с выбранным пользователем стилем;
- создает содержимое балуна с информацией об объекте;
- в балун заносятся также ссылки для редактирования и удаления созданной метки;
- помещает метку на карту и центр карты устанавливает в созданной метке.

xAjax-функция `Add_Marker()` находится в файле add_marker.php, содержимое которого представлено в листинге 4.62.

Листинг 4.62

```
<?php
function Add_Marker($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  // заменить переводы строки на <br>
  $Id[descriptionpoint]=str_replace("\r","<br>",$Id[descriptionpoint]);
  ... ... ...
  // найти id выбранного значка для метки
  $query1="SELECT id FROM style_metka WHERE name='".$Id[imageMarker]."' ";
  $rez1=mysql_query($query1);
  $row1=mysql_fetch_assoc($rez1);
  // записать данные в базу данных
  $query2="INSERT INTO info_metka SET
           id_metka='".$row1[id]."', name='".$Id[namepoint]."',
           lng='".$Id[pcoordLng]."',lat='".$Id[pcoordLat]."',
           description='".$Id[descriptionpoint]."',
           info='".$Id[infopoint]."', img='".$Id[marker_img]."' ";
  $rez2=mysql_query($query2);
  $query3="SELECT name FROM style_metka WHERE id=".$row1[id]." ";
  $rez3=mysql_query($query3);
  $style=mysql_result($rez3,0);
  if(!$rez2)
  { $objResponse->alert($query2."Ошибка занесения в базу !!!");
    return $objResponse;
  }
  $maxid=mysql_insert_id();
  $objResponse->assign("maxid","value",$maxid);
  // js-код для создания метки
  $script="placemark[".$maxid."] = new YMaps.Placemark(new
      YMaps.GeoPoint(".$Id[pcoordLng].", ".$Id[pcoordLat]."),
     {draggable:true,hasHint: true, hideIcon: false, style:
     \"".$style."\"});";
  // js-код для содержимого балуна
  $script.="placemark[".$maxid."].name ='".$Id[namepoint]."';";
  $description=$Id[descriptionpoint];
  if($Id[marker_img]<>"")
  { $descriptionpoint="<br><img src=\"resize_200.php?
    pic=".$Id[marker_img]."\"><br>".$descriptionpoint;
  }
  $description.="<br><a href='javascript:void();'
    onclick='xajax_Edit_Marker(".$maxid.");'>Редактировать</a>";
  $description.="<br><a href='javascript:void();'
    onclick='xajax_Delete_Marker(".$maxid.");'>Удалить</a>";
  $script.="placemark[".$maxid."].description =\"".$description."\";";
```

```
  // Добавляет метку на карту
  $script.="map.addOverlay(placemark[".$maxid."]);";
  // установить центр карты
  $script.="map.setCenter(new YMaps.GeoPoint(".$Id[pcoordLng].",
            ".$Id[pcoordLat]."),12);";
  $objResponse->script($script);
  $balun="<br><b>".$Id[namepoint]."</b><br>".$description;
  $script="map.getBalloon().setContent(\"".$balun."\");";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

4.4.5. Редактирование содержимого метки

При щелчке по метке открывается балун, в котором отображается информация для данной метки (рис. 4.34). Кроме этого, в нижней части балуна находятся две ссылки — **Редактировать** и **Удалить**.

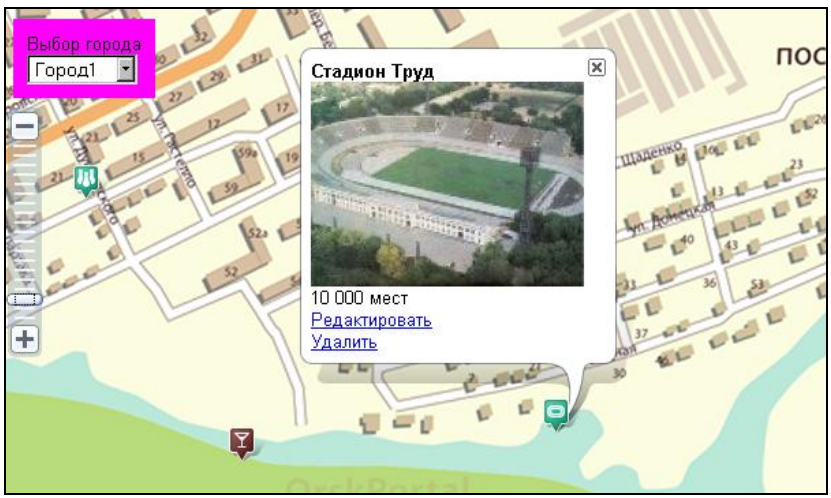


Рис. 4.34. Балун с информацией о метке

При щелчке по ссылке **Редактировать** вызывается xAjax-функция `Edit_Marker()`, которая получает информацию из базы данных о данной метке и в открытый балун заносит новый контент — форму редактирования информации для данной метки (рис. 4.35). Функция `Edit_Marker()` находится в файле edit_marker.php, содержимое которого представлено в листинге 4.63.

Листинг 4.63

```
<?php
function Edit_Marker($Id)
```

```
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  require_once("my.php");
  // получить информацию о метке из базы данных
  $query1="SELECT * FROM info_metka WHERE id=".$Id." ";
  $rez1=mysql_query($query1);
  $row1=mysql_fetch_assoc($rez1);
  $query2="SELECT name,info FROM style_metka WHERE id='".$row1[id_metka]."' ";
  $rez2=mysql_query($query2);
  $row2=mysql_fetch_assoc($rez2);
  $img="".DIR_IMG.urlencode($row2[name]).".jpg";
// формирование списка значков для выбора
  $text01="<div id=imagePreview1 style=\"float:left; width:30px;\">
  <img src=\"".$img."\"></div><select name=imageMarker1 id=imageMarker1
  onchange=\"previewImages1(this.form.imageMarker1.options);\">";
  $query01="SELECT * FROM style_metka WHERE visible='yes'
           ORDER BY sort ASC ";
  $rez01=mysql_query($query01);
  while($row01=mysql_fetch_assoc($rez01))
  { $selected="";
    if($row1[id_metka]==$row01[id])  $selected="selected";
    $text01.="<option value=".$row01[name]." ".$selected.">
             ".$row01[id]." — " .$row01[info];
  }
  $text01.="</select>";
// форма
  $text1="";
  $text1.="<center><form id=\"formadd_point\" name=\"formadd_point\"
    method=\"post\" action=\"javascript:void(null);\"
    onsubmit=\"xajax_Add_Edit_Marker(
    xajax.getFormValues(formadd_point));\">
    <input name=\"subpoint1\" id=\"subpoint1\" type=\"submit\"
    value=\"Изменить\">
    <input name=\"id1\" id=\"id1\" type=\"hidden\"
    value=\"".$row1[id]."\"><br>
    ... ... ...
   </form></center>";
  // js-код для изменения контента открытого балуна
  $script="map.getBalloon().setContent('".$text1."');";
  $objResponse->script($script);
  // занести содержание метки в поля формы
  $objResponse->assign("namepoint","value",$row1[name]);
  $objResponse->assign("descriptionpoint","value",$row1[description]);
  $objResponse->assign("marker_img","value",$row1[img]);
  return $objResponse;
}
?>
```

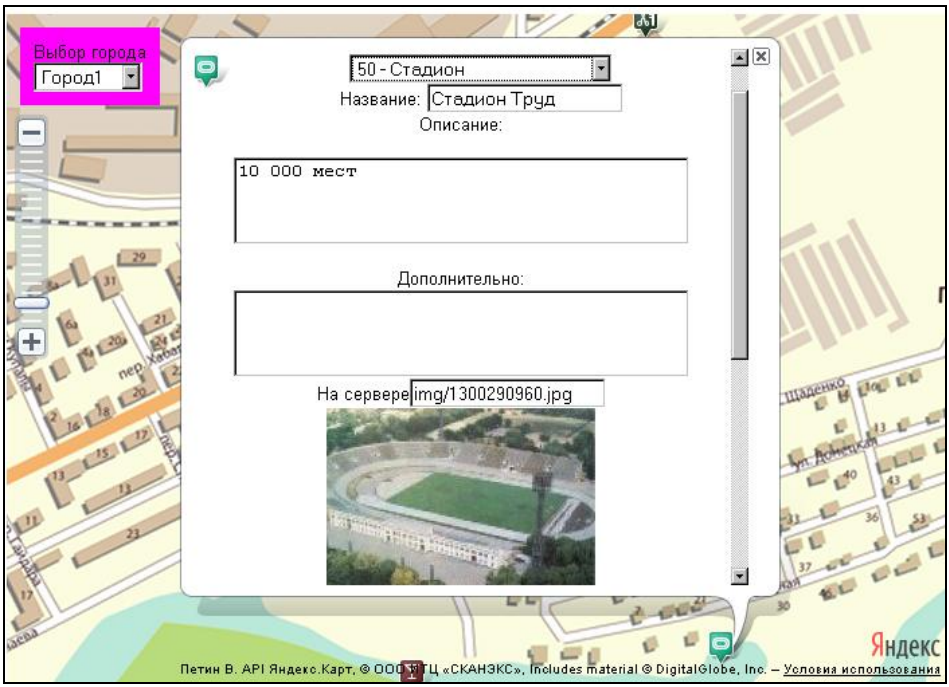


Рис. 4.35. Форма редактирования информации о метке

При нажатии кнопки **Изменить** вызывается xAjax-функция `Add_Edit_Marker()`, которая выполняет следующие действия:

- заносит измененные данные об объекте в базу данных;
- формирует измененный контент с информацией об объекте для содержимого балуна;
- заносит измененную информацию в открытый балун;
- в балун заносятся также ссылки для редактирования и удаления созданной метки.

xAjax-функция `Add_Marker()` находится в файле add_marker.php, содержимое которого представлено в листинге 4.64.

Листинг 4.64

```
<?php
function Add_Edit_Marker($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  // заменить переводы строки на <br>
  $Id[descriptionpoint]=str_replace("\r","<br>",$Id[descriptionpoint]);
  $Id[descriptionpoint]=str_replace("\n","<br>",$Id[descriptionpoint]);
  $Id[descriptionpoint]=str_replace("<br><br>","<br>",
     $Id[descriptionpoint]);
```

```
  // изменить информацию в базе данных
  $query1="SELECT * FROM style_metka WHERE name='".$Id[imageMarker1]."' ";
  $rez1=mysql_query($query1);
  $id_metka=mysql_result($rez1,0);
  $query2="UPDATE info_metka SET id_metka='".$id_metka."',
          name='".$Id[namepoint]."',
          description='".$Id[descriptionpoint]."',
          img='".$Id[marker_img]."' WHERE id=".$Id[id1]." ";
  $rez2=mysql_query($query2);
  if(!$rez2)
  { $objResponse->alert("Ошибка изменения метки!!!");
    return $objResponse;
  }
  $objResponse->alert("Метка изменена в базу!!!");
  // формируем новый контент
  // если есть картинка
  if($Id[marker_img]<>"")
  { $Id[descriptionpoint]="<br><img
     src=\"resize_200.php?pic=".$Id[marker_img]."\">
    <br>".$Id[descriptionpoint];
  }
  // название, информация
  $text="<b>".$Id[namepoint]."</b><br>".$Id[descriptionpoint]."<br>
    <a href=\"javascript:void();\"
    onclick=\"xajax_Edit_Marker(".$Id[id1].");\">
    Редактировать</a><br><a href=\"javascript:void();\"
    onclick=\"xajax_Delete_Marker(".$Id[id1].");\">Удалить</a>";
  // js-код для занесения контента в открытый балун
  $script="map.getBalloon().setContent('".$text."');";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

4.4.6. Изменение местоположения метки

Для изменения положения метки разрешим перетаскивание метки. Для этого при создании метки для каждой метки будем устанавливать опцию `draggable` в значение `true`. При перетаскивании метка последовательно генерирует три события: `DragStart`, `Drag` и `DragEnd`. Нам необходимо написать обработчик события `DragEnd`, чтобы получать измененные значения координат для метки. Код для формирования меток при загрузке страницы представлен в листинге 4.65.

Листинг 4.65

```
//Вывести все метки из базы
<?php
  $text01="";
```

```
  // получить неудаленные метки из базы
  $query01="SELECT * FROM info_metka WHERE visible='yes'
           ORDER BY id ASC LIMIT 0,20 ";
  $rez01=mysql_query($query01);
  while($row01=mysql_fetch_assoc($rez01))
  { $query02="SELECT name FROM style_metka WHERE id=".$row01[id_metka]." ";
    $rez02=mysql_query($query02);
    $row02=mysql_fetch_assoc($rez02);
    // создание метки
    $text01.="placemark[".$row01[id]."] = new YMaps.Placemark(new
      YMaps.GeoPoint(".$row01[lng].",".$row01[lat]."),
      {draggable:true,hideIcon: false,style: '".$row02[name]."'});";
    // содержимое балуна
    $text01.="placemark[".$row01[id]."].name = '".$row01[name]."';";
    // если есть картинка
    if($row01[img]<>"")
    { $row01[description]="<img
      src=\"resize_200.php?pic=".$row01[img]."\">
      <br>".$row01[description];
    }
    // добавление ссылок Редактировать
    $row01[description].="<br><a href=\"javascript:void();\"
      onclick=\"xajax_Edit_Marker(".$row01[id].");\">Редактировать</a>";
    $row01[description].="<br><a href=\"javascript:void();\"
      onclick=\"xajax_Delete_Marker(".$row01[id].");\">Удалить</a>";
    // содержимое полное
    $text01.="placemark[".$row01[id]."].description =
      '".$row01[description]."';";
    // поместить на карту
    $text01.="map.addOverlay(placemark[".$row01[id]."]);";
    // обработчик события Events.DragEnd для метки
    $text01.="YMaps.Events.observe(placemark[".$row01[id]."],
       placemark[".$row01[id]."].Events.DragEnd, function (obj)
       {new_lng=obj.getCoordPoint().getLng();
        new_lat=obj.getCoordPoint().getLat();
        xajax_Change_Poz_Marker(".$row01[id].",
        new_lng,new_lat);});";
  }
  echo $text01;
?>
```

Обработчик события `Events.DragEnd` для метки вызывает xAjax-функцию `Change_Poz_Marker()`, в которую передаются новые координаты метки. Эта функция записывает в базу данных новые координаты метки и выводит alert-сообщение (рис. 4.36). Функция `Change_Poz_Marker()` находится в файле change_poz_marker.php, содержимое которого представлено в листинге 4.66.

Листинг 4.66

```
<?php
function Change_Poz_Marker($id,$lng,$lat)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  // изменить координаты в базе данных
  $query1="UPDATE info_metka SET lng='".$lng."',lat='".$lat."'
           WHERE id=".$id." ";
  $rez1=mysql_query($query1);
  if(!$rez1)
  { $objResponse->alert("Ошибка изменения положения метки !!!");
    return $objResponse;
  }
  //$objResponse->alert($id." ".$lng." ".$lat);
  $objResponse->alert("Изменение положения метки — успешно!!!");
  return $objResponse;
}
?>
```

Рис. 4.36. Изменение координат метки

4.4.7. Удаление метки

Нам может потребоваться и удаление метки. Для этого необходимо щелкнуть по ссылке **Удалить** в балуне (см. рис. 4.34). При этом вызывется xAjax-функция `Delete_Marker()`. Удаление метки из базы не происходит: мы устанавливаем значение поля `visible=no`, и метка на карту не выводится. Функция также удаляет метку

с карты, формируя и отправляя на выполнение js-код. При этом выводится информационное alert-сообщение (рис. 4.37). Функция `Delete_Marker()` расположена в файле delete_marker.php, содержимое которого представлено в листинге 4.67.

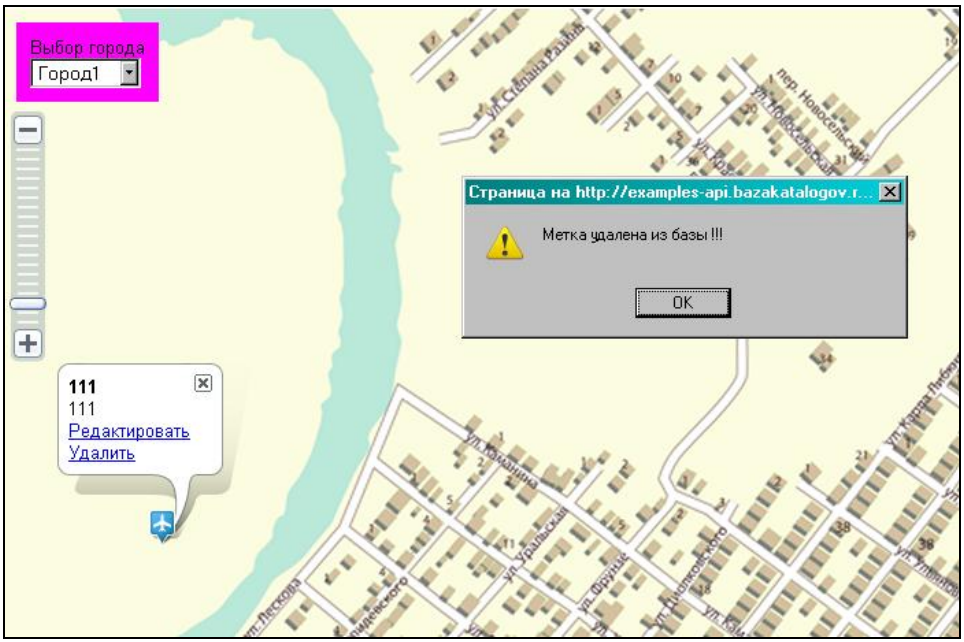


Рис. 4.37. Сообщение при удалении метки

Листинг 4.67

```
<?php
function Delete_Marker($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  // изменение в базе данных
  $query2="UPDATE info_metka SET visible='no' WHERE id=".$Id." ";
  $rez2=mysql_query($query2);
  if(!$rez2)
  { $objResponse->alert("Ошибка удаления метки!!!");
    return $objResponse;
  }
  $objResponse->alert("Метка удалена из базы!!!");
  // js-код удаления метки с карты
  $script="map.removeOverlay(placemark[".$Id."]);";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

Если вы хотите полностью удалять метку из базы, просто в коде измените запрос `update` на `delete`.

4.4.8. Загрузка на сервер файлов через форму без перезагрузки страницы

Для загрузки на сервер фото или роликов в формате SWF будем создавать в форме создания или редактирования метки фрейм с параметром `url=upload_img.php?id=1` или `url=upload_img.php?id=2`. Во фрейме расположена форма выбора файла. По событию `onchange input`-поля с `type=file` происходит загрузка файла на сервер, содержимое фрейма перезагружается, при загрузке новой страницы идет передача параметра (имя загруженного файла) в родительский фрейм, родительская страница при этом не перезагружается. Содержимое файла upload_img.php представлено в листинге 4.68.

Листинг 4.68

```
<?php
if ($_FILES["image"]["error"] == 0)
{ // получение параметров загруженного файла
  $ftmp = $_FILES['image']['tmp_name'];$oname = $_FILES['image']['name'];
  $fname = "imgtovar/".time()."";$pictype="";
  // определение типа загруженного файла
  switch(strtolower($_FILES['image']['type']))
  { case "image/jpeg" : $pictype=".jpg"; break;
    case "image/pjpeg": $pictype=".jpg"; break;
    case "image/gif"  : $pictype=".gif"; break;
    case "image/png"  : $pictype=".png"; break;
    case "image/x-png": $pictype=".png"; break;
    case "application/x-shockwave-flash": $pictype=".swf"; break;
    default           : $pictype="";     break;
  }
  if($pictype=="") // неверный тип файла
  {
  ... ... ...
  }
  else // верный тип файла
  { $fname = "img/".time().$pictype;
    // копирование временного файла в нужный файл
    move_uploaded_file($ftmp, $fname);
    chmod($fname,0777);
    ?>
    <html><head><script>
    var par = window.parent.document;
    var img1 = par.getElementById('img1');
    img1.src = 'resize_200.php?pic=<? echo $fname; ?>';
```

```
    par.forms.formadd_point.elements.marker_img.value=
          "<? echo $fname;   ?>";
    par.getElementById('rez_img').innerHTML="<font
       color='blue'>Загружено</font>";
    <?php
    }
  }
else  // ошибка загрузки файла
  ... ... ...
?>
<html><head>
<script language="javascript">function upload_img(){
var par = window.parent.document; var img1 = par.getElementById('img1');
// при начале загрузки – картинку загрузки
img1.src = 'img/zagruzka.gif';
document.iform.submit();
}
</script>
<style>
iframe { border-width: 0px; height: avto; width: 200px; }
iframe.hidden { visibility: hidden; width:0px; height:0px; }
#file {width: 150px;}
</style>
</head>
<body><center>
  <form name="iform" action="" method="post" enctype="multipart/form-data">
    Фото  *.jpg,*.png,*.gif<br>Или ролик *.swg<br>
    <input id="file" type="file" name="image" onchange="upload_img()" />
  </form>
</center></body>
</html>
```

4.4.9. Форма поиска меток

Для удобства поиска меток создадим форму. В ней находятся следующие элементы управления:

- `select` — поле, позволяющее сразу сделать множественный выбор объектов;
- `input` — поле для поиска объекта по наименованию;
- `checkbox` — для определения области поиска по наименованию;
- `input` — скрытое поле, номер страницы по поиску.

Код формирования формы представлен в листинге 4.69.

Листинг 4.69

```
<div id="formrez" style="margin-left: 700px; width:300px">
  <div id='zagformsearch'>Фильтр поиска</div>
```

```
  <div id='formsearch'>
  <form id='FormSearch' action='javascript:void(null);' onsubmit='
               xajax.$("ButtonFormSearch").disabled=true;
          xajax.$("ButtonFormSearch").value="Подождите...";
               xajax_View_Search_Markers(xajax.getFormValues("FormSearch"));'
  <?php
   // select типа объекта для формы поиска
   $text21="<select name=imageMarkerSearch id=imageMarkerSearch multiple
    size=4 onchange='xajax_Select_Marker_Search(
    xajax.getFormValues(\"FormSearch\"))'>
    <option value='' selected><Все>";
   $text22="</select><div id='imageSelect'></div>";
   echo $text21.$text1.$text22;
  ?>
  <br>Название
    <input type='text' name='name' value='' size=30 maxlength=30>
    <input type='hidden' id='pagesearch' name='pagesearch'
       value='1'></td></tr>
    <br><input type='checkbox' name='inname' value='1' checked> в
       названии
    <br><input type='checkbox' name='ininfo' value='1' > в описании  <br>
       <br>
  <input type='submit' id='ButtonFormSearch' value='Найти ->' disabled>
  </form>
  </div>
  <div id='zagformrez'></div>
  <div id='rez'></div>
  <div id='pagerez'></div>
  <div id='reklama'></div>
  <div id='contact'>Для добавления объекта -<br>
                    форма добавления по щелчку мышью <br>
                    на карте <br>
                    Для редактирования объекта -<br>
                    щелчок по значку объекта
  </div>
</div>
```

При формировании `select`-поля опции берутся из базы данных, из таблицы `style_metka`. Внизу поля `select` будем выводить значки выбираемых объектов (рис. 4.38). Добавление или удаление выбранных значков происходит при выполнении xAjax-функции `Select_Marker_Search()`, вызываемой по событию `onchange` для поля `select`. Функция `Select_Marker_Search()` расположена в файле select_marker_search.php, содержимое которого представлено в листинге 4.70.

Листинг 4.70

```
<?php
function Select_Marker_Search($Id)
{ $objResponse = new xajaxResponse();
  require_once("my.php");
  $img="";
  // ничего не выбрано
  if(count($Id[imageMarkerSearch])<1)
  { $objResponse->assign("ButtonFormSearch","disabled",true);
    return $objResponse;
  }
  $objResponse->assign("ButtonFormSearch","disabled",false);
  for($i=0;$i<count($Id[imageMarkerSearch]);$i++)
  { if($Id[imageMarkerSearch][$i]=='')
    { $img='';
      break; }
    else
    { // добавить картинку
      $img.="<img src=\"".DIR_IMG.urlencode(
         $Id[imageMarkerSearch][$i]).".png\" >";
    }
  }
  // вывести значки
  $objResponse->assign("imageSelect","innerHTML",$img);
  return $objResponse;
}
?>
```

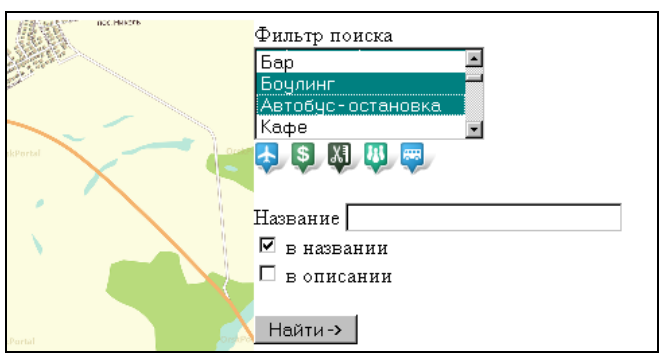


Рис. 4.38. Подгрузка выбранных значков

Для отправки данных формы на сервер используется xAjax-функция `View_Search_Markers()`, которая вызывается при нажатии кнопки **Найти**. Функция осуществляет следующие действия:

- поиск в базе данных объектов, удовлетворяющих условиям данного фильтра для текущей страницы (скрытое поле `pagesearch`);

- создание контента результата поиска;
- создание навигатора страниц для перехода между страницами результата поиска;
- удаление всех меток с карты;
- создание коллекции геообъектов, создание и добавление в коллекцию меток, удовлетворяющих условиям данного фильтра, вывод коллекции меток на карту.

Содержимое функции `View_Search_Markers()`, расположенной в файле view_search_markers.php, приведено в листинге 4.71.

Листинг 4.71

```
<?php
function View_Search_Markers($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");require_once("my.php");
  // не выбран тип метки
  if(count($Id[imageMarkerSearch])<1)
  { $objResponse->alert("Не выбран тип метки!!!");
    return $objResponse;
  }
  $text1="";$text2="";
  // формирование запроса
  $query0="SELECT * FROM info_metka WHERE visible='yes' ";
  $query01="&& id_metka IN (";
  $nn=count($Id[imageMarkerSearch])-1;
  for($i=0;$i<=$nn;$i++)
  {
    ... ... ...
  }
  $name=utftowin($Id[name]);
  if(strlen(rtrim(ltrim($name)))>0)
    { if($Id[inname]==1 && $Id[ininfo]==1)
        $query01.="&& (name LIKE '%".$name."%' || description LIKE
                '%".$name."%') ";
    elseif($Id[inname]==1)
      $query01.="&& name LIKE '%".$name."%' ";
    elseif($Id[ininfo]==1)
      $query01.="&& description LIKE '%".$name."%' ";
    else
      ;
    }
  $query0.=$query01;
  ... ... ...
  $count=mysql_num_rows($rez0);
  $pages=ceil($count/NN);
  $page=min($Id[pagesearch],$pages);
```

```
  $poz=($page-1)*NN;$num=$poz;
  if($count>0)
  { // удалить все метки
    $script="map.removeAllOverlays();";
    $objResponse->script($script);
    // создание таблицы с результатом поиска для страницы
    // шапка таблицы
    ... ... ...
    $query1=$query0." LIMIT ".$poz.", ".NN." ";
    $rez1=mysql_query($query1);
    ... ... ...
    // строки таблицы
    while($row1=mysql_fetch_assoc($rez1))
    { $num++;
    ... ... ...
    }
    // метки выводим для всех страниц поиска
    $query1=$query0."  ";
    $rez1=mysql_query($query1);
    while($row1=mysql_fetch_assoc($rez1))
    {
    ... ... ...
     // создание метки
    ... ... ...
     // добавление метки в коллекцию
      $script.="gCollection.add(placemark[".$row1[id]."]);";
    }
    // добавить коллекцию меток на карту
    $script.="map.addOverlay(gCollection);";
    $objResponse->script($script);
    // список ссылок перехода по страницам
    ... ... ...
  }
  else
  { $text1="<center>Поиск не дал результатов<br>
            Попробуйте изменить параметры поиска</center>";
    $text2="<table></table>";
  }
  //
  $objResponse->assign("ButtonFormSearch","value","Найти ->");
  $objResponse->assign("ButtonFormSearch","disabled",false);
  $objResponse->assign("zagformrez","innerHTML","Результаты поиска");
  $objResponse->assign("rez","innerHTML",$text1);
  $objResponse->assign("pagerez","innerHTML",$text2);
  return $objResponse;
}
?>
```

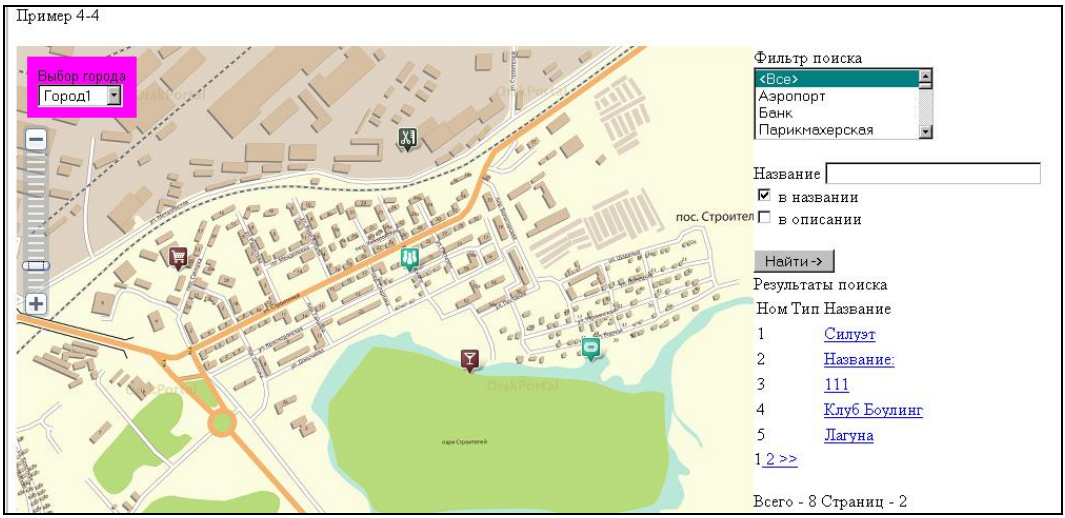


Рис. 4.39. Вывод результата поиска

Результаты поиска представляются в виде, как на рис. 4.39.

При щелчке по ссылке с названием объекта вызывается xAjax-функция `View_Search_Marker1()`, которая открывает балун для выбранного объекта и позиционирует центр карты в координатах выбранного объекта (рис. 4.40). Эта функция расположена в файле view_search_marker1.php, содержимое которого представлено в листинге 4.72.

Листинг 4.72

```
<?php
function View_Search_Marker1($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  require_once("my.php");
  // поиск метки в базе и получение ее параметров
  $query1="SELECT * FROM info_metka WHERE id=".$Id." ";
  $rez1=mysql_query($query1);
  $row1=mysql_fetch_assoc($rez1);
  $query2="SELECT name FROM style_metka WHERE id=".$row1[id_metka]." ";
  $rez2=mysql_query($query2);
  $row2=mysql_fetch_assoc($rez2);
  // формирование контента для балуна
  if($row1[img]<>"")
  { $row1[description]="<img src=\"resize_200.php?pic=".$row1[img]."\">
    <br>".$row1[description];
  }
  $row1[description].="<br><a href=\"javascript:void();\"
    onclick=\"xajax_Edit_Marker(".$row1[id].");\">Редактировать</a>";
```

```
  $row1[description].="<br><a href=\"javascript:void();\"
    onclick=\"xajax_Delete_Marker(".$row1[id].");\">Удалить</a>";
  $balun="<b>".$row1[name]."</b><br>".$row1[description];
  $script="map.openBalloon(new YMaps.GeoPoint(".$row1[lng].",
          ".$row1[lat]."), '".$balun."');";
  // открытие балуна в координатах выбранной метки
  $objResponse->script($script);
  // плавный перелет к метке
  $script="map.panTo(new YMaps.GeoPoint(".$row1[lng]."， ".$row1[lat]."));";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

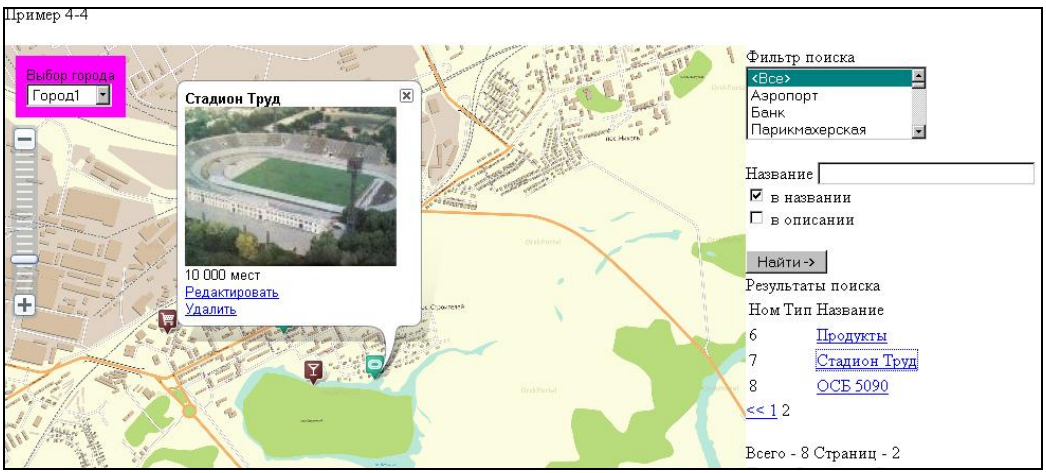


Рис. 4.40. Переход к выбранной метке

4.4.10. Варианты изменения скрипта

В клиентском приложении *(см. разд. 4.3)* мы не использовали форму поиска, которая может быть полезна для фильтрации объектов на карте, либо просто для поиска нужного объекта. Думаю, форму поиска с нужными процедурами вы сможете добавить самостоятельно. Как вариант, в качестве клиентской части можно использовать и администраторский скрипт. Для этого необходимо убрать возможность перетаскивания меток, формирование ссылок **Редактировать** и **Удалить** и, возможно, добавление пользователем меток на карту. Изменения минимальные, и, я думаю, вы сами с этим справитесь. Посмотреть данный вариант можно по ссылке **http://examples-api.bazakatalogov.ru/yandex/4-5/map1.php**, файлы данной реализации представлены на компакт-диске в папке glava_04\4-5.

ГЛАВА 5

ISPmanager API

Панель управления хостингом — это программное обеспечение для управления (администрирования) веб-сервером при помощи удобного браузерного интерфейса. Основная задача панели управления — упростить для пользователя работу со сложными функциями. Так, например, легко и без лишних операций можно создавать субдомены, устанавливать настройки баз данных и электронной почты, добавлять новые сайты и многое-многое другое. ISPmanager — продукт российских разработчиков "ISPsystem" — стал достаточно популярным в последнее время. Панель универсальна и имеет широкий список ПО и технологий, с которыми работает. Из преимуществ можно отметить: логичный интерфейс, рациональное использование ресурсов, качественная система разграничения прав использования функций панели управления.

5.1. ISPmanager API

ISPmanager предоставляет возможность использовать любые свои функции из внешних программ, которые могут располагаться как локально (на том же сервере), так и удаленно (на другом сервере). Для вызова какой-либо функции ISPmanager может потребоваться авторизация.

ISPmanager предоставляет возможность получения результата выполнения своих функций как в формате XML, так и в текстовом формате.

Для получения данных в виде XML необходимо передать панели управления в запросе дополнительный параметр `out=xml`. При этом результат будет зависеть от типа функции, к которой вы обращаетесь. Так, в случае получения списка элементов функция вернет XML-документ, состоящий из списка XML-узлов по одному на каждый элемент. Каждый узел в свою очередь состоит из набора узлов, определяющих параметры данного элемента. Например, при обращении к панели управления для получения списка WWW-доменов мы увидим примерно следующее:

```
<doc>
  <elem>
    <name>foo.org</name>
```

```
    <owner>user</owner>
    <ssl/>
    <php/>
    <ssi/>
  </elem>
  <elem>
    <name>mydomain.com</name>
    <owner>user2</owner>
    <php/>
    <cgi/>
  </elem>
  <elem>
    <name>rotate.com</name>
    <owner>alm</owner>
    <ssl/>
    <php/>
  </elem>
  <elem>
    <name>test.com</name>
    <owner>john</owner>
    <php/>
    <cgi/>
  </elem>
  <elem>
    <name>test.net</name>
    <owner>user</owner>
  </elem>
</doc>
```

При вызове функции, возвращающей набор параметров элемента, например, при его просмотре или редактировании, панель управления вернет XML-документ со списком узлов, соответствующих параметрам редактируемого элемента:

```
<doc>
  <elid>mydomain.com</elid>
  <domain>mydomain.com</domain>
  <alias>www.mydomain.com</alias>
  <ip>123.45.67.89</ip>
  <owner>user</owner>
  <admin>webmaster@mydomain.com</admin>
  <index>index.php index.htm</index>
  <php>phpcgi</php>
  <cgi/>
</doc>
```

Если же вы вызываете функцию, которая должна произвести какое-то действие, например удалить WWW-домен, панель управления вернет XML-документ об успешном выполнении операции:

```
<doc>
  <ok>restart</ok>
</doc>
```

или сообщение об ошибке

```
<doc>
  <error>abrakadabra not found.</error>
</doc>
```

Для получения данных в текстовом виде необходимо передать панели управления в запросе дополнительный параметр `out=text`. При этом результат также будет зависеть от типа функции, к которой вы обращаетесь. Так, в случае получения списка элементов функция вернет список строк, каждая из которых соответствует одному элементу и состоит из набора параметров этого элемента. Например, при обращении к панели управления для получения списка WWW-доменов мы увидим примерно следующее:

```
name=foo.org owner=user ssl php ssi
name=mydomain.com owner=user2 php cgi
name=rotate.com owner=alm ssl php
name=test.com owner=john php cgi
name=test.net owner=user
```

При вызове функции, возвращающей набор параметров элемента, например при его просмотре или редактировании, панель управления вернет список параметров элемента по одному на каждую строчку. Например, при просмотре свойств WWW-домена мы увидим:

```
elid=mydomain.com
domain=mydomain.com
alias=www.mydomain.com
ip=123.45.67.89
owner=user
php=phpcgi
admin=webmaster@mydomain.com
index=index.php index.htm
cgi
```

Если же вы вызываете функцию, которая должна произвести какое-то действие, например удалить WWW-домен, панель управления в случае успешного выполнения операции вернет

```
OK
```

или сообщение об ошибке

```
ERROR: abrakadabra not found.
```

5.1.1. Методы авторизации

ISPmanager предоставляет возможность использовать различные способы авторизации для использования панели управления как в интерактивном режиме (через

браузер), так и с помощью локальных либо удаленных вызовов из внешних программ или скриптов. На данный момент существуют четыре метода авторизации:

- авторизация с использованием уникального номера сессии;
- авторизация с использованием `authinfo`;
- авторизация с использованием доверенных IP-адресов;
- авторизация при локальном вызове функций ISPmanager.

Рассмотрим и возможность выполнения функций ISPmanager от имени другого пользователя, а также использование протоколов HTTP и HTTPS при работе с панелью управления.

Авторизация с использованием уникального номера сессии

Данный метод наиболее подходит для использования панели управления через браузер. Для того чтобы вызвать какую-либо функцию ISPmanager, нужно обратиться по следующему URL, используя соответствующие функции языков программирования или команды shell.

```
https://IP-адрес/manager/ispmgr?auth=номер_сессии&out=xml&func=
функция&параметр1=значение&параметр2=значение...
```

Авторизация происходит путем обращения по следующему URL:

```
https://IP-адрес/manager/ispmgr?out=xml&func=auth&username=
имя_пользователя&password=пароль
```

После этого ISPmanager вернет либо сообщение об ошибке, либо XML-документ следующего вида:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
<auth id="номер сессии"/>
</doc>
```

Далее вы должны будете передавать полученный номер сессии с каждым запросом к ISPmanager в параметре `auth`. Номер сессии хранится в течение 30 минут. По истечении этого срока вы должны будете заново пройти авторизацию.

Авторизация с использованием параметра *authinfo*

Данный метод авторизации удобен для удаленного обращения к панели управления. Для вызова какой-либо функции ISPmanager необходимо добавить дополнительный параметр `authinfo` и указать в нем имя пользователя и пароль, под которыми вы хотите выполнить операцию, например,

```
https://IP-адрес/manager/ispmgr?authinfo=admin1:mypasswd&out=xml&func=
функция&параметр1=значение&параметр2=значение...
```

Данный метод авторизации является разовым, т. е. вы должны посылать параметр `authinfo` с каждым запросом к панели управления.

Авторизация с использованием доверенных IP-адресов

Данный метод авторизации особенно удобен для удаленного обращения к панели управления, когда все обращения производятся с определенного IP-адреса. В качестве примера можно привести сервер, на котором располагается биллинговая система, периодически опрашивающая панель управления для получения информации об использовании трафика, дискового пространства, количества WWW-доменов и прочего. Данный метод позволяет отказаться от авторизации как таковой, указав в файле конфигурации панели управления /usr/local/ispmgr/etc/ispmgr.conf следующую строку:

```
TrustIP IP-адрес пользователь
```

В качестве параметра *IP-адрес* необходимо указать адрес сервера, с которого будут приходить запросы к панели управления. В качестве параметра *пользователь* — имя пользователя, с правами которого будут осуществляться операции с панелью управления. Данный параметр является необязательным. Если его нет, запросы будут обрабатываться с правами root.

Авторизация при локальном вызове функций ISPmanager

Данный метод авторизации является наиболее предпочтительным при локальном обращении к панели управления из внешних программ или скриптов. В качестве примера можно привести обращение к панели управления из своего скрипта, который выполняется с определенной периодичностью с помощью `cron`. В данном случае панель управления самостоятельно отслеживает UID процесса, который сделал запрос, и обрабатывает его с правами пользователя, который имеет этот UID. В данном случае никакой дополнительной авторизации не требуется.

HTTP или HTTPS?

Так как HTTP является незащищенным протоколом, передавать пароли или номера сессий с его помощью небезопасно. Поэтому панель управления отслеживает, совпадает ли IP-адрес вызывающей стороны с IP-адресом сервера, на котором она расположена, и, если совпадает, т. е. в случае локального вызова функций ISPmanager, разрешает использование протокола HTTP. В противном случае разрешается использование только HTTPS, и при попытке обращения к панели управления по протоколу HTTP будет возвращена ошибка.

Вызов функций ISPmanager с правами другого пользователя

Для того чтобы обратиться к какой-либо функции ISPmanager с правами другого пользователя, нужно добавить к URL дополнительный параметр `su=`*`имя_пользователя`*. Администратор сервера может вызывать функции с правами любых пользователей, реселлер — только с правами своих пользователей. Для всех остальных эта возможность запрещена.

5.1.2. Вызов функций ISPmanager из PHP

Язык PHP предоставляет возможность работы с URL как со стандартными файлами. Для обработки XML-документов используется библиотека DOM XML.

```
<?php
  $result = "";
  $fh = fopen("http://127.0.0.1/manager/ispmgr?out=xml&func=wwwdomain", "r");
  while($data = fread($fh, 4096))
  { $result .= $data; }
  fclose( $fh );
  // После этого переменная $result содержит XML-документ со списком
  // WWW-доменов либо сообщение об ошибке
  if($doc = domxml_open_mem($result))
  {  $root = $doc->document_element();
     for($elem = $root->first_child(); $elem;
         $elem = $elem->next_sibling())
     { for($node = $elem->first_child(); $node;
           $node = $node->next_sibling())
        { if( $node->node_name() == "name" )
          { echo $node->get_content();
            echo "\n";
          }
        }
     }
  }
?>
```

Далее приведем справочное описание всех функций API ISPmanager Описания параметров объекта или дополнительных параметров запроса представлены в соответствующих таблицах.

5.1.3. Администратор сервера

Функция: `mgradmin()`.

Результат — список элементов:

- `name` — имя;
- `disabled` — учетная запись временно отключена;
- `uid0` — администратор может использовать shell с правами root;
- `note` — заметки.

Параметры администратора, создание, изменение

Функция: `mgradmin.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `mgradmin()`).

Результат — список параметров объекта (табл. 5.1).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.1).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `mgradmin()`);
- дополнительные параметры запроса (табл. 5.1).

Таблица 5.1

Параметр	Описание
`name`	Логин
`passwd`	Пароль
`confirm`	Подтверждение
`uid0`	Доступ. Возможные значения: • `allow` — ко всем функциям; • `allow_partial` — к указанным функциям; • `deny_partial` — ко всем функциям, кроме указанных
`Fname`	Список функций (одно или несколько значений, разделенных пробелом)
`Email`	Адрес электронной почты
`Welcome`	Послать сообщение. Необязательный параметр. Чтобы включить данную опцию, используйте значение `"on"`

Удаление администраторов

Функция: `mgradmin.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `mgradmin()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Включение администратора

Функция: `mgradmin.enable()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `mgradmin()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Выключение администратора

Функция: `mgradmin.disable()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `mgradmin()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.4. Реселлер

Функция: `reseller()`.

Результат — список элементов:

- `name` — имя;
- `disabled` — реселлер отключен;
- `cgi` — реселлер может использовать CGI для своих WWW-доменов;
- `php` — реселлер может использовать PHP для своих WWW-доменов;
- `ssi` — реселлер может использовать SSI для своих WWW-доменов;
- `ssl` — реселлер может использовать безопасное соединение по протоколу HTTPs для своих WWW-доменов;
- `shell` — реселлер может разрешать своим пользователям использовать shell;
- `note` — заметки;
- `user` — пользователи. Атрибуты:
  - `used` — использованное количество;
  - `limit` — максимально возможное значение;
- `disk` — диск. Атрибуты:
  - `used` — использованное количество;
  - `limit` — максимально возможное значение;
- `bandwidth` — трафик. Атрибуты:
  - `used` — использованное количество;
  - `limit` — максимально возможное значение.

Создание, изменение, параметры реселлера

Функция: `reseller.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `reseller()`).

Результат — список параметров объекта (табл. 5.2).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.2).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `reseller()`);
- дополнительные параметры запроса (табл. 5.2).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.2

Параметр	Описание
`name`	Имя реселлера
`passwd`	Пароль
`confirm`	Подтверждение
`ip`	IP-адрес
`preset`	Шаблон
`email`	Адрес электронной почты
`welcome`	Послать сообщение*
`shell`	Доступ к shell*
`ssl`	SSL*
`phpmod`	PHP как модуль Apache*
`phpcgi`	PHP как CGI*
`cgi`	CGI*
`ssi`	SSI*

Таблица 5.2 *(окончание)*

Параметр	Описание
disklimit	Диск (числовое значение, для указания "бесконечности" используйте "1000000")
userlimit	Пользователи**
ftplimit	FTP-аккаунты**
maillimit	Почтовые ящики**
domainlimit	Домены**
webdomainlimit	WWW-домены**
maildomainlimit	WWW-домены**
baselimit	Базы данных**
baseuserlimit	Пользователи баз данных**
bandwidthlimit	Трафик (числовое значение, для указания "бесконечности" используйте "100000000")
cpulimit	Ограничение на CPU***
memlimit	Ограничение на память***
proclimit	Количество процессов***
note	Примечание

* Необязательный параметр. Чтобы включить данную опцию, используйте значение "on".

** Числовое значение. Для указания "бесконечности" используйте "10000".

*** Числовое значение. Для указания "бесконечности" используйте "0".

Удаление реселлеров

Функция: `reseller.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `reseller()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Включение реселлера и его пользователей

Функция: `reseller.enable()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `reseller()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Отключение реселлера и его пользователей

Функция: `reseller.disable()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `reseller()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Доступ к функциям

Функция: `reseller.access()`.

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `reseller()`).

Результат — список элементов:

- `ftitle` — название модуля;
- `name` — имя функции;
- `disabled` — реселлер не имеет доступа к данной функции.

Разрешение доступа к выбранным функциям

Функция: `reseller.access.enable()`.

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `reseller()`);
- `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `reseller.access()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Запрещение доступа к выбранным функциям

Функция: `reseller.access.disable()`.

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `reseller()`);
- `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `reseller.access()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Сообщение в центр поддержки

Функция: `new.msg()`.

Параметры:

- `subj` — тема;
- `prio` — приоритет;

- `category` — категория;
- `text` — текст.

5.1.5. Пользователь

Функция: `user()`.

Результат — список элементов:

- `name` — имя;
- `slave` — сервер
- `owner` — владелец;
- `preset` — шаблон;
- `disabled` — учетная запись и ее WWW-домены временно отключены;
- `cgi` — пользователь может использовать CGI для своих WWW-доменов;
- `php` — пользователь может использовать PHP для своих WWW-доменов;
- `ssi` — пользователь может использовать SSI для своих WWW-доменов;
- `ssl` — пользователь может использовать безопасное соединение по протоколу HTTPs для своих WWW-доменов;
- `shell` — пользователь может использовать shell;
- `note` — заметки (доступно только в ISPmanager Pro);
- `disk` — диск. Атрибуты:
  - `used` — использованное количество;
  - `limit` — максимально возможное значение;
- `bandwidth` — трафик. Атрибуты:
  - `used` — использованное количество;
  - `limit` — максимально возможное значение.

Создание, изменение, параметры пользователя

Функция: `user.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `user()`).

Результат — список параметров объекта (табл. 5.3).

Таблица 5.3

Параметр	Описание
name	Имя пользователя
passwd	Пароль
confirm	Подтверждение
domain	Домен
ip	IP-адрес
owner	Владелец
preset	Шаблон
email	Адрес электронной почты
welcome	Послать сообщение*
shell	Доступ к shell*
ssl	SSL*
cgi	SGI*
ssi	SSI*
phpmod	PHP как модуль Apache*
phpcgi	PHP как CGI*
phpfcgi	PHP как FastCGI*
safemode	Безопасный режим
disklimit	Диск***
ftplimit	FTP аккаунты**
maillimit	Почтовые ящики**
domainlimit	Домены**
webdomainlimit	WWW-домены**
maildomainlimit	Почтовые домены**
baselimit	Базы данных**
baseuserlimit	Пользователи баз данных**
bandwidthlimit	Трафик (числовое значение, для указания "бесконечности" используйте "100000000")
cpulimit	Ограничение на CPU***
memlimit	Ограничение на память***
proclimit	Количество процессов***
note	Примечание

* Необязательный параметр. Чтобы включить данную опцию, используйте значение "on".

** Числовое значение. Для указания "бесконечности" используйте "10000".

*** Числовое значение. Для указания "бесконечности" используйте "0".

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.3).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `user()`);
- дополнительные параметры запроса (табл. 5.3).

Результат — успешное выполнение операции или сообщение об ошибке.

Удаление пользователей

Функция: `user.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `user()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Включение пользователя и всех его WWW-доменов

Функция: `user.enable()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `user()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Отключение пользователя и всех его WWW-доменов

Функция: `user.disable()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `user()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Доступ к функциям

Функция: `user.access()`.

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `user()`).

Результат — список элементов:

- `name` — имя функции;
- `ftitle` — название модуля;
- `disabled` — пользователь не имеет доступа к данной функции.

Разрешение доступа к выбранным функциям

Функция: `user.access.enable()`.

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `user()`);
- `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `user.access()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Запрещение доступа к выбранным функциям

Функция: `user.access.disable()`.

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `user()`);
- `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `user.access()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.6. Почтовые ящики

Функция: `email()`.

Результат — список элементов:

- `name` — имя;
- `pop3` — логин POP3;
- `disabled` — почтовый ящик временно отключен;
- `note` — примечание;
- `size` — размер:
  - `used` — использованное количество;
  - `limit` — максимально возможное значение.

Создание, изменение, почтовый ящик

Функция: `email.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `email()`).

Результат — список параметров объекта (табл. 5.4).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса.

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `email()`);
- дополнительные параметры запроса (табл. 5.4).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.4

Параметр	Описание
`name`	Имя
`domain`	Домен
`aliases`	Псевдонимы (одно или несколько значений, разделенных пробелом)
`passwd`	Пароль
`confirm`	Подтверждение
`quota`	Максимальный размер (числовое значение, для указания "бесконечности" используйте "0")
`forward`	Отправлять копии писем на e-mail (одно или несколько значений, разделенных пробелом)
`rmlocal`	Не сохранять в ящик*
`greylist`	Включить "серый" список*
`spamassassin`	Включить SpamAssassin*
`note`	Примечание

* Необязательный параметр. Чтобы включить данную опцию, используйте значение "on".

Автоответчик (vacation), просмотр, изменение

Функция: `email.vacation()`.

Данная функция одновременно используется для просмотра и изменения параметров объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `email()`).

Результат — список параметров (табл. 5.5).

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно "yes";
- `elid` — уникальный идентификатор (элемент `name` из функции `email()`);
- дополнительные параметры запроса (табл. 5.5).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.5

Параметр	Описание	Параметр	Описание
`Arfrom`	От	`Arbody`	Сообщение
`Arsubj`	Тема		

Сортировка почты

Функция: `email.filter()`.

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `email()`).

Результат — список элементов, `name` — название почтового фильтра.

Создание, изменение, почтовый фильтр

Функция: `email.filter.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `email()`);
- `elid` — уникальный идентификатор (элемент `name` из функции `email.filter()`).

Результат — список параметров объекта (табл. 5.6).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `email()`);
- дополнительные параметры запроса (табл. 5.6).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `email()`);
- `elid` — уникальный идентификатор (элемент `name` из функции `email.filter()`);
- дополнительные параметры запроса (табл. 5.6).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.6

Параметр	Описание
`name`	Название
`pos`	Расположить перед
`cond_param`	Параметр условия
`cond`	Условие
`cond_value`	Значение
`act`	Действие
`act_email`	Адрес электронной почты
`act_imap`	Папка IMAP
`act_imap_new`	Новая папка IMAP
`act_command`	Программа
`copy`	Работать с копией. Необязательный параметр. Чтобы включить данную опцию, используйте значение `"on"`

Удаление почтового фильтра

Функция: `email.filter.delete()`.

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `email()`);

- elid — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент name из функции email.filter().

Результат — успешное выполнение операции или сообщение об ошибке.

Условия почтового фильтра

Функция: email.filter.cond().

Параметр: elid — уникальный идентификатор (элемент name из функции email.filter()).

Результат — список элементов:

- id — уникальный идентификатор элемента списка;
- cond_param — параметр;
- cond — условие;
- cond_value — значение.

Создание, изменение, условие

Функция: email.filter.cond.edit().

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметры:

- plid — уникальный идентификатор родительского списка (элемент name из функции email.filter());
- elid — уникальный идентификатор (элемент id из функции email.filter.cond()).

Результат — список параметров объекта (табл. 5.7).

Создание объекта

Параметры:

- sok — значение параметра должно быть равно "yes";
- plid — уникальный идентификатор родительского списка (элемент name из функции email.filter());
- дополнительные параметры запроса (табл. 5.7).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- sok — значение параметра должно быть равно "yes";
- plid — уникальный идентификатор родительского списка (элемент name из функции email.filter());

- elid — уникальный идентификатор (элемент id из функции email.filter.cond());
- дополнительные параметры запроса (табл. 5.7).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.7

Параметр	Описание	Параметр	Описание
cond_param	Параметр	cond_value	Значение
cond	Условие		

Удаление условия

Функция: email.filter.cond.delete().

Параметры:

- plid — уникальный идентификатор родительского списка (элемент name из функции email.filter());
- elid — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент id из функции email.filter.cond().

Результат — успешное выполнение операции или сообщение об ошибке.

Удаление почтовых ящиков

Функция: email.delete().

Параметр: elid — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент name из функции email().

Результат — успешное выполнение операции или сообщение об ошибке.

Очистка почтовых ящиков

Функция: email.clear().

Параметр: elid — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент name из функции email().

Результат — успешное выполнение операции или сообщение об ошибке.

Включение почтовых ящиков

Функция: email.enable().

Параметр: elid — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент name из функции email().

Результат — успешное выполнение операции или сообщение об ошибке.

Отключение почтовых ящиков

Функция: `email.disable()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `email()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.7. WWW-домены

Функция: `wwwdomain()`.

Результат — список элементов:

- `name` — имя;
- `ip` — IP-адрес;
- `owner` — владелец;
- `docroot` — каталог;
- `cgi` — данный WWW-домен может использовать CGI;
- `php` — данный WWW-домен может использовать PHP;
- `ssi` — данный WWW-домен может использовать SSI;
- `frp` — данный WWW-домен может использовать расширение FrontPage;
- `ssl` — для доступа к данному WWW-домену может быть использован протокол HTTPS.

Создание, изменение, параметры WWW-домена

Функция: `wwwdomain.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `wwwdomain()`).

Результат — список параметров объекта (табл. 5.8).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.8).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `wwwdomain()`);
- дополнительные параметры запроса (табл. 5.8).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.8

Параметр	Описание
`domain`	Доменное имя
`alias`	Псевдонимы (одно или несколько значений, разделенных пробелом)
`docroot`	Корневая папка
`owner`	Владелец
`ip`	IP-адрес
`admin`	Адрес электронной почты администратора
`charset`	Кодировка
`index`	Индексная страница (одно или несколько значений, разделенных пробелом)
`utosubdomain`	Автоподдомены. Возможные значения: • `asdnone` — отключены; • `asddir` — в отдельном каталоге; • `asdsubdir` — в подкаталоге WWW-домена
`php`	PHP. Возможные значения: • `phpnone` — нет поддержки PHP; • `phpmod` — PHP как модуль Apache; • `phpcgi` — PHP как CGI; • `phpfcgi` — PHP как FastCGI
`cgi`	Cgi-bin*
`ssi`	SSI*
`ssiext`	Расширения файлов SSI
`frp`	FrontPage*
`fppasswd`	Пароль для FrontPage
`ror`	Ruby on rails*
`ssl`	SSL*
`sslport`	SSL-порт

* Необязательный параметр. Чтобы включить данную опцию, используйте значение "on".

Удаление WWW-доменов

Функция: `wwwdomain.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `wwwdomain()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Ротация логов, просмотр, изменение

Функция: `wwwdomain.log()`.

Одновременно используется для просмотра и изменения параметров объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `wwwdomain()`).

Результат — список параметров (табл. 5.9).

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `wwwdomain()`);
- дополнительные параметры запроса (табл. 5.9).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.9

Параметр	Описание
`rotper`	Период ротации
`arcnum`	Количество архивов, которые надо хранить
`errlog`	Журнал ошибок
`analyzer`	Анализатор
`analper`	Период анализа
`fixall`	Применить для всех WWW-доменов. Необязательный параметр. Чтобы включить данную опцию, используйте значение `"on"`

5.1.8. Почтовые домены

Функция: `emaildomain()`.

Результат — список элементов:

- `name` — доменное имя;
- `default` — действие по умолчанию;
- `owner` — владелец.

Создание, изменение, настройки почтового домена

Функция: `emaildomain.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `emaildomain()`).

Результат — список параметров объекта (табл. 5.10).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.10).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `emaildomain()`);
- дополнительные параметры запроса (табл. 5.10).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.10

Параметр	Описание
`name`	Доменное имя
`dtype`	Действие по умолчанию. Возможные значения: • `nouser` — сообщение об ошибке; • `toemail` — перенаправить на адрес; • `todomain` — перенаправить на домен
`domain`	Перенаправить на домен
`mbox`	Перенаправить на почтовый адрес
`owner`	Владелец
`greylist`	Включить "серый" список*
`spamassassin`	Включить SpamAssassin*

* Необязательный параметр. Чтобы включить данную опцию, используйте значение "on".

Удаление почтового домена

Функция: `emaildomain.delete.confirm()`.

Параметр: `dns` — сохранить записи в DNS. Необязательный параметр. Чтобы включить данную опцию, используйте значение `"on"`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.9. Доменные имена (DNS)

Функция: `domain()`.

Результат — список элементов:

- `name` — доменное имя;
- `owner` — владелец.

Создание, изменение, параметры домена

Функция: `domain.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `domain()`).

Результат — список параметров объекта (табл. 5.11).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.11).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.11

Параметр	Описание
`name`	Доменное имя
`ip`	IP-адрес
`ns`	Серверы имен (одно или несколько значений, разделенных пробелом)
`mx`	Почтовые серверы (одно или несколько значений, разделенных пробелом)
`owner`	Владелец
`webdomain`	Необязательный параметр. Чтобы включить данную опцию, используйте значение `"on"`

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `domain()`);
- дополнительные параметры запроса (табл. 5.11).

Результат — успешное выполнение операции или сообщение об ошибке.

Управление записями

Функция: `domain.sublist()`.

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `domain()`).

Результат — список элементов:

- `key` — уникальный идентификатор элемента списка;
- `name` — имя;
- `type` — тип;
- `addr` — адрес.

Создание, изменение, параметры записи

Функция: `domain.sublist.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `domain()`);
- `elid` — уникальный идентификатор (элемент `key` из функции `domain.sublist()`).

Результат — список параметров объекта (табл. 5.12).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `domain()`);
- дополнительные параметры запроса (табл. 5.12).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `domain()`);
- `elid` — уникальный идентификатор (элемент `key` из функции `domain.sublist()`);
- дополнительные параметры запроса (табл. 5.12).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.12

Параметр	Описание
`name`	Имя
`sdtype`	Тип. Возможные значения: • `A` — интернет-адрес; • `CNAME` — каноническое имя; • `NS` — сервер имен; • `MX` — почтовый сервер; • `TXT` — текстовая запись
`addr`	Адрес
`prio`	Приоритет

Удаление записей

Функция: `domain.sublist.delete()`.

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `domain()`);
- `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `key` из функции `domain.sublist()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Подтверждение удаления домена

Функция: `domain.delete.confirm()`.

Параметры:

- `webdomain` — удалить WWW-домен (необязательный параметр; чтобы включить данную опцию, используйте значение `"on"`);

- `maildomain` — удалить почтовый домен (необязательный параметр; чтобы включить данную опцию, используйте значение `"on"`).

Результат — успешное выполнение операции или сообщение об ошибке.

Обновление домена на внешнем сервере имен

Функция: `domain.fix()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `domain()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Настройки доменов по умолчанию, просмотр, изменение

Функция: `domain.prop()`.

Данная функция одновременно используется для просмотра и изменения данных формы.

Просмотр данных

Результат — список параметров (табл. 5.13).

Изменение данных

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.13).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.13

Параметр	Описание
`ns`	Серверы имен (одно или несколько значений, разделенных пробелом)
`mx`	Почтовые серверы (одно или несколько значений, разделенных пробелом)
`email`	Адрес службы технической поддержки
`subdom`	Дополнительные записи (одно или несколько значений, разделенных пробелом)
`fixall`	Изменить для всех доменов. Необязательный параметр. Чтобы включить данную опцию, используйте значение `"on"`

Внешние серверы имен

Функция: `domain.slave()`.

Результат — список элементов:

- `addr` — IP-адрес;
- `type` — тип.

Создание, изменение, вторичный сервер имен

Функция: `domain.slave.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `addr` из функции `domain.slave()`).

Результат — список параметров объекта (табл. 5.14).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.14).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `addr` из функции `domain.slave()`);
- дополнительные параметры запроса (табл. 5.14).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.14

Параметр	Описание
`addr`	IP-адрес
`stype`	Тип. Возможные значения: • `ispmgr` — ISPmanager; • `dnsmgr` — DNSmanager
`user`	Пользователь
`passwd`	Пароль

Удаление внешних серверов имен

Функция: `domain.slave.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `addr` из функции `domain.slave()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.10. Базы данных

Функция: `db()`.

Результат — список элементов:

- `dbkey` — уникальный идентификатор элемента списка;
- `name` — имя базы;
- `dbtype` — тип базы данных;
- `dbuser` — пользователь;
- `owner` — владелец;
- `size` — размер (в мегабайтах).

Создание, изменение, параметры базы данных

Функция: `db.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `dbkey` из функции `db()`).

Результат — список параметров объекта (табл. 5.15).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.15).

Результат — успешное выполнение операции или сообщение об ошибке (табл. 5.15).

Таблица 5.15

Параметр	Описание
`name`	Имя базы
`dbtype`	Тип базы данных
`owner`	Владелец
`dbencoding`	Кодировка
`dbuser`	Пользователь
`dbusername`	Новый пользователь
`dbpassword`	Пароль
`dbconfirm`	Подтверждение
`dbuserhost`	Удаленный доступ. Необязательный параметр. Чтобы включить данную опцию, используйте значение `"on"`

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `dbkey` из функции `db()`);
- дополнительные параметры запроса (табл. 5.15).

Результат — успешное выполнение операции или сообщение об ошибке.

Удаление выбранных баз

Функция: `db.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `dbkey` из функции `db()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Управление пользователями базы данных

Функция: `db.users()`.

Параметр: `elid` — уникальный идентификатор (элемент `dbkey` из функции `db()`).

Результат — список элементов:

- `name` — имя пользователя;
- `read` — пользователь может осуществлять операции чтения из базы;
- `write` — пользователь может осуществлять операции записи в базу;
- `manage` — пользователь может изменять структуру базы.

Создание, изменение, параметры пользователя

Функция: `db.users.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `dbkey` из функции `db()`);
- `elid` — уникальный идентификатор (элемент `name` из функции `db.users()`).

Результат — список параметров объекта (табл. 5.16).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;

- `plid` — уникальный идентификатор родительского списка (элемент `dbkey` из функции `db()`);
- дополнительные параметры запроса (табл. 5.16).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `plid` — уникальный идентификатор родительского списка (элемент `dbkey` из функции `db()`);
- `elid` — уникальный идентификатор (элемент `name` из функции `db.users()`);
- дополнительные параметры запроса (табл. 5.16).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.16

Параметр	Описание	Параметр	Описание
`dbuser`	Пользователь	`update_priv`	Другое*
`dbusername`	Имя пользователя	`create_priv`	Другое*
`dbpassword`	Пароль	`drop_priv`	Другое*
`dbconfirm`	Подтверждение	`alter_priv`	Другое*
`dbuserhost`	Удаленный доступ*	`index_priv`	Другое*
`select_priv`	Другое*	`grant_priv`	Другое*
`delete_priv`	Другое*	`references_priv`	Другое*
`insert_priv`	Другое*	`lock_priv`	Другое*

* Необязательный параметр. Чтобы включить данную опцию, используйте значение "on".

Удаление пользователей базы данных

Функция: `db.users.delete()`.

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `dbkey` из функции `db()`);
- `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `db.users()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Проверка выбранных баз

Функция: `db.repair()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `dbkey` из функции `db()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.11. Брандмауэр (firewall)

Функция: `firewall()`.

Результат — список элементов:

- `port` — уникальный идентификатор элемента списка;
- `name` — имя сервиса;
- `status` — статус.

Настройка фильтрации для порта, просмотр, изменение

Функция: `firewall.edit()`.

Данная функция одновременно используется для просмотра и изменения параметров объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `port` из функции `firewall()`).

Результат — список параметров (табл. 5.17).

Таблица 5.17

Параметр	Описание
`name`	Имя сервиса
`ftype`	Тип защиты. Возможные значения: • `open` — открытый; • `closed` — закрытый; • `open_partial` — частично открытый; • `closed_partial` — частично закрытый
`Openip`	Разрешить IP-адресам (одно или несколько значений, разделенных пробелом)
`Closeip`	Запретить IP-адресам (одно или несколько значений, разделенных пробелом)

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `port` из функции `firewall()`);
- дополнительные параметры запроса (табл. 5.17).

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.12. Сервисы

Функция: `services()`.

Результат — список элементов:

- `name` — название;
- `proc` — имя процесса;
- `count` — количество;
- `active` — сервис запущен;
- `autostart` — автоматический запуск сервиса после перезагрузки;
- `monitored` — мониторинг сервиса с помощью watchdog;
- `unavail` — не найден скрипт автозапуска сервиса.

Создание, изменение, настройка сервиса

Функция: `services.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `services()`).

Результат — список параметров объекта (табл. 5.18).

Таблица 5.18

Параметр	Описание	Параметр	Описание
`name`	Название	`confname`	Системное имя
`base`	Режим	`type`	Тип сервиса
`proc`	Имя процесса	`ip`	IP-адрес
`start`	Команда для запуска	`autostart`	Автозапуск*
`stop`	Команда для остановки	`monitor`	Мониторинг*

* Необязательный параметр. Чтобы включить данную опцию, используйте значение `"on"`.

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.18).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `services()`);
- дополнительные параметры запроса (табл. 5.18).

Результат — успешное выполнение операции или сообщение об ошибке.

Удаление сервиса

Функция: `services.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `services()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Остановка сервисов

Функция: `services.stop()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `services()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Запуск сервисов

Функция: `services.start()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `services()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Перезапуск сервисов

Функция: `services.restart()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `services()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Глобальные настройки сервисов, просмотр, изменение

Функция: `services.options()`.

Данная функция одновременно используется для просмотра и изменения данных формы.

Просмотр данных

Результат — список параметров (табл. 5.19).

Изменение данных

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.19).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.19

Параметр	Описание
`inetdconf`	Файл конфигурации
`inetdstart`	Команда запуска
`inetdstop`	Команда остановки
`inetdconfname`	Системное имя
`xinetdconf`	Файл конфигурации
`xinetdstart`	Команда запуска
`xinetdstop`	Команда остановки
`xinetdconfname`	Системное имя
`enflag`	Мониторинг. Необязательный параметр. Чтобы включить данную опцию, используйте значение `"on"`
`period`	Периодичность
`script`	Скрипт
`notify`	Уведомления
`email`	E-mail для уведомлений. Одно или несколько значений, разделенных пробелом
`from`	Обратный e-mail
`subject`	Заголовок
`mail`	Сообщение

5.1.13. Задания резервного копирования

Функция: `backupplan()`.

Результат — список элементов:

- `id` — идентификатор;
- `name` — название задания;
- `disabled` — задание отключено;
- `local` — локальное хранилище;
- `ftp` — хранилище на FTP-сервере;
- `sftp` — хранилище на сервере sFTP;
- `tgz` — формат архива TGZ;
- `tar` — формат архива TAR;
- `tbz2` — формат архива TBZ2;
- `zip` — формат архива ZIP;
- `rar` — формат архива RAR;
- `running` — в процессе;
- `empty` — нет данных для резервного копирования;
- `period` — период.

Создание, изменение, задание

Функция: `backupplan.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `id` из функции `backupplan()`).

Результат — список параметров объекта (табл. 5.20).

Таблица 5.20

Параметр	Описание	Параметр	Описание
`name`	Название задания	`ftpdir`	Каталог хранилища на сервере FTP
`storage`	Хранилище	`archiver`	Архиватор
`ldir`	Каталог хранилища на локальном сервере	`period`	Период
`ftphost`	Сервер	`starttime`	Время запуска
`ftpuser`	Логин	`limit`	Количество
`ftppasswd`	Пароль	`id`	Идентификатор

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.20).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `id` из функции `backupplan()`);
- дополнительные параметры запроса (табл. 5.20).

Результат — успешное выполнение операции или сообщение об ошибке.

Удаление задания

Функция: `backupplan.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `id` из функции `backupplan()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Включение задания

Функция: `backupplan.enable()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `id` из функции `backupplan()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Отключение задания

Функция: `backupplan.disable()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `id` из функции `backupplan()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Сделать резервную копию сейчас

Функция: `backupplan.dobackup()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `id` из функции "`backupplan()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Данные для резервного копирования

Функция: `backupplan.content()`.

Параметр: `elid` — уникальный идентификатор (элемент `id` из функции `backupplan()`).

Результат — список элементов:

- `name` — уникальный идентификатор элемента списка;
- `module` — тип данных;
- `data` — данные;
- `disabled` — данные будут исключены из архива.

Создание, изменение

Функция: `backupplan.content.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `id` из функции `backupplan()`);
- `elid` — уникальный идентификатор (элемент `name` из функции `backupplan.content()`).

Результат — список параметров объекта (табл. 5.21).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `plid` — уникальный идентификатор родительского списка (элемент `id` из функции `backupplan()`);
- дополнительные параметры запроса (табл. 5.21).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `plid` — уникальный идентификатор родительского списка (элемент `id` из функции `backupplan()`);
- `elid` — уникальный идентификатор (элемент `name` из функции `backupplan.content()`);
- дополнительные параметры запроса (табл. 5.21).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.21

Параметр	Описание	Параметр	Описание
type	Действие	postgresqldata	База данных postgresql
module	Тип данных	mgrdatadata	Настройки панели
filedata	Путь	emaildata	Почтовый ящик
mysqldata	База данных mysql	userdata	Пользователь

Удаление данных

Функция: `backupplan.content.delete()`.

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `id` из функции `backupplan()`);
- `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `backupplan.content()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.14. Перенос пользователя

Функция: `backup.import()`.

Данная функция одновременно используется для просмотра и изменения данных формы.

Просмотр данных

Результат — список параметров (табл. 5.22).

Изменение данных

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.22).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.22

Параметр	Описание	Параметр	Описание
user	Пользователь	login	Логин
host	Сервер	passwd	Пароль

5.1.15. Списки блокировки dnsbl

Функция: `dnsbl()`.

Результат — список элементов: `name` — списки блокировки.

Создание, изменение, просмотр параметров

Функция: `dnsbl.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `dnsbl()`).

Результат — список параметров объекта (табл. 5.23).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.23).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `dnsbl()`);
- дополнительные параметры запроса (табл. 5.23).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.23

Параметр	Описание
`name`	Список блокировки

Удаление списков блокировки dnsbl

Функция: `dnsbl.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `dnsbl()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.16. "Серый" список (greylisting)

Функция: `greylist()`.

Результат — список элементов:

- `key` — уникальный идентификатор элемента списка;
- `from` — отправитель;
- `rcpt` — получатель.

Создание, изменение, правило для "серого" списка

Функция: `greylist.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `key` из функции `greylist()`).

Результат — список параметров объекта (табл. 5.24).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.24).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `key` из функции `greylist()`);
- дополнительные параметры запроса (табл. 5.24).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.24

Параметр	Описание	Параметр	Описание
`tsender`	Тип отправителя	`trcpt`	Тип получателя
`sender`	Отправитель	`rcpt`	Получатель

Удаление правила серого списка

Функция: `greylist.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `key` из функции `greylist()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.17. "Белый" список

Функция: `whitelist()`.

Результат — список элементов:

- `sender` — отправитель;
- `action` — действие.

Создание, изменение, параметры записи

Функция: `whitelist.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `sender` из функции `whitelist()`).

Результат — список параметров объекта (табл. 5.25).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.25).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `sender` из функции `whitelist()`);
- дополнительные параметры запроса (табл. 5.25).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.25

Параметр	Описание	Параметр	Описание
sender	Отправитель	action	Действие

Удаление

Функция: `whitelist.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `sender` из функции `whitelist()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.18. "Черный" список

Функция: `blacklist()`.

Результат — список элементов:

- `sender` — отправитель;
- `action` — действие.

Создание, изменение, параметры записи

Функция: `blacklist.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `sender` из функции `blacklist()`).

Результат — список параметров объекта (табл. 5.26).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.26).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.26

Параметр	Описание	Параметр	Описание
`sender`	Отправитель	`errmsg`	Текст ошибки
`action`	Отклонить		

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;

- elid — уникальный идентификатор (элемент sender из функции blacklist());
- дополнительные параметры запроса (табл. 5.26).

Результат — успешное выполнение операции или сообщение об ошибке.

Удаление

Функция: blacklist.delete().

Параметр: elid — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент sender из функции blacklist().

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.19. Используемые ресурсы

Функция: usagestat().

Результат — список элементов:

- name — название;
- value — использовано/всего. Атрибуты:
  - used — использованное количество;
  - limit — максимально возможное значение.

5.1.20. Информация о системе

Функция: sysinfo().

Результат — список элементов:

- name — параметр;
- value — значение.

5.1.21. Параметры сервера

Функция: srvparam().

Данная функция одновременно используется для просмотра и изменения данных формы.

Просмотр данных

Результат — список параметров (табл. 5.27).

Изменение данных

Параметры:

- sok — значение параметра должно быть равно "yes";
- дополнительные параметры запроса (табл. 5.27).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.27

Параметр	Описание	Параметр	Описание
srvname	Имя сервера	rootfwd	Перенаправление почты root
timezone	Часовой пояс		

5.1.22. IP-адреса

Функция: `iplist()`.

Результат — список элементов:

- `name` — IP-адрес;
- `stat` — статус;
- `owner` — владелец;
- `count` — WWW-доменов.

Создание, изменение, параметры IP-адреса

Функция: `iplist.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `iplist()`).

Результат — список параметров объекта (табл. 5.28).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.28).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `iplist()`);
- дополнительные параметры запроса (табл. 5.28).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.28

Параметр	Описание
name	IP-адрес
stat	Статус. Возможные значения: • free — свободный; • shared — общедоступный; • assigned — назначенный
owner	Владелец

Удаление IP-адреса

Функция: iplist.delete().

Параметр: elid — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент name из функции iplist().

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.23. Настройки PHP

Функция: phpconf().

Данная функция одновременно используется для просмотра и изменения данных формы.

Просмотр данных

Результат — список параметров (табл. 5.29).

Изменение данных

Параметры:

- sok — значение параметра должно быть равно "yes";
- дополнительные параметры запроса (табл. 5.29).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.29

Параметр	Описание
exectime	Время выполнения
memlimit	Лимит памяти
uploadsize	Максимальный размер файлов

Таблица 5.29 *(окончание)*

Параметр	Описание
regglobal	Использование глобальных переменных (register_globals): • on — разрешены; • off — запрещены
displayerror	Вывод сообщений об ошибках php-скриптов (display_errors): • on — выводить; • off — скрывать от пользователя
logerror	Запись в log-файл сообщений об ошибках php-скриптов (log_errors): • on — записывать; • off — не записывать

* Необязательный параметр. Чтобы включить данную опцию, используйте значение "on".

5.1.24. Расширения PHP

Функция: phpextensions().

Результат — список элементов:

- name — имя;
- enabled — расширение PHP включено;
- builtin — расширение PHP заблокировано.

Включение выбранных расширений PHP

Функция: phpextensions.enable().

Параметр: elid — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент name из функции phpextensions().

Результат — успешное выполнение операции или сообщение об ошибке.

Отключение выбранных расширений PHP

Функция: phpextensions.disable().

Параметр: elid — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент name из функции phpextensions().

Результат — успешное выполнение операции или сообщение об ошибке.

Установка других расширений PHP, просмотр, изменение

Функция: `phpextensions.edit()`.

Данная функция одновременно используется для просмотра и изменения данных формы.

Просмотр данных

Результат — список параметров (табл. 5.30).

Изменение данных

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`.
- дополнительные параметры запроса (табл. 5.30).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.30

Параметр	Описание
`name`	Название

5.1.25. Модули Perl

Функция: `perlmodules()`.

Результат — список элементов:

- `name` — название;
- `installing` — осуществляется действие.

Добавление модуля Perl, просмотр, изменение

Функция: `perlmodules.edit()`.

Функция одновременно используется для просмотра и изменения данных формы.

Просмотр данных

Результат — список параметров (табл. 5.31).

Изменение данных

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.31).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.31

Параметр	Описание
name	Название

5.1.26. Возможности

Функция: `feature()`.

Результат — список элементов:

- `name` — название;
- `version` — установленная версия;
- `active` — установлена и включена;
- `installing` — осуществляется действие.

Просмотр, изменение

Функция: `feature.edit()`.

Одновременно используется для просмотра и изменения параметров объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `feature()`).

Результат — список параметров (табл. 5.32).

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `feature()`);
- дополнительные параметры запроса (табл. 5.32).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.32

Параметр	Описание
version	Версия

Удаление

Функция: `feature.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `feature()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Включение

Функция: `feature.enable()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `feature()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Выключение

Функция: `feature.disable()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `feature()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.27. Шаблоны пользователей

Функция: `preset()`.

Результат — список элементов:

- `name` — название;
- `type` — тип;
- `usecount` — использован раз;
- `disk` — диск;
- `domain` — домены;
- `database` — базы данных;
- `bandwidth` — трафик;
- `cgi` — пользователь (реселлер) может использовать CGI для своих WWW-доменов;
- `php` — пользователь (реселлер) может использовать PHP для своих WWW-доменов;
- `ssi` — пользователь (реселлер) может использовать SSI для своих WWW-доменов;
- `ssl` — пользователь (реселлер) может использовать безопасное соединение по протоколу HTTPs для своих WWW-доменов;
- `shell` — пользователь (реселлер) может использовать shell.

Создание, изменение, параметры шаблона

Функция: `preset.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `preset()`).

Результат — список параметров объекта (табл. 5.33).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.33).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.33

Параметр	Описание
`name`	Название
`ptype`	Тип. Возможные значения: `user`; `reseller`
`shell`	Доступ к shell*
`ssl`	SSL*
`cgi`	CGI*
`ssi`	SSI*
`phpmod`	PHP как модуль Apache*
`phpcgi`	PHP как CGI*
`phpfcgi`	PHP как FastCGI*
`safemode`	PHP в режиме safe_mode*
`userlimit`	Пользователи**
`disklimit`	Диск***
`ftplimit`	FTP аккаунты**
`maillimit`	Почтовые ящики**
`domainlimit`	Домены**
`webdomainlimit`	WWW-домены**
`maildomainlimit`	Почтовые домены**
`baselimit`	Базы данных**
`baseuserlimit`	Пользователи баз данных**
`bandwidthlimit`	Трафик (числовое значение, для указания "бесконечности" используйте "100000000")
`cpulimit`	Ограничение на CPU***
`memlimit`	Ограничение на память***
`proclimit`	Количество процессов***

* Необязательный параметр. Чтобы включить данную опцию, используйте значение "on".

** Числовое значение. Для указания "бесконечности" используйте "10000".

*** Числовое значение. Для указания "бесконечности" используйте "0".

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `preset()`);
- дополнительные параметры запроса (табл. 5.33).

Результат — успешное выполнение операции или сообщение об ошибке.

Удаление шаблонов

Функция: `preset.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `preset()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Доступ к функциям

Функция: `preset.access()`.

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `preset()`).

Результат — список элементов:

- `ftitle` — название модуля;
- `name` — имя функции;
- `disabled` — пользователь шаблона не имеет доступа к данной функции.

Разрешение доступа к выбранным функциям

Функция: `preset.access.enable()`.

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `preset()`);
- `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `preset.access()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Запрет доступа к выбранным функциям

Функция: `preset.access.disable()`.

Параметры:

- `plid` — уникальный идентификатор родительского списка (элемент `name` из функции `preset()`);

- `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `preset.access()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Импорт шаблонов, просмотр, изменение

Функция: `preset.import()`.

Одновременно используется для просмотра и изменения данных формы.

Просмотр данных

Результат — список параметров (табл. 5.34).

Изменение данных

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.34).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.34

Параметр	Описание	Параметр	Описание
`host`	Сервер	`passwd`	Пароль
`user`	Логин		

5.1.28. Настройки доменов по умолчанию

Функция: `domain.prop()`.

Данная функция одновременно используется для просмотра и изменения данных формы.

Просмотр данных

Результат — список параметров (табл. 5.35).

Изменение данных

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.35).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.35

Параметр	Описание
ns	Серверы имен (одно или несколько значений, разделенных пробелом)
mx	Почтовые серверы (одно или несколько значений, разделенных пробелом)
email	Адрес техподдержки
subdom	Дополнительные записи (одно или несколько значений, разделенных пробелом)
fixall	Изменить для всех доменов. Необязательный параметр. Чтобы включить данную опцию, используйте значение "on"

5.1.29. Ротация журналов WWW-домена

Функция: wwwdomain.log().

Одновременно используется для просмотра и изменения данных формы.

Просмотр данных

Результат — список параметров (табл. 5.36).

Изменение данных

Параметры:

- sok — значение параметра должно быть равно "yes";
- дополнительные параметры запроса (табл. 5.36).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.36

Параметр	Описание
rotper	Период ротации
arcnum	Количество сохраняемых архивов
errlog	Журнал ошибок
analyzer	Анализатор
analper	Период анализа
fixall	Применить для всех WWW-доменов. Необязательный параметр. Чтобы включить данную опцию, используйте значение "on"

5.1.30. FTP-аккаунты

Функция: ftp().

Результат — список элементов:

- name — имя;
- home — домашняя директория;

- ❒ `disabled` — доступ временно закрыт;
- ❒ `shell` — доступ к shell;
- ❒ `note` — заметки;
- ❒ `disk` — диск. Атрибуты:
  - `used` — использованное количество;
  - `limit` — максимально возможное значение.

Создание, изменение

Функция: `ftp.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `ftp()`);

Результат — список параметров объекта (табл. 5.37).

Создание объекта

Параметры:

- ❒ `sok` — значение параметра должно быть равно `"yes"`;
- ❒ дополнительные параметры запроса (табл. 5.37).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.37

Параметр	Описание
`name`	Имя
`passwd`	Пароль
`confirm`	Подтверждение
`htype`	Домашний каталог
`domain`	Домен
`dir`	Директория
`disklimit`	Дисковая квота (числовое значение, для указания "бесконечности" используйте "0")
`shell`	Доступ к shell. Необязательный параметр. Чтобы включить данную опцию, используйте значение `"on"`
`note`	Заметки

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `ftp()`);
- дополнительные параметры запроса (табл. 5.37).

Результат — успешное выполнение операции или сообщение об ошибке.

Удаление FTP-аккаунтов

Функция: `ftp.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `ftp()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Включение FTP-аккаунтов

Функция: `ftp.enable()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `ftp()`.

Результат — успешное выполнение операции или сообщение об ошибке.

Временное отключение FTP-аккаунтов

Функция: `ftp.disable()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `ftp()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.31. Редиректы (перенаправление URL)

Функция: `wwwredirect()`.

Результат — список элементов:

- `name` — путь;
- `url` — URL.

Создание, изменение, параметры перенаправления

Функция: `wwwredirect.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `wwwredirect()`).

Результат — список параметров объекта (табл. 5.38).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.38).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `wwwredirect()`);
- дополнительные параметры запроса (табл. 5.38).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.38

Параметр	Описание	Параметр	Описание
`domain`	WWW-домен	`url`	URL
`name`	URL-путь		

Удаление перенаправления

Функция: `wwwredirect.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `wwwredirect()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.32. Страницы ошибок

Функция: `errpage()`.

Результат — список элементов:

- `key` — уникальный идентификатор элемента списка;
- `domain` — WWW-домен;
- `name` — код ошибки;
- `url` — URL.

Создание, изменение, параметры страницы ошибки

Функция: `errpage.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `key` из функции `errpage()`).

Результат — список параметров объекта (табл. 5.39).

Таблица 5.39

Параметр	Описание
`Domain`	WWW-домен
`name`	Код ошибки. Возможные значения: • 400 — Bad Request; • 401 — Unauthorized; • 402 — Payment Required; • 403 — Forbidden; • 404 — Not Found; • 405 — Method Not Allowed; • 406 — Not Acceptable; • 407 — Proxy Authentication Required; • 408 — Request Timeout; • 409 — Conflict; • 410 — Gone; • 411 — Length Required; • 412 — Precondition Failed; • 413 — Request Entity Too Large; • 414 — Request-URI Too Long; • 415 — Unsupported Media Type; • 416 — Requested Range Not Satisfiable; • 417 — Expectation Failed; • 500 — Internal Server Error; • 501 — Not Implemented; • 502 — Bad Gateway; • 503 — Service Unavailable; • 504 — Gateway Timeout; • 505 — HTTP Version Not Supported
`url`	URL

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.39).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `key` из функции `errpage()`);
- дополнительные параметры запроса (табл. 5.39).

Результат — успешное выполнение операции или сообщение об ошибке.

Удаление страницы ошибки

Функция: `errpage.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `key` из функции `errpage()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.33. Ограничение доступа к каталогу

Функция: `diraccess()`.

Результат — список элементов: `name` — путь.

Просмотр, изменение

Функция: `diraccess.edit()`.

Данная функция одновременно используется для просмотра и изменения данных формы.

Просмотр данных

Результат — список параметров (табл. 5.40).

Изменение данных

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.40).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.40

Параметр	Описание
name	Путь

Снятие защиты с каталога

Функция: diraccess.delete().

Параметр: elid — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент name из функции diraccess().

Результат — успешное выполнение операции или сообщение об ошибке.

Пользователи защищенного каталога

Функция: diraccess.user().

Параметр: elid — уникальный идентификатор (элемент name из функции diraccess()).

Результат — список элементов: name — имя пользователя.

Создание, изменение, параметры пользователя

Функция: diraccess.user.edit().

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметры:

- plid — уникальный идентификатор родительского списка (элемент name из функции diraccess());
- elid — уникальный идентификатор (элемент name из функции diraccess.user()).

Результат — список параметров объекта (табл. 5.41).

Создание объекта

Параметры:

- sok — значение параметра должно быть равно "yes";
- plid — уникальный идентификатор родительского списка (элемент name из функции diraccess());
- дополнительные параметры запроса (табл. 5.41).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- sok — значение параметра должно быть равно "yes";

- plid — уникальный идентификатор родительского списка (элемент name из функции diraccess());
- elid — уникальный идентификатор (элемент name из функции diraccess.user());
- дополнительные параметры запроса (табл. 5.41).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.41

Параметр	Описание	Параметр	Описание
name	Имя пользователя	confirm	Подтверждение
passwd	Пароль		

Удаление пользователей

Функция: diraccess.user.delete().

Параметры:

- plid — уникальный идентификатор родительского списка (элемент name из функции diraccess());
- elid — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент name из функции diraccess.user().

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.34. Почтовые группы

Функция: emailgroup().

Результат — список элементов:

- name — e-mail адрес;
- members — участники.

Создание, изменение, параметры почтовой группы

Функция: emailgroup.edit().

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: elid — уникальный идентификатор (элемент name из функции emailgroup()).

Результат — список параметров объекта (табл. 5.42).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.42).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `emailgroup()`);
- дополнительные параметры запроса (табл. 5.42).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.42

Параметр	Описание
name	Имя группы
domain	Почтовый домен
members	Участники (одно или несколько значений, разделенных пробелом)

Удаление почтовых групп

Функция: `emailgroup.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `emailgroup()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.35. Почтовые редиректы

Функция: `emailredirect()`.

Результат — список элементов:

- `name` — адрес электронной почты;
- `forward` — цель перенаправления.

Создание, изменение, параметры почтового редиректа

Функция: `emailredirect.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `name` из функции `emailredirect()`).

Результат — список параметров объекта (табл. 5.43).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.43).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `name` из функции `emailredirect()`);
- дополнительные параметры запроса (табл. 5.43).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.43

Параметр	Описание	Параметр	Описание
`name`	Имя редиректа	`forward`	Перенаправлять на домен
`domain`	Почтовый домен		

Удаление почтовых редиректов

Функция: `emailredirect.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `name` из функции `emailredirect()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.1.36. Почтовые автоответчики

Функция: `emailresponder()`.

Результат — список элементов: `email` — адрес электронной почты.

Создание, изменение, параметры автоответчика

Функция: `emailresponder.edit()`.

Данная функция одновременно используется для просмотра параметров объекта, изменения объекта и создания нового объекта.

Просмотр параметров объекта

Параметр: `elid` — уникальный идентификатор (элемент `email` из функции `emailresponder()`).

Результат — список параметров объекта (табл. 5.44).

Создание объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- дополнительные параметры запроса (табл. 5.44).

Результат — успешное выполнение операции или сообщение об ошибке.

Изменение объекта

Параметры:

- `sok` — значение параметра должно быть равно `"yes"`;
- `elid` — уникальный идентификатор (элемент `email` из функции `emailresponder()`);
- дополнительные параметры запроса (табл. 5.44).

Результат — успешное выполнение операции или сообщение об ошибке.

Таблица 5.44

Параметр	Описание	Параметр	Описание
`name`	Имя	`subject`	Тема
`domain`	Домен	`body`	Текст
`rtype`	Действие	`file`	Файл
`command`	Скрипт		

Удаление почтовых автоответчиков

Функция: `emailresponder.delete()`.

Параметр: `elid` — один или несколько уникальных идентификаторов объекта, разделенных запятой и следующим за ней пробелом (", "). Уникальный идентификатор — это элемент `email` из функции `emailresponder()`.

Результат — успешное выполнение операции или сообщение об ошибке.

5.2. Сайт-тренажер для изучения запросов к API ISPmanager

От теории перейдем к практике. Создадим сайт, где будем автоматически формировать запросы к API ISPmanager и смотреть ответы сервера и результаты наших запросов.

Рассматривать все команды, думаю, нет необходимости, обсудим только следующие возможности:

- операции с шаблонами (тарифными планами):
  - просмотр всех шаблонов;
  - создание нового шаблона;
  - редактирование шаблона;
  - удаление шаблона;
- операции с пользователями:
  - просмотр всех пользователей;
  - создание нового пользователя;
  - редактирование параметров пользователя;
  - удаление пользователя.

Все файлы проекта находятся на прилагаемом компакт-диске в папке glava_05\api_isp_manager.

5.2.1. Получение доступа к демо-серверу с ISPmanager

Если на вашем сервере установлен ISPmanager, это, конечно, замечательно. А если нет? Тогда воспользуемся демонстрационной онлайн-версией. Демострационный сервер с панелью ISPmanager открывается на 8 часов, но в случае отсутствия активности на нем в течение часа автоматически закрывается. Один посетитель не может открыть более одной демонстрации ISPmanager Lite и одной демонстрации ISPmanager Pro в течение суток. Доступ к демо-серверу будет открыт только для подсети, с которой открыта демострация ISPmanager, как для входящих, так и для исходящих соединений любого типа. Сервер, открываемый в рамках демонстрации ISPmanager, является полнофункциональным UNIX-сервером, с полным набором работающих сервисов. Переходим по ссылке **http://ispsystem.com/software/ispmanager/demo/** и на открывшейся странице (рис. 5.1) нажимаем на кнопку **Демо ISPmanager Pro**.

Получаем данные для доступа к демо-серверу (рис. 5.2).

5.2.2. Создание формы получения данных ISPmanager

Для получения доступа к API ISPmanager будем использовать авторизацию по HTTP с помощью `authinfo` *(см. подразд. "Авторизация с использованием параметра authinfo" разд. 5.1.1)*. Так как параметры демо-сервера непостоянны, предусмотрим возможность ввода параметров сервера и данных для авторизации через форму. Вид формы представлен на рис. 5.3.

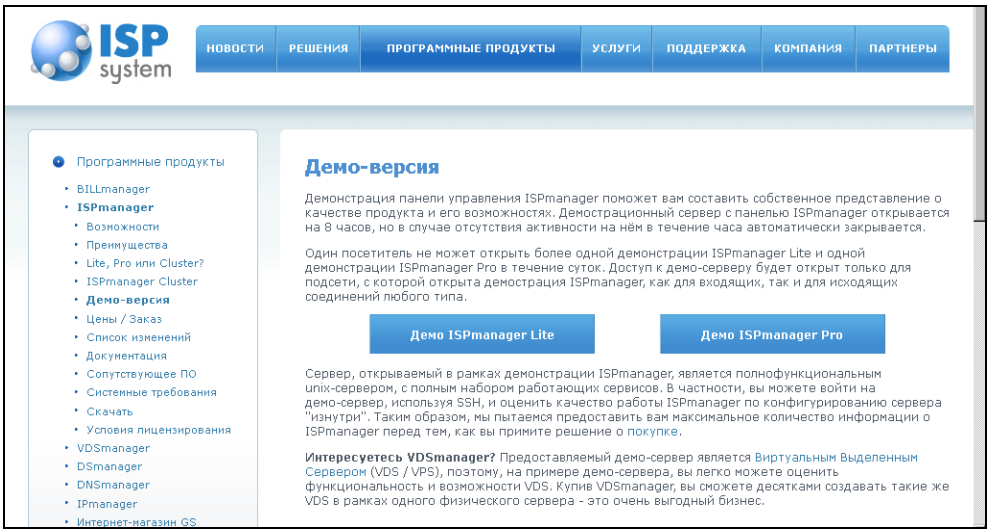


Рис. 5.1. Форма получения демо-версии

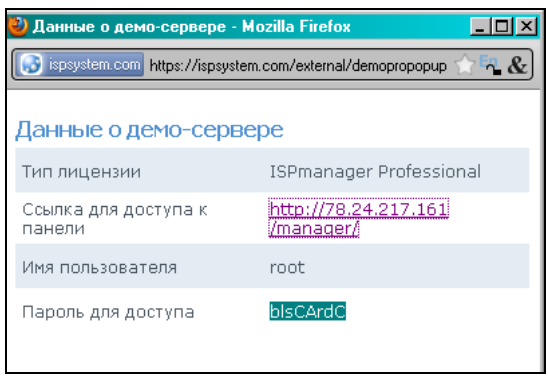


Рис. 5.2. Данные для доступа к серверу

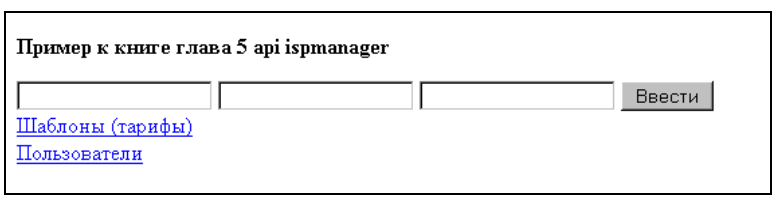


Рис. 5.3. Форма ввода данных для доступа к серверу

Разметка страницы и подключение библиотеки xAjax, которую мы будем использовать при создании сайта, содержатся в файле index.php, содержимое которого представлено в листинге 5.1.

В блоке `id=view_command` мы будем видеть команду отправки запроса к API ISPmanager, в блоке `id=reply_server` — ответ сервера на запрос.

Листинг 5.1

```
<?
// подключение библиотеки xajax
.........
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="ru" xml:lang="ru">
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
 <title>Тренажер команд API ISPmanager</title>
 <link rel="stylesheet" type="text/css" href="style.css" media="all" />
<?php $xajax->printJavascript(''); ?>
</head>
<body >
  <p><b>Пример к главе 5 "API ISPmanager"</b></p>
  <!-- настройки  -->
  <div id=options>
    <form name=form_options id=form_options action='javascript:void();'
       onsubmit='xajax.$("button_options").disabled=true;
       xajax_Set_Options(xajax.getFormValues("form_options"));'>
     <input type=text name=server id=server
      value="<?php echo ISP_SERVER; ?>" <?php echo $readonly; ?> >
     <input type=text name=login id=login
      value="<?php echo ISP_LOGIN; ?>" <?php echo $readonly; ?> >
     <input type=text name=password id=password
      value="<?php echo ISP_PASS; ?>" <?php echo $readonly; ?> >
     <input type=submit id=button_options value="Ввести"
       <?php echo $disabled; ?> >
    </form>
  </div>
  <!-- меню  -->
  <div id=menu>
    <div><a href='javascript:void();' onclick='
         xajax_View_Tarifs();'>Шаблоны (тарифы)</a></div>
    <div><a href='javascript:void();' onclick='
         xajax_View_Users();'>Пользователи </a></div>
  </div>
  <!-- список -->
  <div id=all>
  </div>
  <!-- просмотр редактирование -->
  <div id=view1>
  </div>
  <!-- команда -->
  <div id=view_command>
  </div>
```

```
  <!-- ответ сервера -->
  <div id=reply_server>
  </div>
</body>
</html>
```

После ввода параметров доступа к демо-серверу (см. рис. 5.2) и нажатия кнопки **Ввести** вызывается xAjax-функция `Set_Options()`, которая запоминает параметры сервера и данные для входа в переменных `SESSION` и блокирует поля формы от изменения. xAjax-функция `Set_Options()` находится в файле set_options.php, содержимое которого представлено в листинге 5.2.

Листинг 5.2

```
<?php
// Установить настройки для доступа к серверу ISPmanager
function Set_Options($Id)
{ session_start();
  $objResponse = new xajaxResponse();
  // установить переменные SESSION
  $_SESSION[server]=$Id[server];
  $_SESSION[login]=$Id[login];
  $_SESSION[password]=$Id[password];
  // сделать поля readonly
  $script="document.getElementById('server').setAttribute(
          'readonly','true');";
  $script.="document.getElementById('login').setAttribute(
          'readonly','true');";
  $script.="document.getElementById('password').setAttribute(
          'readonly','true');";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

5.2.3. Получение списка шаблонов (тарифных планов)

Теперь напишем скрипт автоматического формирования запроса к API ISPmanager. При выборе пункта меню **Шаблоны (тарифы)** вызывается xAjax-функция `View_Tarifs()`, которая формирует запрос на получение списка шаблонов, отправляет запрос, получает ответ, обрабатывает его и выводит в табличном виде список тарифных планов и ссылки на функции редактирования и удаления тарифного плана, а также ссылку на функцию добавления нового тарифного плана. Кроме того, на страницу выводится сам запрос и ответ сервера в формате XML (рис. 5.4). xAjax-функция `View_Tarifs()` расположена в файле view_tarifs.php, содержимое которого представлено в листинге 5.3.

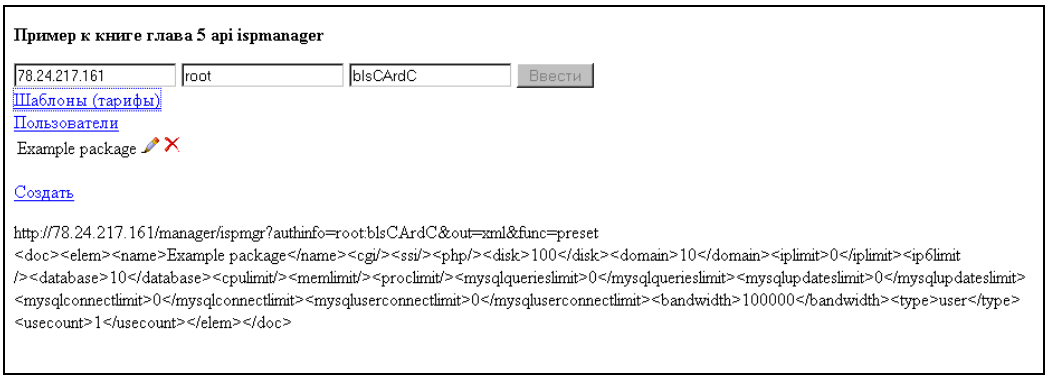


Рис. 5.4. Получение списка шаблонов (тарифных планов)

Листинг 5.3

```
<?php
// Запрос получения шаблонов (тарифов) и вывод
function View_Tarifs()
{ $objResponse = new xajaxResponse();

  $query="";$content="";
  // сформировать запрос
  $query.="http://".$_SESSION[server]."/manager/ispmgr?";
  $query.="authinfo=".$_SESSION[login].":".$_SESSION[password]."&out=xml";
  $query.="&func=preset";
  // получить ответ  +  сформировать контент
  $xml = @file_get_contents($query);    // получаем atom feed
  $data = @simplexml_load_string($xml); // разбираем atom
  $records=$data->xpath('elem');
  if (empty($records))
  { $content.="<p> Шаблонов нет </p>"; }
  else
  { $content.="<table>";
    foreach ($records as $record)
    { $content.="<tr><td>".$record->name."</td>";
      $content.="<td><a href='javascript:void();'
              onclick='xajax_Form_Edit_Tarif(\"".$record->name."\");'>
              <img src='edit.png'></a></td>";
      $content.="<td><a href='javascript:void();'
              onclick='xajax_Delete_Tarif(\"".$record->name."\");'>
              <img src='delete.png'></a></td></tr>";
    }
    $content.="<table>";
  }
  $content.="<p><a href='javascript:void();'
                  onclick='xajax_Form_Add_Tarif();'>Создать</a></p>";
```

```
  // вывод контента
  $objResponse->assign("all","innerHTML",$content);
  $objResponse->assign("view1","innerHTML","");
  // вывод запроса
  $objResponse->assign("view_command","innerHTML",$query);
  // вывод ответа сервера
  $xml=substr($xml,39,strlen($xml)-40);
  $script="var t1=document.getElementById('reply_server');";
  $script.="if(document.all) t1.innerText='".$xml."';";
  $script.="else t1.textContent='".$xml."';";
  $objResponse->script($script);

  return $objResponse;
}
?>
```

5.2.4. Добавление нового шаблона

Для добавления нового шаблона щелкнем по ссылке **Создать**. Откроется форма создания нового шаблона (рис. 5.5). Создание контента формы в xAjax-функции `Form_Add_Tarif()`. Эта функция расположена в файле form_add_tarif.php, содержимое которого представлено в листинге 5.4.

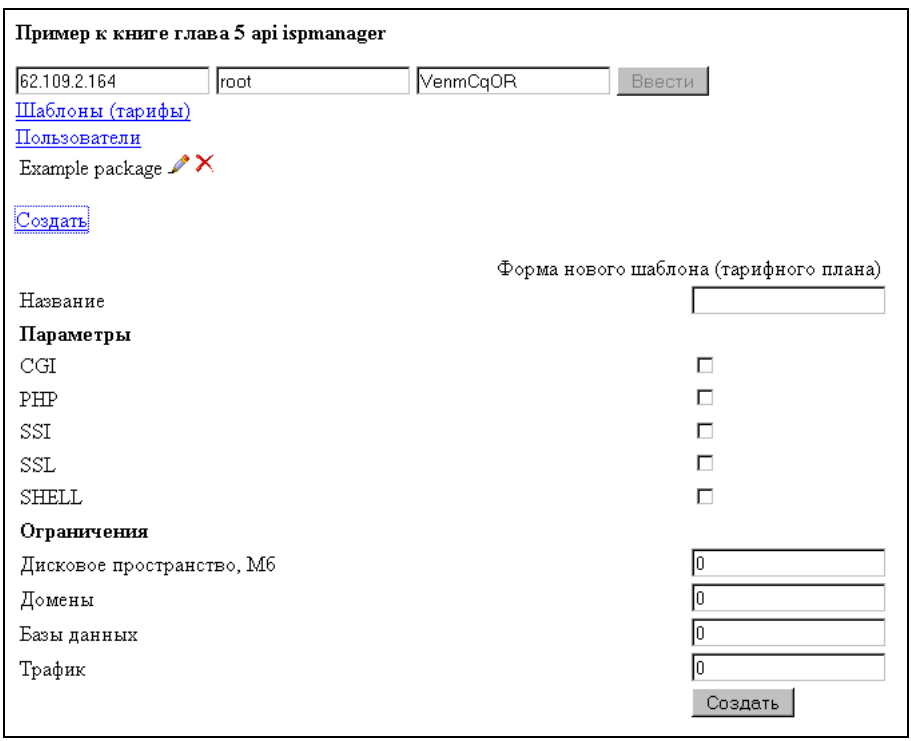


Рис. 5.5. Форма создания нового шаблона

Листинг 5.4

```
<?php
// Форма создания нового шаблона (тарифного плана)
function Form_Add_Tarif()
{ $objResponse = new xajaxResponse();
  // вывод контента
  $content="<form id='FormAddTarif' action='javascript:void(null);'
       onsubmit='xajax_Add_Tarif(xajax.getFormValues(\"FormAddTarif\"));'>";
  $content.="<table width=100%>";
  $content.="<caption>Форма нового шаблона (тарифного плана)</caption>";
  $content.="<tr><td width=50%>Название </td>
           <td width=50%><input type='text' name='name'
             size='20' maxlength='20'></td></tr>";
  $content.="<tr><td width=50%><b>Параметры</b></td>
           <td width=50%></td></tr>";
  $content.="<tr><td width=50%>CGI</td>
           <td width=50%><input type='checkbox' name='cgi'>
           </td></tr>";
  $content.="<tr><td width=50%>PHP</td>
           <td width=50%><input type='checkbox' name='php'>
           </td></tr>";
  $content.="<tr><td width=50%>SSI</td>
           <td width=50%><input type='checkbox' name='ssi'>
           </td></tr>";
  $content.="<tr><td width=50%>SSL</td>
           <td width=50%><input type='checkbox' name='ssl'>
           </td></tr>";
  $content.="<tr><td width=50%>SHELL</td>
           <td width=50%><input type='checkbox' name='shell'>
           </td></tr>";
  $content.="<tr><td width=50%><b>Ограничения</b></td>
           <td width=50%></td></tr>";
  $content.="<tr><td width=50%>Дисковое пространство, Мб</td>
           <td width=50%><input type='text' name='disklimit'
           size=20 maxlength='4' value='0'></td></tr>";
  $content.="<tr><td width=50%>Домены</td>
           <td width=50%><input type='text' name='domainlimit'
           size=20 maxlength='2' value='0'></td></tr>";
  $content.="<tr><td width=50%>Базы данных</td>
           <td width=50%><input type='text' name='baselimit'
           size=20 maxlength='2' value='0'></td></tr>";
  $content.="<tr><td width=50%>Трафик</td>
           <td width=50%><input type='text' name='bandwidthlimit'
           size=20 maxlength='7' value='0'></td></tr>";
  $content.="<tr><td></td>
           <td><input type='submit' value='Создать' d</td></tr>";
```

```
  $content.="</table>";
  $content.="</form>";
  // вывод контента
  $objResponse->assign("view1","innerHTML",$content);
  // очистить вывод запроса
  $objResponse->assign("view_command","innerHTML","");
  // очистить вывод ответа сервера
  $objResponse->assign("reply_server","innerHTML","");
  return $objResponse;
}
?>
```

При нажатии кнопки **Создать** вызывается xAjax-функция `Add_Tarif()`. Из полученных из формы данных функция формирует запрос на создание нового тарифа, отправляет запрос, получает ответ и выводит на страницу сам запрос и ответ сервера в формате XML (рис. 5.6). xAjax-функция `Add_Tarif()` расположена в файле add_tarif.php, содержимое которого представлено в листинге 5.5.

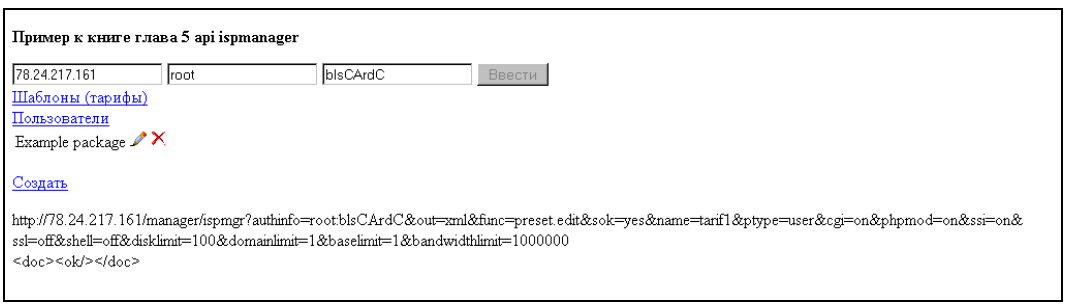


Рис. 5.6. Создание нового шаблона

Листинг 5.5

```
<?php
// Получение данных из формы и отправка команд создания нового шаблона
// или изменения параметров существующего
function Add_Tarif($Id)
{ $objResponse = new xajaxResponse();
  // сформировать запрос
  $query.="http://".$_SESSION[server]."/manager/ispmgr?";
  $query.="authinfo=".$_SESSION[login].":".$_SESSION[password]."&out=xml";
  $query.="&func=preset.edit&sok=yes";
  if(isset($Id[elid]))  $query.="&elid=".$Id[elid];
  // проверка параметров
  $query.="&name=".$Id[name];
  $query.="&ptype=user";
  if(isset($Id[cgi])) $query.="&cgi=on";
  else $query.="&cgi=off";
  if(isset($Id[php])) $query.="&phpmod=on";
```

```
  else $query.="&php=off";
  if(isset($Id[ssi])) $query.="&ssi=on";
  else $query.="&ssi=off";
  if(isset($Id[ssl])) $query.="&ssl=on";
  else $query.="&ssl=off";
  if(isset($Id[shell])) $query.="&shell=on";
  else $query.="&shell=off";
  $query.="&disklimit=".$Id[disklimit];
  $query.="&domainlimit=".$Id[domainlimit];
  $query.="&baselimit=".$Id[baselimit];
  $query.="&bandwidthlimit=".$Id[bandwidthlimit];

  $xml = @file_get_contents($query);    // получаем atom feed
  $data = @simplexml_load_string($xml); // разбираем atom
  // очистить блок
  $objResponse->assign("view1","innerHTML","");
  // вывод запроса
  $objResponse->assign("view_command","innerHTML",$query);
  // вывод ответа сервера
  $xml=substr($xml,39,strlen($xml)-40);
  $script="var t1=document.getElementById('reply_server');";
  $script.="if(document.all) t1.innerText='".$xml."';";
  $script.="else t1.textContent='".$xml."';";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

Пример к книге глава 5 api ispmanager

78.24.217.161 | root | blsCArdC | Ввести

Шаблоны (тарифы)

Пользователи

Example package

tarif1

Создать

http://78.24.217.161/manager/ispmgr?authinfo=root:blsCArdC&out=xml&func=preset
<doc><elem><name>Example package</name><cgi/><ssi/><php/><disk>100</disk><domain>10</domain><
/><database>10</database><cpulimit/><memlimit/><proclimit/><mysqlquerieslimit>0</mysqlquerieslimit><mys
<mysqlconnectlimit>0</mysqlconnectlimit><mysqluserconnectlimit>0</mysqluserconnectlimit><bandwidth>1000
<usecount>1</usecount></elem><elem><name>tarif1</name><cgi/><ssi/><php/><disk>100</disk><domain>
<database>1</database><cpulimit>0</cpulimit><memlimit>0</memlimit><proclimit>0</proclimit><mysqlquerie
<mysqlupdateslimit>0</mysqlupdateslimit><mysqlconnectlimit>0</mysqlconnectlimit><mysqluserconnectlimit>0<
<bandwidth>1000000</bandwidth><type>user</type><usecount>0</usecount></elem></doc>

Рис. 5.7. Проверка добавления нового шаблона запросом

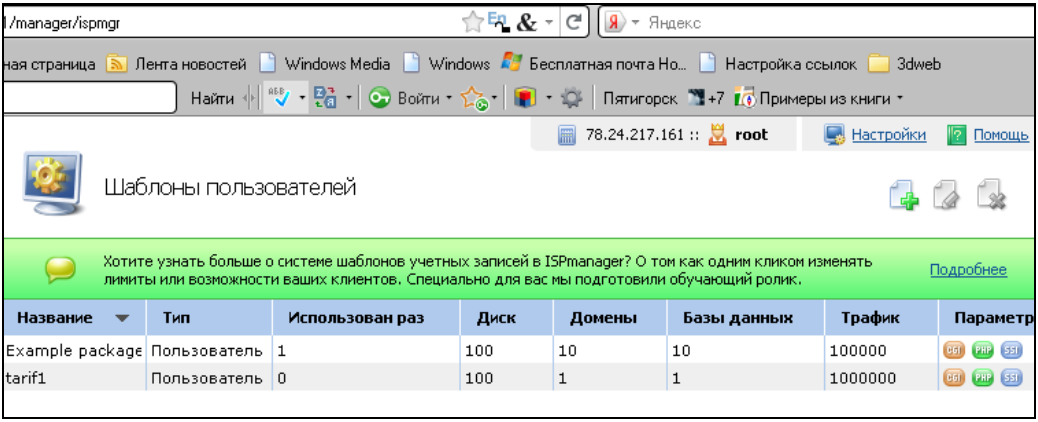


Рис. 5.8. Проверка добавления нового шаблона на сервере

Создадим шаблон с именем tarif1 и посмотрим, что получается при запросе списка тарифов (рис. 5.7) и что оказалось на сервере (рис. 5.8). Шаблон добавлен.

5.2.5. Редактирование шаблона

Для редактирования шаблона щелкнем по значку редактирования. При этом вызывается xAjax-функция `Form_Edit_Tarif()`. Она создает запрос к API ISPmanager для получения параметров выбранного шаблона, получает с сервера параметры запрошенного шаблона, создает форму редактирования данных шаблона (рис. 5.9). Функция `Form_Edit_Tarif()` расположена в файле form_edit_tarif.php, содержимое которого представлено в листинге 5.6.

Листинг 5.6

```
<?php
// Форма редактирования шаблона (тарифного плана)
function Form_Edit_Tarif($name)
{ $objResponse = new xajaxResponse();
  // запрос о тарифе
  $query.="http://".$_SESSION[server]."/manager/ispmgr?";
  $query.="authinfo=".$_SESSION[login].":".$_SESSION[password]."&out=xml";
  $query.="&func=preset.edit&elid=".$name;
  // получить ответ  +  сформировать контент
  $xml = @file_get_contents($query);    // получаем atom feed
  $data = @simplexml_load_string($xml); // разбираем atom
  // вывод контента
  $content="<form id='FormEditTarif' action='javascript:void(null);'
      onsubmit='xajax_Add_Tarif(xajax.getFormValues(\"FormEditTarif\"));'>";
  $content.="<table width=100%>";
  $content.="<caption>Форма редактирования шаблона
            (тарифного плана)</caption>";
```

```
$content.="<tr><td width=50%> </td>
         <td width=50%><input type='hidden' name='elid'
         value='".$name."' </td></tr>";
$content.="<tr><td width=50%>Название </td>
         <td width=50%><input type='text' name='name' value='".$data-
         >name."' size='20' maxlength='20'></td></tr>";
$content.="<tr><td width=50%>Тип (6-12)</td>
         <td width=50%><select name='ptype'>
           <option value='user' selected>user
           <option value='reseller' disabled>reseller
           </select></td></tr>";
$content.="<tr><td width=50%><b>Параметры</b></td>
         <td width=50%></td></tr>";
$content.="<tr><td width=50%>CGI</td>
         <td width=50%><input type='checkbox' name='cgi'
         ".(($data->cgi=='on')?'checked':'').">
         </td></tr>";
$content.="<tr><td width=50%>PHP</td>
         <td width=50%><input type='checkbox' name='php'
         ".(($data->php=='on')?'checked':'').">
         </td></tr>";
$content.="<tr><td width=50%>SSI</td>
         <td width=50%><input type='checkbox' name='ssi'
         ".(($data->ssi=='on')?'checked':'').">
         </td></tr>";
$content.="<tr><td width=50%>SSL</td>
         <td width=50%><input type='checkbox' name='ssl'
         ".(($data->ssl=='on')?'checked':'').">
         </td></tr>";
$content.="<tr><td width=50%>SHELL</td>
         <td width=50%><input type='checkbox' name='shell'
         ".(($data->shell=='on')?'checked':'').">
         </td></tr>";

$content.="<tr><td width=50%><b>Ограничения</b></td>
         <td width=50%></td></tr>";
$content.="<tr><td width=50%>Дисковое пространство, Мб</td>
         <td width=50%><input type='text' name='disklimit'
          value='".$data->disklimit."' size=20 maxlength='4'
          value='0'></td></tr>";
$content.="<tr><td width=50%>Домены</td>
         <td width=50%><input type='text' name='domainlimit'
          value='".$data->domainlimit."' size=20 maxlength='2'
          value='0'></td></tr>";
$content.="<tr><td width=50%>Базы данных</td>
         <td width=50%><input type='text' name='baselimit'
```

```
            value='".$data->baselimit."' size=20 maxlength='2'
            value='0'></td></tr>";
  $content.="<tr><td width=50%>Трафик</td>
            <td width=50%><input type='text' name='bandwidthlimit'
            value='".$data->bandwidthlimit."' size=20 maxlength='7'
            value='0'></td></tr>";

  $content.="<tr><td></td>
            <td><input type='submit' value='Изменить' d</td></tr>";
  $content.="</table>";
  $content.="</form>";
  // вывод контента
  $objResponse->assign("view1","innerHTML",$content);
  // вывод запроса
  $objResponse->assign("view_command","innerHTML","");
  // вывод ответа сервера
  $objResponse->assign("reply_server","innerHTML","");
  return $objResponse;
}
?>
```

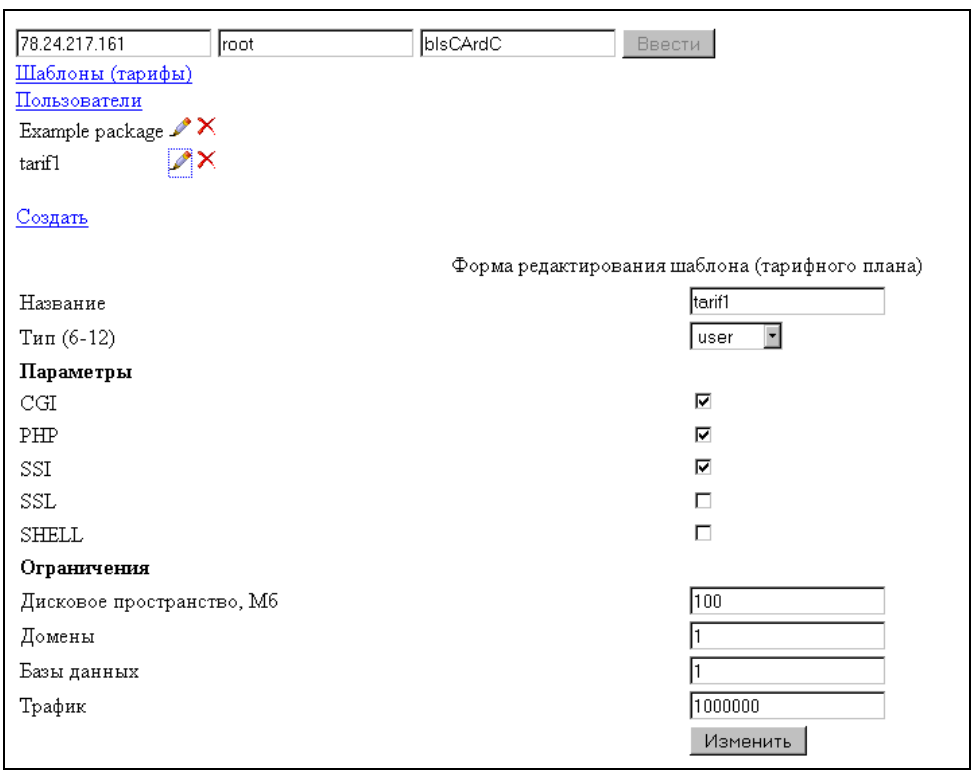


Рис. 5.9. Форма редактирования шаблона

Изменим данные выбранного шаблона (рис. 5.10). При нажатии кнопки **Изменить** вызывается xAjax-функция `Add_Tarif()`, рассмотренная нами в *разд. 5.2.4*. Результат выполнения функции представлен на рис. 5.11, проверяем новые параметры шаблона на сервере (рис. 5.12).

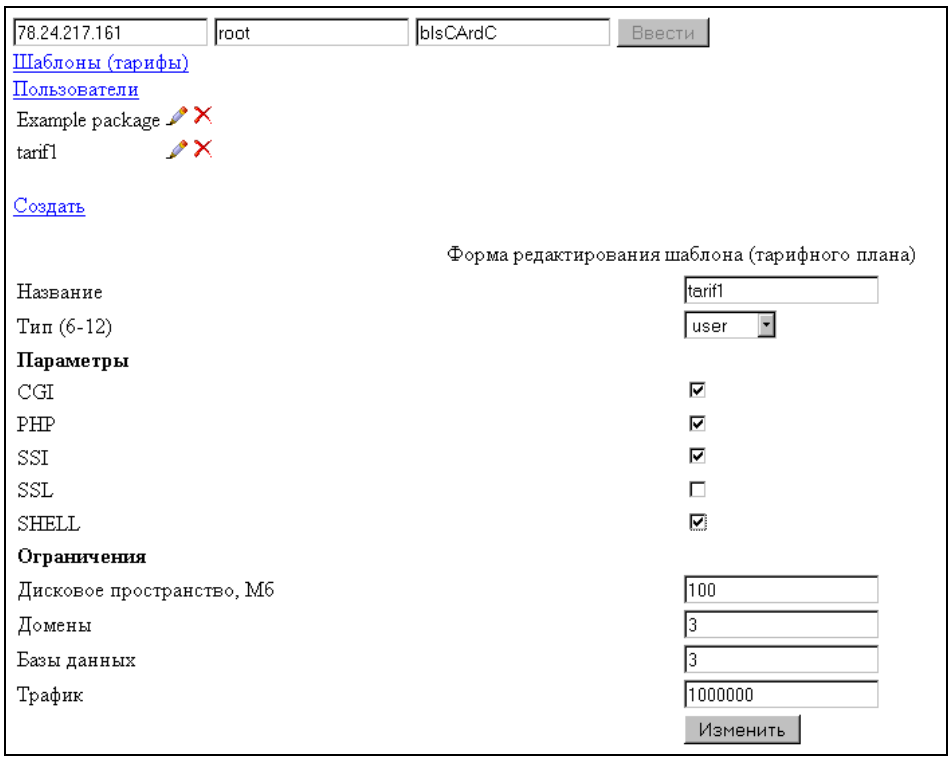


Рис. 5.10. Редактируем данные шаблона запроса

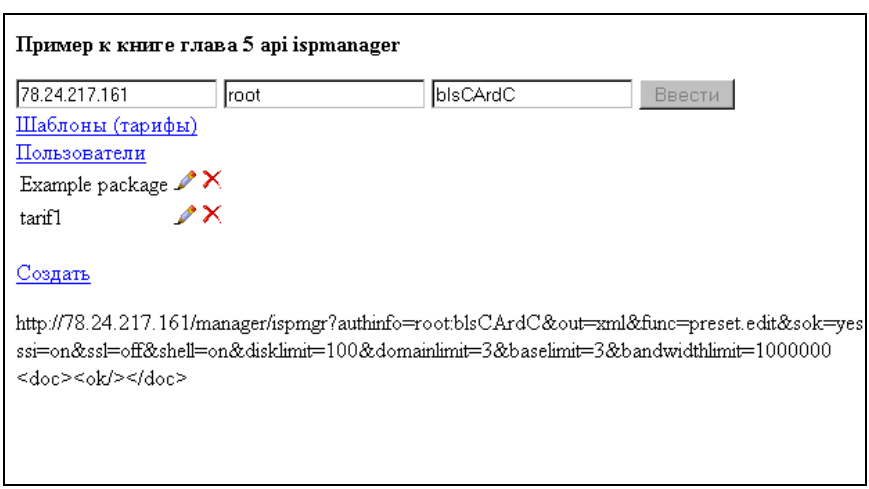


Рис. 5.11. Результат выполнения запроса

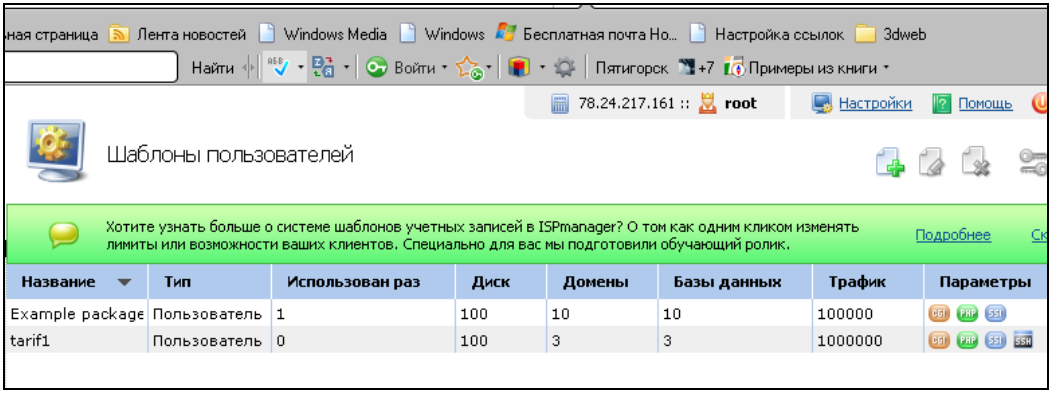


Рис. 5.12. Проверка редактирования шаблона на сервере

5.2.6. Удаление шаблона

Для удаления шаблона щелкнем по значку удаления. Вызывается xAjax-функция `Delete_Tarif()`, которая создает запрос к API ISPmanager для удаления выбранного шаблона. Функция `Delete_Tarif()` расположена в файле delete_tarif.php, содержимое которого представлено в листинге 5.7.

Листинг 5.7

```
<?php
// Отправка команд удаления шаблона
function Delete_Tarif($elid)
{ $objResponse = new xajaxResponse();
  // сформировать запрос
  $query.="http://".$_SESSION[server]."/manager/ispmgr?";
  $query.="authinfo=".$_SESSION[login].":".$_SESSION[password]."&out=xml";
  $query.="&func=preset.delete&elid=".$elid;
  $xml = @file_get_contents($query);    // получаем atom feed
  $data = @simplexml_load_string($xml); // разбираем atom
  // очистить блок
  $objResponse->assign("view1","innerHTML","");
  // вывод запроса
  $objResponse->assign("view_command","innerHTML",$query);
  // вывод запроса
  $objResponse->assign("view_command","innerHTML",$query);
  // вывод ответа сервера
  $xml=substr($xml,39,strlen($xml)-40);
  $script="var t1=document.getElementById('reply_server');";
  $script.="if(document.all) t1.innerText='".$xml."';";
  $script.="else t1.textContent='".$xml."';";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

Результат выполнения функции представлен на рис. 5.13, получаем список шаблонов после удаления (рис. 5.14), проверяем новые параметры шаблона на сервере (рис. 5.15).

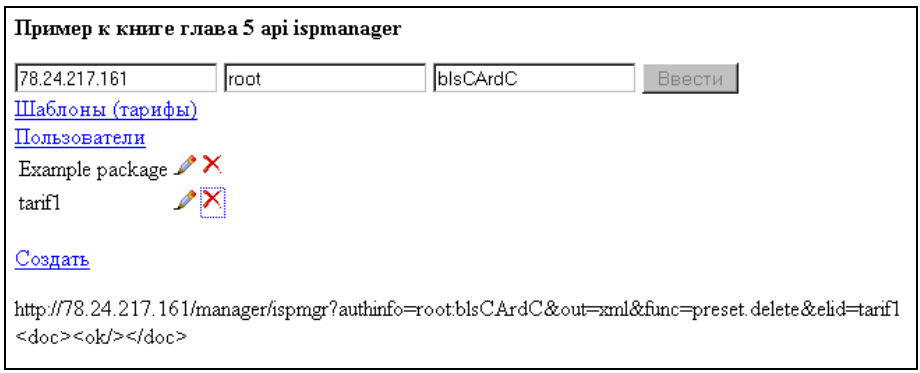


Рис. 5.13. Результат выполнения запроса

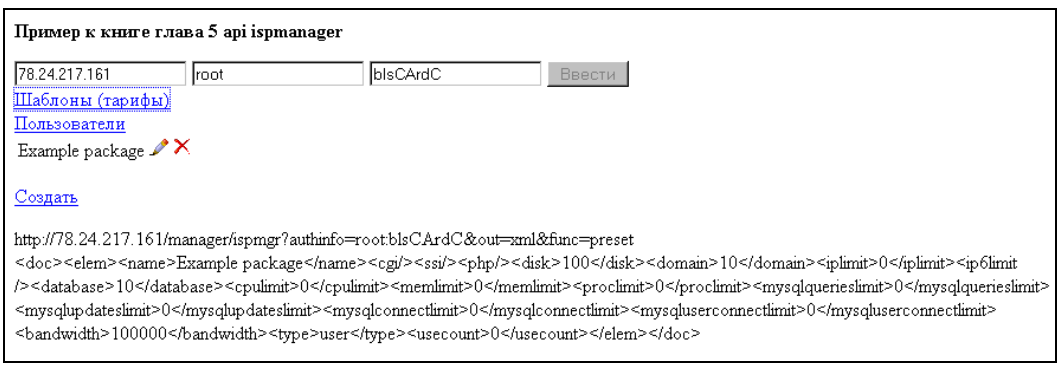


Рис. 5.14. Список шаблонов после удаления

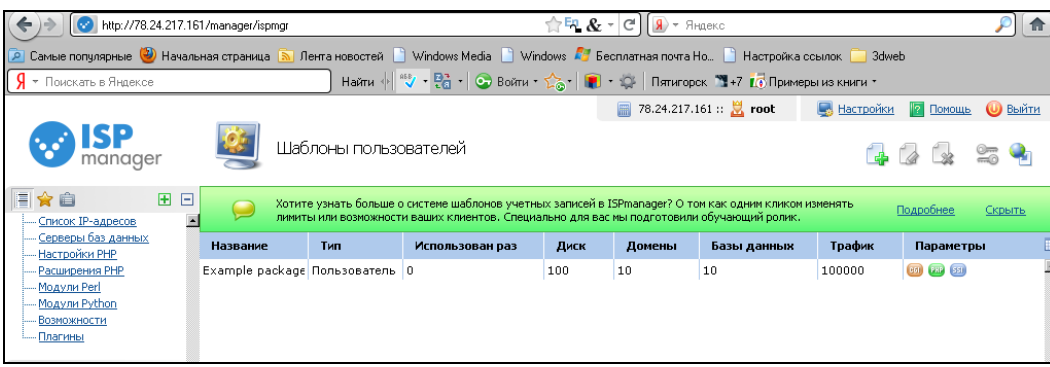


Рис. 5.15. Проверка выполнения запроса на сервере

5.2.7. Получение списка пользователей

Теперь получим список пользователей. При выборе пункта меню **Пользователи** вызывается xAjax-функция `View_Users()`, которая формирует запрос на получение списка шаблонов, отправляет запрос, получает ответ, обрабатывает его и выводит в табличном виде список тарифных планов и ссылки на функции редактирования и удаления тарифного плана, а также ссылку на функцию добавления нового пользователя. На страницу выводится сам запрос и ответ сервера в формате XML (рис. 5.16). xAjax-функция `View_Users()` расположена в файле view_users.php, содержимое которого представлено в листинге 5.8.

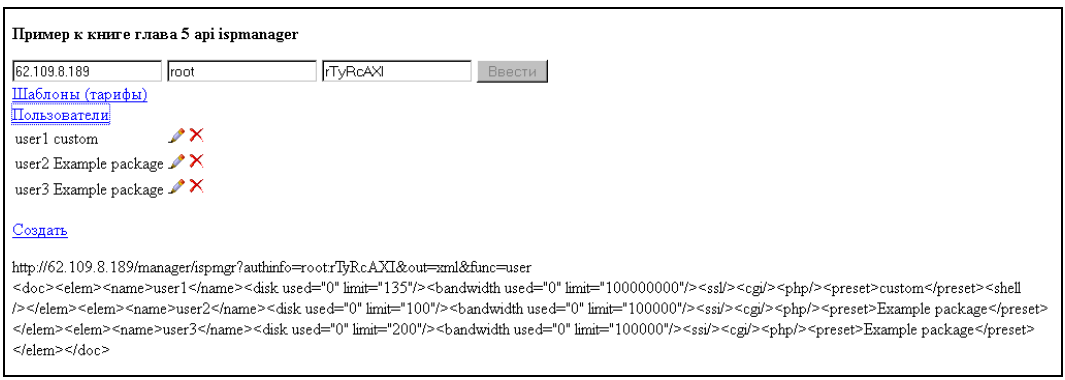


Рис. 5.16. Получение списка пользователей

Листинг 5.8

```
<?php
// Запрос получения пользователей и вывод
function View_Users()
{ $objResponse = new xajaxResponse();
  $query="";
  $reply_server="";
  $content="";
  // сформировать запрос
  $query.="http://".$_SESSION[server]."/manager/ispmgr?";
  $query.="authinfo=".$_SESSION[login].":".$_SESSION[password]."&out=xml";
  $query.="&func=user";
  // получить ответ  +  сформировать контент
  $xml = @file_get_contents($query);    // получаем atom feed
  $data = @simplexml_load_string($xml); // разбираем atom
  $records=$data->xpath('elem');
  if (empty($records))
  { $content.="<p> Пользователей нет </p>";  }
  else
  { $content.="<table>";
    foreach ($records as $record)
```

```
    { $content.="<tr><td>".$record->name."</td>";
      $content.="<td>".$record->preset."</td>";
      $content.="<td><a href='javascript:void();'
              onclick='xajax_Form_Edit_User(\"".$record->name."\");'>
              <img src='edit.png'></a></td>";
      $content.="<td><a href='javascript:void();'
              onclick='xajax_Delete_User(\"".$record->name."\");'>
              <img src='delete.png'></a></td></tr>";
    }
    $content.="<table>";
  }
  $content.="<p><a href='javascript:void();'
           onclick='xajax_Form_Add_User();'>Создать</a></p>";
  // вывод контента
  $objResponse->assign("all","innerHTML",$content);
  $objResponse->assign("view1","innerHTML","");
  // вывод запроса
  $objResponse->assign("view_command","innerHTML",$query);
  // вывод ответа сервера
  $xml=substr($xml,39,strlen($xml)-40);
  $script="var t1=document.getElementById('reply_server');";
  $script.="if(document.all) t1.innerText='".$xml."';";
  $script.="else t1.textContent='".$xml."';";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

5.2.8. Добавление нового пользователя

Для добавления нового пользователя щелкнем по ссылке **Создать**. Откроется форма создания нового пользователя (рис. 5.17). Создание контента формы выполняется в xAjax-функции `Form_Add_User()`. Эта функция расположена в файле form_add_user.php, содержимое которого представлено в листинге 5.9.

Листинг 5.9

```
<?php
// Форма создания нового пользователя
function Form_Add_User()
{ $objResponse = new xajaxResponse();
  // вывод контента
  $content="<form id='FormAddUser' action='javascript:void(null);'
          onsubmit='xajax_Add_User(xajax.getFormValues(\"FormAddUser\"));'>";
  $content.="<table width=100%>";
```

```
  $content.="<caption>Форма нового пользователя</caption>";
  $content.="<tr><td width=50%>Имя пользователя </td>
           <td width=50%><input type='text' name='name'
             size='20' maxlength='20'></td></tr>";
  ... ... ...
  // получить список шаблонов
  $query.="http://".$_SESSION[server]."/manager/ispmgr?";
  $query.="authinfo=".$_SESSION[login].":".$_SESSION[password]."&out=xml";
  $query.="&func=preset";
  // получить ответ  +  сформировать контент
  $xml = @file_get_contents($query); // получаем atom feed
  $data = @simplexml_load_string($xml); // разбираем atom
  $records=$data->xpath('elem');
  foreach ($records as $record)
  { $content.="<option name='".$record->name."'>".$record->name;  }
  $content.="</select></td></tr>";
  $content.="<tr><td width=50%>2. ИЛИ Пользовательские параметры
           <input type='radio' name='type' value='custom'>
           </td><td width=50%></td></tr>";
  $content.="<tr><td width=50%><b>Параметры</b></td>
           <td width=50%></td></tr>";
  ... ... ...
  $content.="<tr><td width=50%>FTP</td>
           <td width=50%><input type='text' name='ftplimit'
           size=20 maxlength='3' value='0'></td></tr>";
  $content.="<tr><td></td>
           <td><input type='submit' value='Создать' d</td></tr>";
  $content.="</table>";
  $content.="</form>";
  // вывод контента
  $objResponse->assign("view1","innerHTML",$content);
  // вывод запроса
  $objResponse->assign("view_command","innerHTML","");
  // вывод ответа сервера
  $objResponse->assign("reply_server","innerHTML","");
  return $objResponse;
}
?>
```

При нажатии кнопки **Создать** вызывается xAjax-функция `Add_User()`. Из полученных из формы данных функция формирует запрос на создание нового пользователя, отправляет запрос, получает ответ и выводит на страницу сам запрос и ответ сервера в формате XML (рис. 5.18). xAjax-функция `Add_User()` расположена в файле add_user.php, содержимое которого представлено в листинге 5.10.

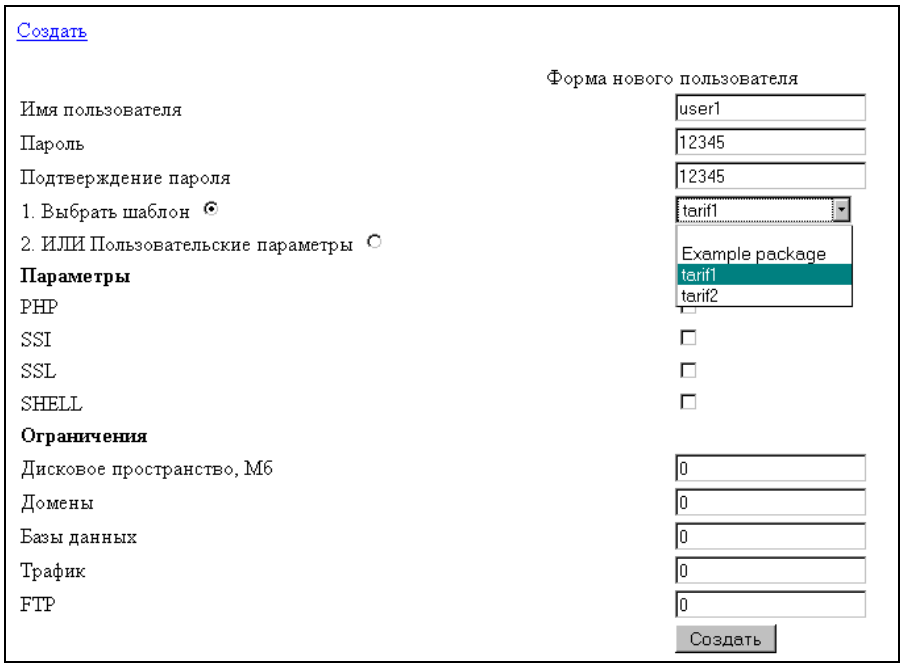


Рис. 5.17. Форма создания нового шаблона

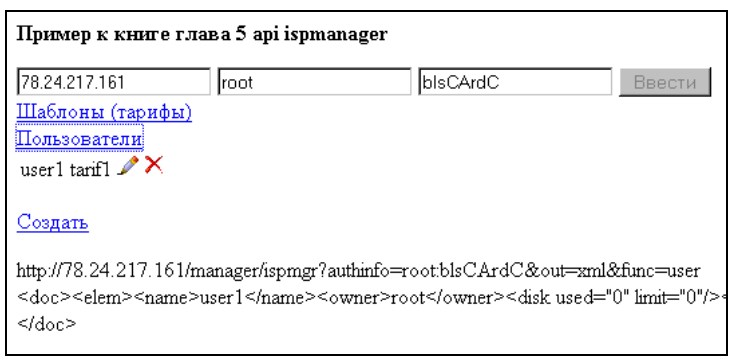


Рис. 5.18. Создание нового пользователя

Листинг 5.10

```
<?php
// Получение данных из формы и отправка команд создания нового пользователя
// или изменения параметров существующего
function Add_User($Id)
{ $objResponse = new xajaxResponse();
  // сформировать запрос
  $query.="http://".$_SESSION[server]."/manager/ispmgr?";
  $query.="authinfo=".$_SESSION[login].":".$_SESSION[password]."&out=xml";
  $query.="&func=user.edit&owner=".$_SESSION[login]."&sok=yes";
```

```
  if(isset($Id[elid])) $query.="&elid=".$Id[elid];
  // проверка параметров
  $query.="&name=".$Id[name];
  $query.="&passwd=".$Id[passwd];
  $query.="&confirm=".$Id[confirm];
  $query.="&preset=".str_replace(" ","%20",$Id[preset]);
  if(isset($Id[cgi])) $query.="&cgi=on";
  else $query.="&cgi=off";
  if(isset($Id[php])) $query.="&phpmod=on";
  else $query.="&php=off";
  if(isset($Id[ssi])) $query.="&ssi=on";
  else $query.="&ssi=off";
  if(isset($Id[ssl])) $query.="&ssl=on";
  else $query.="&ssl=off";
  if(isset($Id[shell])) $query.="&shell=on";
  else $query.="&shell=off";
  $query.="&disklimit=".$Id[disklimit];
  $query.="&domainlimit=".$Id[domainlimit];
  $query.="&baselimit=".$Id[baselimit];
  $query.="&bandwidthlimit=".$Id[bandwidthlimit];
  $query.="&ftplimit=".$Id[ftplimit];
  // отправка запроса
  $xml = @file_get_contents($query); // получаем atom feed
  $data = @simplexml_load_string($xml); // разбираем atom
  // очистить блок
  $objResponse->assign("view1","innerHTML","");
  // вывод запроса
  $objResponse->assign("view_command","innerHTML",$query);
  // вывод ответа сервера
  $xml=substr($xml,39,strlen($xml)-40);
  $script="var t1=document.getElementById('reply_server');";
  $script.="if(document.all) t1.innerText='".$xml."';";
  $script.="else t1.textContent='".$xml."';";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

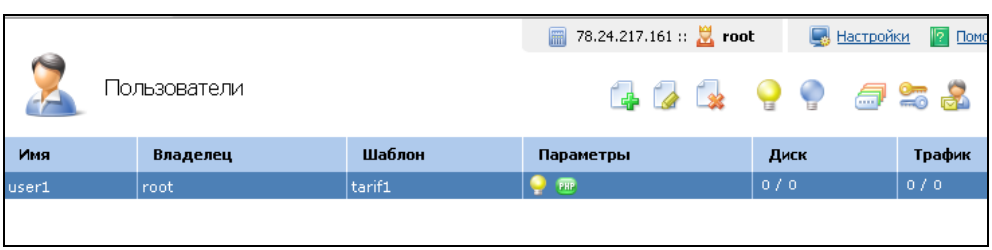


Рис. 5.19. Проверка добавления нового пользователя на сервере

Создадим пользователя с именем user1 и посмотрим, что оказалось на сервере (рис. 5.19). Пользователь добавлен.

5.2.9. Редактирование параметров пользователя

Для редактирования параметров пользователя щелкнем по значку редактирования. При этом вызывается xAjax-функция `Form_Edit_User()`. Функция создает запрос к API ISPmanager для получения параметров выбранного шаблона, получает с сервера параметры запрошенного шаблона, создает форму редактирования данных шаблона (рис. 5.20). Функция `Form_Edit_User()` расположена в файле form_edit_user.php, содержимое которого представлено в листинге 5.11.

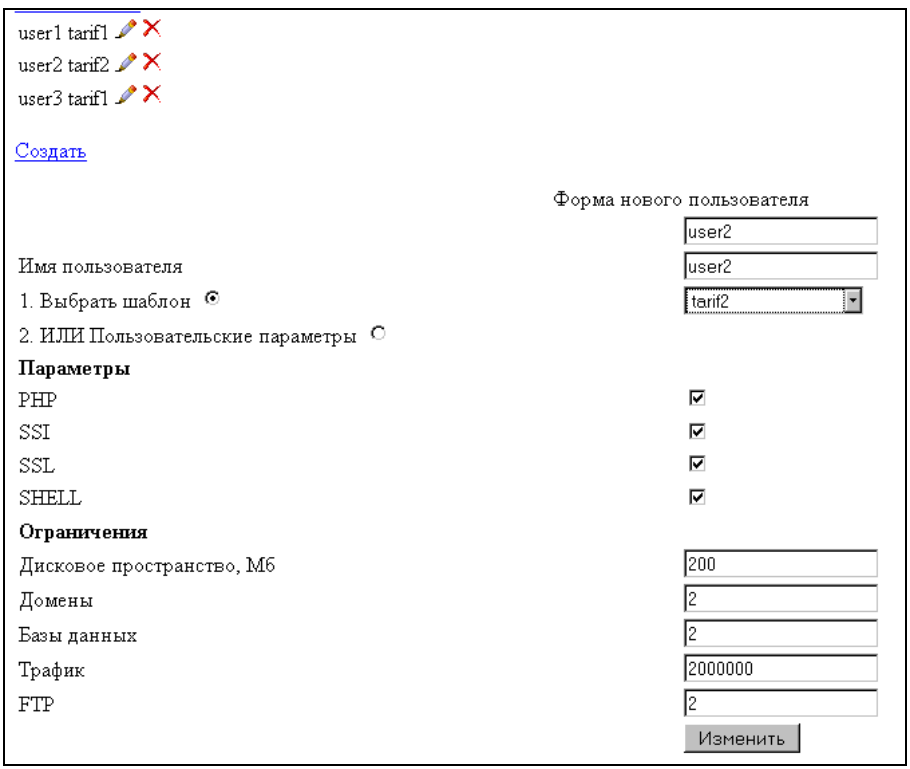


Рис. 5.20. Форма редактирования параметров пользователя

Листинг 5.11

```
<?php
// Форма редактирования параметров пользователя
function Form_Edit_User($name)
{ $objResponse = new xajaxResponse();
  // запрос о параметрах пользователя
  $query.="http://".$_SESSION[server]."/manager/ispmgr?";
```

```
$query.="authinfo=".$_SESSION[login].":".$_SESSION[password]."&out=xml";
$query.="&func=user.edit&elid=".$name;
// получить ответ  +  сформировать контент
$xml = @file_get_contents($query);   // получаем atom feed
$data = @simplexml_load_string($xml); // разбираем atom
// вывод контента
$content="<form id='FormAddUser' action='javascript:void(null);'
     onsubmit='xajax_Add_User(xajax.getFormValues(\"FormAddUser\"));'>";
$content.="<table width=100%>";
$content.="<caption>Форма нового пользователя</caption>";
$content.="<tr><td width=50%> </td>
         <td width=50%><input type='text' name='elid' value='".$name."'
          </td></tr>";
$content.="<tr><td width=50%>Имя пользователя </td>
         <td width=50%><input type='text' name='name'
         value='".$data->name."' size='20' maxlength='20'>
         </td></tr>";
$content.="<tr><td width=50%>1. Выбрать шаблон
         <input type='radio' name='type' value='preset' checked></td>
         <td width=50%><select name=preset><option value=''>";
// получить список шаблонов
$query="http://".$_SESSION[server]."/manager/ispmgr?";
$query.="authinfo=".$_SESSION[login].":".$_SESSION[password]."&out=xml";
$query.="&func=preset";
// получить ответ  +  сформировать контент
$xml1 = @file_get_contents($query);     // получаем atom feed
$data1 = @simplexml_load_string($xml1); // разбираем atom
$records=$data1->xpath('elem');
foreach ($records as $record)
{ if($data->preset==$record->name)
    $content.="<option name='".$record->name."' selected>".$record->name;
  else
    $content.="<option name='".$record->name."'>".$record->name;
}
$content.="</select></td></tr>"; /**/
$content.="<tr><td width=50%>2. ИЛИ Пользовательские параметры
         <input type='radio' name='type' value='custom'>
         </td><td width=50%></td></tr>";
$content.="<tr><td width=50%><b>Параметры</b></td>
         <td width=50%></td></tr>";
$content.="<tr><td width=50%>PHP</td>
         <td width=50%><input type='checkbox' name='php'
         ".(($data->phpmod=='on')?'checked':'').">
         </td></tr>";
... ... ...
$content.="<tr><td></td>
         <td><input type='submit' value='Изменить' d</td></tr>";
$content.="</table></form>";
```

```
  // вывод контента
  $objResponse->assign("view1","innerHTML",$content);
  // вывод запроса
  $objResponse->assign("view_command","innerHTML","");
  // вывод ответа сервера
  $objResponse->assign("reply_server","innerHTML","");
  return $objResponse;
}
?>
```

Изменим данные выбранного шаблона (рис. 5.10). При нажатии кнопки **Изменить** вызывается xAjax-функция `Add_User()`, рассмотренная нами в *разд. 5.2.8*. Результат выполнения функции представлен на рис. 5.21. Проверим новые параметры шаблона на сервере (рис. 5.22).

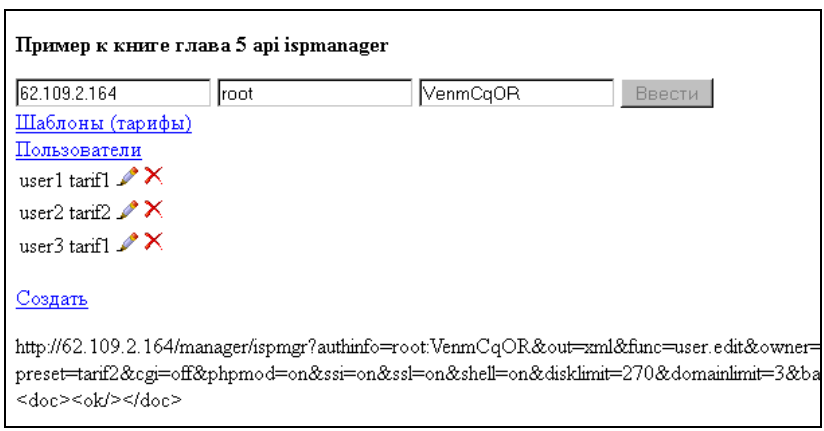


Рис. 5.21. Результат выполнения запроса

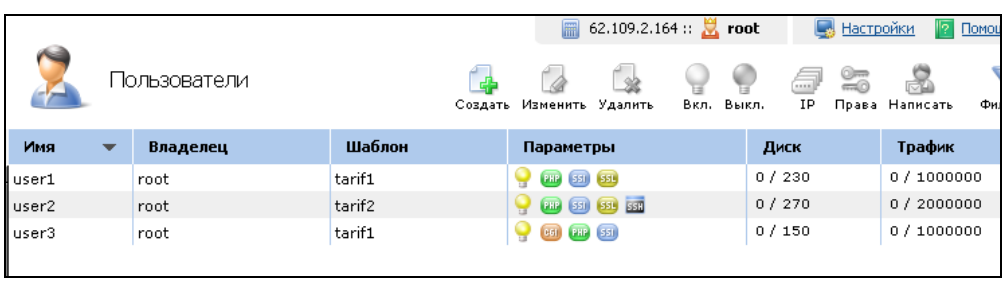


Рис. 5.22. Проверка редактирования шаблона на сервере

5.2.10. Удаление пользователя

Для удаления пользователя щелкнем по значку удаления. Вызывается xAjax-функция `Delete_User()`. Она создает запрос к API ISPmanager для удаления выбранного шаблона. Функция `Delete_User()` расположена в файле delete_user.php, содержимое которого представлено в листинге 5.12.

Листинг 5.12

```
<?php
// Отправка команд удаления пользователя
function Delete_User($elid)
{ $objResponse = new xajaxResponse();
  // сформировать запрос
  $query.="http://".$_SESSION[server]."/manager/ispmgr?";
  $query.="authinfo=".$_SESSION[login].":".$_SESSION[password]. "&out=xml";
  $query.="&func=user.delete&elid=".$elid;
  // отправка запроса и получение ответа
  $xml = @file_get_contents($query);    // получаем atom feed
  $data = @simplexml_load_string($xml); // разбираем atom
  // очистить блок
  $objResponse->assign("view1","innerHTML","");
  // вывод запроса
  $objResponse->assign("view_command","innerHTML",$query);
  // вывод запроса
  $objResponse->assign("view_command","innerHTML",$query);
  // вывод ответа сервера
  $xml=substr($xml,39,strlen($xml)-40);
  $script="var t1=document.getElementById('reply_server');";
  $script.="if(document.all) t1.innerText='".$xml."';";
  $script.="else t1.textContent='".$xml."';";
  $objResponse->script($script);
  return $objResponse;
}
?>
```

Результат выполнения функции представлен на рис. 5.23, получаем список шаблонов после удаления (рис. 5.24), проверяем новые параметры шаблона на сервере (рис. 5.25).

Мы рассмотрели малую часть возможностей API ISPManager, но, думаю, вы получили практический опыт составления запросов.

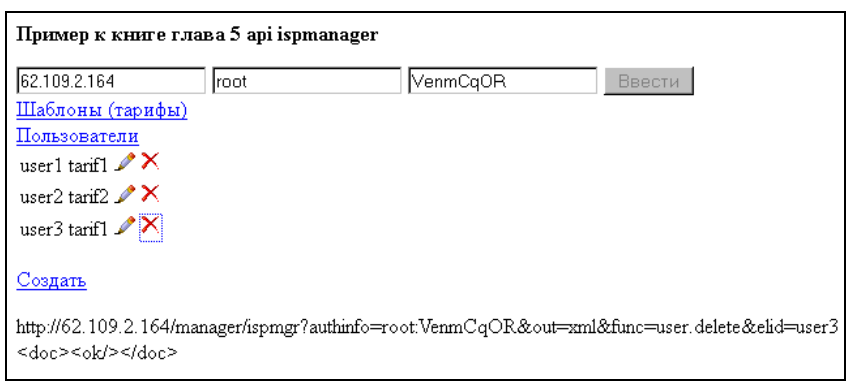


Рис. 5.23. Результат выполнения запроса

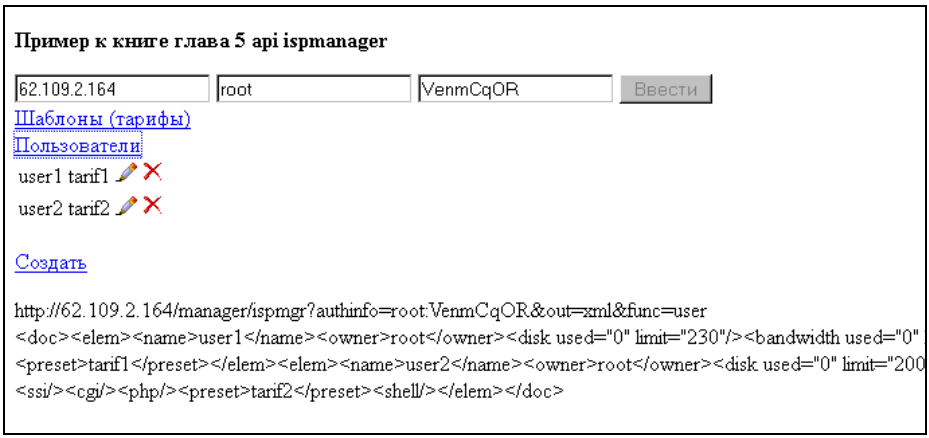


Рис. 5.24. Проверка выполнения запроса на сервере

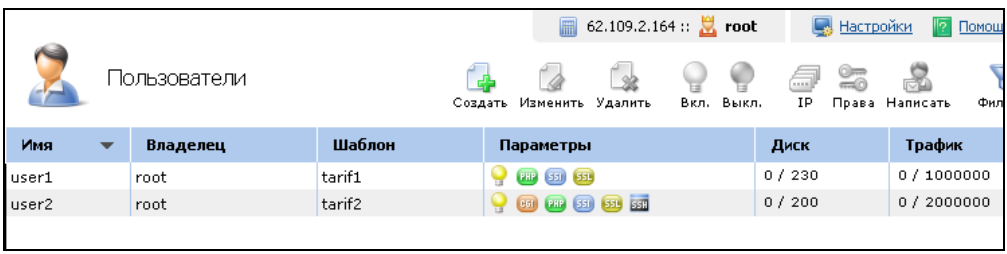


Рис. 5.25. Список шаблонов после удаления

ГЛАВА 6

Создание личного кабинета для сайта хостинговой компании

Создадим сайт личного кабинета пользователя хостинговой компании. Часть проекта будем использовать как демонстрацию API ISPmanager. Сам проект создадим в полном объеме. Его вы можете использовать на вашем хостинге.

6.1. Необходимый функционал сайта

Нашему будущему сайту необходим следующий функционал:

- форма авторизации для входа в личный кабинет;
- форма восстановления пароля на e-mail;
- форма регистрации пользователей (для юридических или физических лиц);
- просмотр, заказ существующих тарифов;
- дополнительные услуги для тарифов;
- редактирование заказанных тарифов;
- выбор будущего тарифа;
- оплата услуг (квитанция, счет, электронные деньги);
- ежедневное списывание средств;
- формирование ежедневных отчетов.

Для админского профиля создадим следующий функционал:

- просмотр, редактирование профилей пользователей, установка VIP-статуса;
- добавление новых пользователей;
- поиск пользователей;
- поиск, просмотр счетов пользователей;
- просмотр и редактирование тарифов пользователя;
- отчеты по пользователю.

6.2. Проектирование баз данных

Для полноценного функционирования сайта необходимо создать следующие таблицы в базе данных:

- `my_users` — пользователи;
- `my_users_info2` — данные о пользователе — юридическом лице;
- `my_users_info3` — данные о пользователе — физическом лице;
- `my_l_schet` — лицевые счета клиентов;
- `my_tarifs` — тарифные планы и дополнительные услуги к тарифным планам;
- `my_schets` — счета;
- `my_history` — история событий с профилем пользователя (изменение тарифного плана, списывание средств, заказ дополнительных услуг).

Рассмотрим подробно структуру и назначение вышеприведенных таблиц.

Таблица `my_users` содержит информацию о пользователях и их тарифных планах и дополнительных услугах к тарифным планам. Структура таблицы следующая:

- `id` — первичный ключ;
- `type` — тип пользователя:
  - `2` — юридическое лицо;
  - `3` — физическое лицо;
  - `4` — юридическое лицо (статус VIP);
  - `5` — физическое лицо (статус VIP);
  - `8` — администратор;
- `type2` — тип пользователя (реселлер, хостинг):
  - `1` — хостинг;
  - `2` — реселлер.
- `id_info` — идентификатор поля, где хранятся анкетные данные, для физических лиц — таблица `my_users_info3`, для юридических — `my_users_info2`;
- `id_l_schet` — идентификатор лицевого счета;
- `password` — пароль пользователя;
- `id_tarif` — идентификатор действующего тарифного плана;
- `schet_id_tarif` — идентификатор счета, по которому заказан действующий тарифный план;
- `days_id_tarif` — количество дней действующего тарифного плана;
- `uslugi_id_tarif` — дополнительные услуги действующего тарифного плана;
- `id_tarif_sled` — идентификатор следующего тарифного плана;

- `schet_id_tarif_sled` — идентификатор счета, по которому заказан следующий тарифный план;
- `days_id_tarif _sled` — количество дней следующего тарифного плана:
- `uslugi_id_tarif_sled` — дополнительные услуги следующего тарифного плана;
- `oplata_podkl` — возможность единовременного бесплатного пробного периода:
  - `yes` — был использован пробный период;
  - `no` — не был использован пробный период.
- `oplata_period` — т. к. будет возможность отсрочки платежа для некоторых клиентов, введем поле фактической оплаты действующего тарифа:
  - `yes` — оплачено;
  - `no` — не оплачено.
- `data_start` — дата начала действия действующего тарифного плана;
- `data_end` — дата окончания действия действующего тарифного плана;
- `data_reg` — дата регистрации пользователя;
- `visible` — статус пользователя:
  - `yes` — активен;
  - `no` — отключен.

В таблице `my_users` будут храниться сведения о заказанных пользователем тарифных планах. При оплате счета информация будет заноситься в поля `id_tarif`, `schet_id_tarif`, `days_id_tarif`, `uslugi_id_tarif` или `id_tarif_sled`, `schet_id_tarif_sled`, `days_id_tarif_sled`, `uslugi_id_tarif_sled`, если мы оплачиваем следующий тарифный план. Также для текущего тарифного плана заполняем поля `data_start` и `data_end`. Тарифный план пользователь может изменить в любой момент. При этом будем производить пересчет данных для поля `data_end`. В день, следующий за днем окончания действия тарифного плана, поле `id_tarif` либо обнуляется и пользователь будет отключаться, либо происходит перенос значений следующего тарифного плана и его активация.

Дамп для создания таблицы `my_users` представлен в листинге 6.1.

Листинг 6.1

```
CREATE TABLE `my_users` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`type` set( '2', '3', '4', '5', '8') default NULL,
`type2` set('1', '2', '8') default '1',
`id_info` int(11) NOT NULL,
`id_l_schet` int(11) default NULL,
`password` varchar(10) NOT NULL,
`id_tarif` int(11) default '0',
`schet_id_tarif` int(9) NOT NULL,
`days_id_tarif` int(3) default '0',
```

```
`uslugi_id_tarif` tinytext NOT NULL,
`id_tarif_sled` int(11) NOT NULL default '0',
`schet_id_tarif_sled` int(9) NOT NULL,
`days_id_tarif_sled` int(3) NOT NULL,
`uslugi_id_tarif_sled` tinytext NOT NULL,
`oplata_podkl` set('no', 'yes') default 'no',
`oplata_period` set('no', 'yes') NOT NULL default 'no',
`data_start` datetime NOT NULL,
`data_end` datetime NOT NULL,
`visible` set('yes', 'no') NOT NULL default 'yes',
`data_reg` datetime NOT NULL,
UNIQUE KEY `id` (`id`),
KEY `data_reg` (`data_reg`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

Анкетные данные для физических и юридических лиц очень различаются, поэтому создадим две разные таблицы для хранения анкетных данных. Таблица `my_users_info2` хранит анкетные данные юридических лиц. Структура этой таблицы такова:

- `id` — первичный ключ;
- `name` — наименование организации;
- `contact_person` — ФИО контактного лица;
- `email` — адрес электронной почты;
- `phones` — контактные телефоны;
- `fax` — факс;
- `inn` — ИНН организации;
- `kpp` — КПП организации;
- `schet` — номер расчетного счета;
- `bank` — наименование банка;
- `bank_bik` — БИК банка;
- `bank_ks` — корреспондентский счет;
- `address` — юридический адрес;
- `mail_address` — почтовый адрес.

Набор полей, конечно, велик, но они все нам пригодятся для автоматического формирования формы счета к оплате. Дамп для создания таблицы `my_users_info2` представлен в листинге 6.2.

Листинг 6.2

```
CREATE TABLE `my_users_info2` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`name` varchar(50) NOT NULL,
```

```
`contact_person` varchar(30) default NULL,
`email` varchar(30) NOT NULL,
`phones` varchar(30) NOT NULL,
`fax` varchar(30) NOT NULL,
`inn` varchar(10) NOT NULL,
`kpp` varchar(10) NOT NULL,
`bank` varchar(50) NOT NULL,
`bank_bik` varchar(10) NOT NULL,
`bank_ks` varchar(20) NOT NULL,
`schet` varchar(20) NOT NULL,
`address` varchar(100) default NULL,
`mail_address` varchar(100) NOT NULL,
UNIQUE KEY `id` (`id`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

Для физических лиц набор анкетных данных меньше. Таблица `my_users_info3` хранит анкетные данные физических лиц. Структура этой таблицы такова:

- `id` — первичный ключ;
- `name1` — фамилия пользователя;
- `name2` — имя пользователя;
- `name3` — отчество пользователя;
- `email` — адрес электронной почты;
- `phone` — контактный телефон;
- `address` — адрес;
- `passport_number` — серия и номер паспорта;
- `passport_place` — кем выдан паспорт;
- `passport_data` — дата выдычи паспорта.

Дамп для создания таблицы `my_users_info3` представлен в листинге 6.3.

Листинг 6.3

```
CREATE TABLE `my_users_info3` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`name1` varchar(30) NOT NULL,
`name2` varchar(30) NOT NULL,
`name3` varchar(30) NOT NULL,
`email` varchar(30) NOT NULL,
`phone` varchar(30) NOT NULL,
`address` varchar(100) NOT NULL,
`pasport_number` varchar(15) NOT NULL,
`pasport_place` varchar(50) NOT NULL,
`pasport_data` date NOT NULL,
UNIQUE KEY `id` (`id`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

Таблица my_l_schet содержит информацию о лицевых счетах клиентов. Каждому id клиента будем ставить в соответствие лицевой счет. Структура таблицы состоит из двух полей:

- id — первичный ключ;
- name — лицевой счет.

Дамп для создания таблицы my_l_schet представлен в листинге 6.4.

Листинг 6.4

```
CREATE TABLE `my_l_schet` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`name` varchar(10) NOT NULL,
UNIQUE KEY `id` (`id`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

Таблица my_tarifs содержит информацию о тарифных планах и дополнительных услугах к тарифным планам. Структура таблицы my_tarifs следующая:

- id — первичный ключ;
- id_tarif — для тарифов — 0, для дополнительных услуг — идентификатор тарифного плана, к которой привязана услуга;
- type_users — тип пользователей, которые могут выбирать данный тарифный план;
- type — тип хостинг/реселлер;
- sort — поле сортировки;
- name — название тарифного плана;
- info — информация о тарифном плане;
- pay1 — данные по стоимости услуги;
- pay2 — данные по стоимости услуги за день;
- activ — активность профиля:
  - yes — профиль активен;
  - no — профиль отключен;
- oplata — способы оплаты.

Для физических и юридических лиц, для VIP-клиентов, а также для типов профилей пользователей (хостинг, реселлер) существует свой набор тарифных планов, некоторые тарифные планы существуют для нескольких типов пользователей. Поле type_users — список типов пользователей, имеющих возможность заказать данный тариф. Поле oplata определяет список способов оплаты, доступных пользователям, заказывающих этот тариф. Для тарифов значение поля id_tarif равно 0. Для до-

полнительных услуг это поле осуществляет привязку дополнительной услуги и тарифному плану. Для каждого тарифного плана можно определить свой набор дополнительных услуг.

Дамп для создания таблицы `my_tarifs` представлен в листинге 6.5.

Листинг 6.5

```
CREATE TABLE `my_tarifs` (
`id` int(9) NOT NULL AUTO_INCREMENT,
`id_tarif` varchar(100) NOT NULL default '0',
`type_users` varchar(10) default NULL,
`type` int(1) NOT NULL,
`sort` int(2) NOT NULL,
`name` varchar(50) default NULL,
`info` varchar(100) default NULL,
`pay1` varchar(20) default NULL,
`activ` set('yes', 'no') NOT NULL,
`oplata` varchar(100) default NULL,
UNIQUE KEY `id` (`id`),
KEY `sort` (`sort`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

Таблица `my_schets` содержит информацию о счетах пользователей. Структура этой таблицы такова:

- `id` — первичный ключ;
- `number` — номер счета;
- `id_users` — пользователь — создатель счета;
- `id_tarif` — идентификатор выбранного тарифного плана;
- `days_id_tarif` — количество дней;
- `dop_uslugi` — заказанные дополнительные услуги;
- `summa` — сумма по выбранному плану;
- `summa_oplata` — оплаченная сумма;
- `info` — информация по заказанному тарифному плану;
- `oplata` — статус оплаты счета:
  - `yes` — оплачен;
  - `no` — не оплачен;
  - `del` — удален.
- `data` — дата создания счета;

- `data_oplata` — дата оплаты;
- `info_oplata` — информация по оплате.

Счета имеют статус "оплачен", "не оплачен" либо "удален". При оплате счета информация из него переносится в таблицу `my_users` в поля для текущего либо будущего тарифного плана. Дамп для создания таблицы `my_schets` представлен в листинге 6.6.

Листинг 6.6

```
CREATE TABLE `my_schets` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`number` varchar(10) NOT NULL,
`id_users` int(11) NOT NULL,
`id_tarif` int(11) NOT NULL,
`days_id_tarif` int(3) default '0',
`dop_uslugi` tinytext NOT NULL,
`summa` float(10, 2) default NULL,
`summa_oplata` float(10, 2) default NULL,
`info` tinytext NOT NULL,
`oplata` set('yes', 'no', 'del') default 'no',
`data` datetime NOT NULL,
`data_oplata` datetime NOT NULL,
`info_oplata` tinytext NOT NULL,
UNIQUE KEY `id` (`id`),
KEY `data` (`data`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

В функционал сайта будет включена возможность смены действующего тарифного плана и набора дополнительных услуг в любой момент, без внесения дополнительных средств. При этом происходит пересчет срока действия тарифного плана. Все эти изменения, а также каждодневная история списывания средств за тарифный план и дополнительные услуги будут храниться в таблице `my_history`. Это позволит иметь полный журнал событий с тарифными планами для каждого пользователя. Структура таблицы `my_history` следующая:

- `id` — первичный ключ;
- `id_user` — идентификатор пользователя;
- `id_schet` — идентификатор счета для данного тарифного плана;
- `id_tarif` — идентификатор тарифного плана;
- `dop_uslugi` — заказанные дополнительные услуги;
- `days` — остаток дней действия;
- `data_start` — дата начала тарифного плана;
- `data_end` — дата окончания тарифного плана;

- `visible` — статус данного тарифного плана для пользователя:
  - `yes` — действующий;
  - `no` — не действует (будущего периода);
  - `end` — закончился;
- `data` — дата и время создания данной записи (для изменения тарифного плана — дата изменения, для снятия суммы за услуги — дата и время выполнения программы по cron);
- `summa` — сумма;
- `info` — информация (например, ежедневное снятие комиссии по действующему тарифному плану).

Записи о ежедневном снятии комиссии за пользование тарифным планом с формированием остатка средств и количества дней до окончания тарифа необходимо производить запуском скрипта по cron. Дамп для создания таблицы `my_history` представлен в листинге 6.7.

Листинг 6.7

```
CREATE TABLE `my_history` (
`id` int(9) NOT NULL AUTO_INCREMENT,
`id_user` int(9) NOT NULL,
`id_schet` int(9) NOT NULL,
`id_tarif` varchar(9) NOT NULL,
`dop_uslugi` tinytext NOT NULL,
`days` int(9) NOT NULL,
`data_end` datetime NOT NULL,
`data_start` datetime NOT NULL,
`visible` set('yes', 'no', 'end') default NULL,
`data` datetime NOT NULL,
`summa` float(15, 6) NOT NULL,
`info` varchar(30) NOT NULL,
UNIQUE KEY `id` (`id`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251
```

Дамп базы данных с заполнением таблиц находится на прилагаемом компакт-диске (файл glava_06\hosting.sql). Мы рассмотрели и создали необходимые таблицы в базе данных, теперь приступим к программированию сайта.

6.3. Главная страница

Главная страница личного кабинета — это форма входа в личный кабинет (рис. 6.1). Так как мы создаем сайт на Ajax без перезагрузки страницы, всю разбивку и все стили необходимо определить для этой страницы (файл index.php). Фрагмент файла index.php представлен в листинге 6.8. Внешний файл стилей можно посмотреть на прилагаемом к книге компакт-диске (hosting\styles.css).

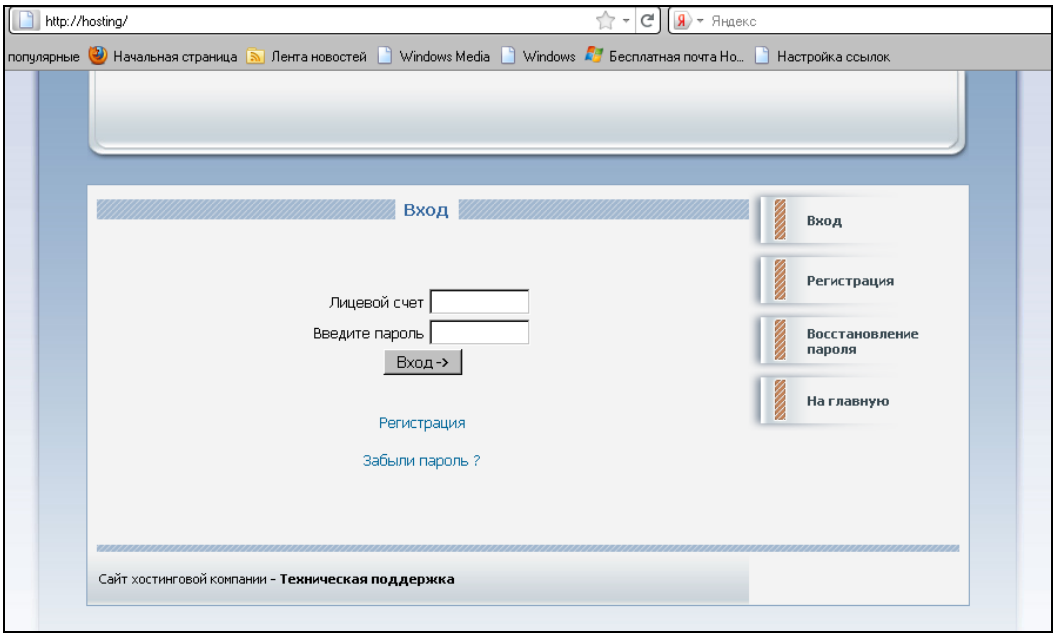


Рис 6.1. Форма входа в личный кабинет

Листинг 6.8

```
<?php
// подключение библиотеки xAjax, подключение к базе данных
... ... ...
?>
<html>
<head>
... ... ...
<script type="text/javascript" src="epoch_v106_ru/epoch_classes.js">
  var calendar1;var calendar2;
</script>
<?php $xajax->printJavascript(''); ?>
</head>
<body>
      ... ... ...
      <div id="center2">
      <form name='Form_Vhod' id='Form_Vhod' action='javascript:void(n);'
          onsubmit='xajax.$("Button_Form_Vhod").disabled=true;
          xajax.$("Button_Form_Vhod").value="Подождите...";
          xajax_Vhod(xajax.getFormValues("Form_Vhod"));'>
       ... ... ...
       <input type='text' id='l_schet' name='l_schet' value='' size=10
       maxlength=10>
       ... ... ...
```

```
      <input type='password' id='password' name='password' value=''
             size=10 maxlength=10>
      ... ... ...
      <input type='submit' id='Button_Form_Vhod' value='Вход ->'>
    </form>
    <div id='div_vhod'></div>
    <a href='javascript:void();' onclick='xajax_Form_Reg1();'>
      Регистрация</a>
      <a href='javascript:void();' onclick='xajax_Form_Password_Mail();'
      >Забыли пароль?</a>
    </div>
    <div id="center3">
    </div>
    <div id="center4">
    </div>
    <div id="center5">
    </div>
    <div id="center6">
    </div>
   ... ... ...
   <div id="submenu">
   <!-- Start Submenu -->
      ... ... ...
      <a href="javascript:void();" onclick="xajax_Form_Vhod();"
      class="linksubmenu">
      Вход</a>
      ... ... ...
      <a href="javascript:void();" onclick="xajax_Form_Reg1();"
      class="linksubmenu">
      Регистрация</a>
      ... ... ...
      <a href="javascript:void();" onclick="xajax_Form_Password_Mail();"
      class="linksubmenu">
      Восстановление<br>пароля</a>
      ... ... ...
      <a href="http://hosting.netfast.ru" class="linksubmenu">
      На главную</a>
      ... ... ...
   </div>
</body>
</html>
```

6.4. Регистрация пользователей

При щелчке по ссылке **Регистрация** переходим к регистрации. Регистрация на таких сайтах проходит в несколько шагов. На первом шаге необходимо согласиться с условиями предоставления услуг (рис. 6.2). xAjax-функция `Form_Reg1()` выводит

текст соглашения (находится в файле hosting\dogovor.txt) и открывает возможность перехода к следующему шагу только после подтверждения согласия с правилами. Эта функция находится в файле hosting\ my_prg_reg\form_reg1.php, содержимое которого представлено в листинге 6.9.

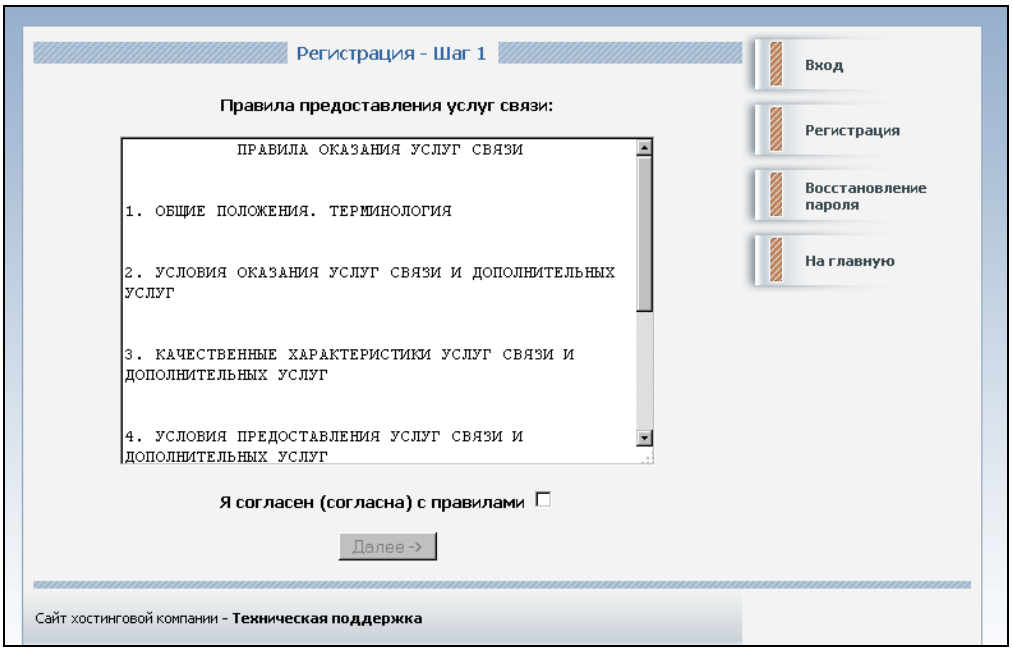


Рис 6.2. Форма подтверждения с ознакомлением с правилами предоставления услуг

Листинг 6.9

```
<?php
// Регистрация пользователя ШАГ 1.
// Соглашение с условиями договора
function Form_Reg1()
{ $objResponse = new xajaxResponse();
  // получить текст договора
  $text2=file_get_contents("dogovor.txt",1);
  // формирование контента
  $text1="<center><br><b>Правила предоставления услуг связи:<b><br><br>";
  $text1.="<form id='Form_Reg1' action='javascript:void(null);'
      onsubmit='xajax.$(\"Button_Form_Reg1\").disabled=true;
      xajax.$(\"Button_Form_Reg1\").value=\"Подождите...\";
      var x=new Array();x[0]=1;x[1]=2;xajax_Form_Reg21(x);'>";
  $text1.="<textarea cols=50 rows=15 value='".$text2."' readonly>
          ".$text2."</textarea>";
  $text1.="<br><br>Я согласен (согласна) с правилами
          <input type='checkbox' onclick='if(this.checked)
```

```
            {document.getElementById(
               \"Button_Form_Reg1\").disabled=false;}
            else {document.getElementById(
               \"Button_Form_Reg1\").disabled=true;}'>";
  // дезактивировать кнопку
  $text1.="<br><br><input type='submit' id='Button_Form_Reg1'
            value='Далее ->' disabled=true>";
  $text1.="</form>";
  // вывод контента
  $objResponse->assign("zag1","innerHTML",
    "<font class='blocktitle'> Регистрация — Шаг 1 </font>");
  $objResponse->assign("center1","innerHTML","<table></table>");
  $objResponse->assign("center2","innerHTML",$text1);
  $objResponse->assign("center3","innerHTML","<table></table>");
  $objResponse->assign("center4","innerHTML","<table></table>");
  $objResponse->assign("center5","innerHTML","<table></table>");
  $objResponse->assign("center6","innerHTML","<table></table>");
  return $objResponse;
}
?>
```

Следующий шаг — выбор реселлер/хостинг и далее юридическое/физическое лицо (рис. 6.3 и 6.4).

xAjax-функции `Form_Reg21()` и `Form_Reg22()` находятся в файле hosting\my_prg_reg\form_reg2.php, содержимое которого представлено в листинге 6.10.

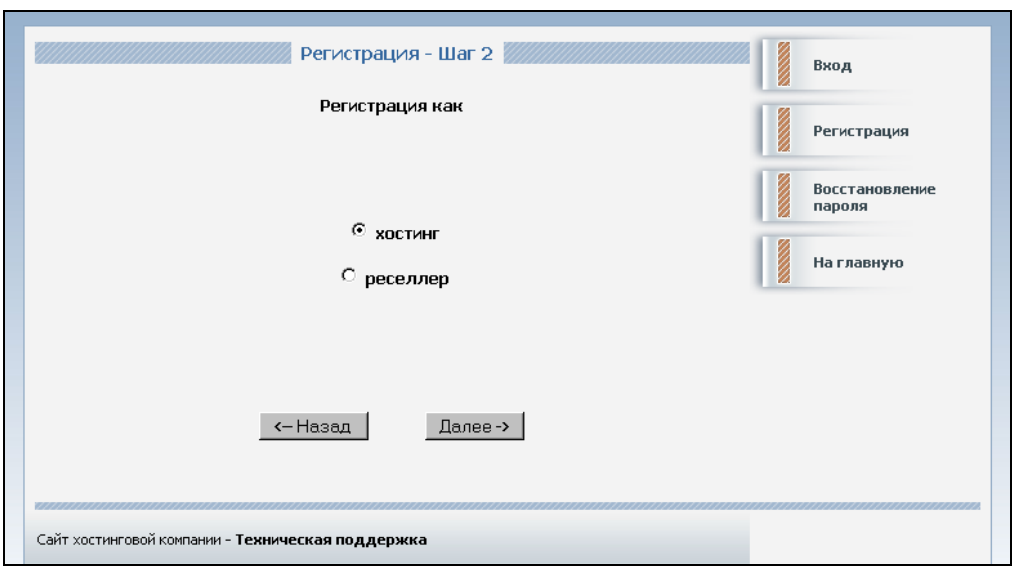


Рис 6.3. Выбор: хостинг/реселлер

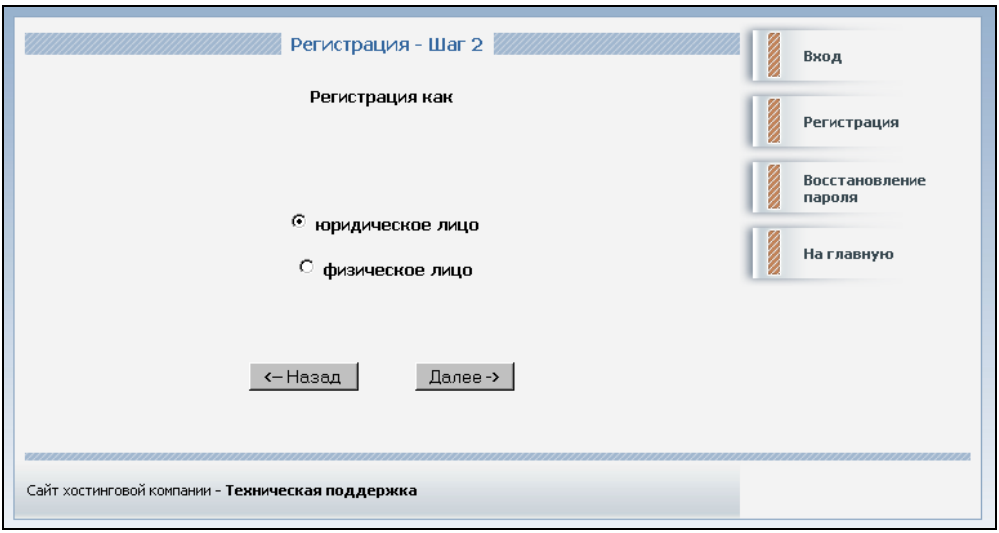


Рис 6.4. Выбор: юридическое/физическое лицо

Листинг 6.10

```
<?php
// Регистрация пользователя ШАГ 2-1.
// Выбор: реселлер/хостинг
function Form_Reg21($Id)
{ $objResponse = new xajaxResponse();
  // формирование кода формы
  $text1="<center><br><b>Регистрация как<b><br><br><br><br>";
  $text1.="<form id='Form_Reg2' action='javascript:void(null);'
         onsubmit='xajax.$(\"Button_Form_Reg2\").disabled=true;
         xajax.$(\"Button_Form_Reg2\").value=\"Подождите...\";
         xajax_Form_Reg22(xajax.getFormValues(\"Form_Reg2\"));'>";
  if($Id[type2]==2)
    {$checked1="";$checked2="checked";}
  else
    {$checked1="checked";$checked2="";}
  $text1.="<br><br><input name=type2 type='radio' value='1'
         ".$checked1.">хостинг ";
  $text1.="<br><br><input name=type2 type='radio' value='2'
         ".$checked2.">реселлер ";
  if(!isset($Id[type]))
     $Id[type]=$Id[1];
  // скрытое поле передачи для возвращения к предыдущему выбору
  $text1.="<br><br><input name=type type='hidden'
      value='".$Id[type]."'>";
  $text1.="<br><br><br><br><br><input type='button' onclick='
          xajax_Form_Reg1();' value='<-- Назад ' >";
```

```
  $text1.="<input type='submit' id='Button_Form_Reg2' value='Далее ->'>
           <br><br><br></form>";
  // выдача контента в блоки
  $objResponse->assign("zag1","innerHTML",
  "<font class='blocktitle'> Регистрация — Шаг 2 </font>");
  $objResponse->assign("center1","innerHTML","<table></table>");
  $objResponse->assign("center2","innerHTML",$text1);
  $objResponse->assign("center3","innerHTML","<table></table>");
  $objResponse->assign("center4","innerHTML","<table></table>");
  $objResponse->assign("center5","innerHTML","<table></table>");
  $objResponse->assign("center6","innerHTML","<table></table>");
  return $objResponse;
}
// Регистрация пользователя ШАГ 2-2.
// Выбор: физическое лицо/юридическое лицо
function Form_Reg22($Id)
{ $objResponse = new xajaxResponse();
  $text1="<center><br><b>Регистрация как<b><br><br><br><br>";
  $text1.="<form id='Form_Reg2' action='javascript:void(null);'
     onsubmit='xajax.$(\"Button_Form_Reg2\").disabled=true;
     xajax.$(\"Button_Form_Reg2\").value=\"Подождите...\";
     xajax_Form_Reg3(xajax.getFormValues(\"Form_Reg2\"));'>";
  $text1.="<input name=type2 type='hidden' value='".$Id[type2]."'>";
  if($Id[type]==2)
    {$checked2="checked";$checked3="";}
  else
    {$checked2="";$checked3="checked";}
  $text1.="<br><br><input name=type type='radio' value='2' ".$checked2.">
           юридическое лицо";
  $text1.="<br><br><input name=type type='radio' value='3' ".$checked3.">
            физическое лицо";
  $text1.="<br><br><br><br><br><input type='button'
           onclick='xajax_Form_Reg21(xajax.getFormValues(\"Form_Reg2\"));'
            value='<-- Назад ' >";
  $text1.=" <input type='submit' id='Button_Form_Reg2' value='Далее ->'>
           <br><br><br></form>";
  // выдача контента в блоки
  $objResponse->assign("zag1","innerHTML",
  "<font class='blocktitle'> Регистрация — Шаг 2 </font>");
  $objResponse->assign("center1","innerHTML","<table></table>");
  $objResponse->assign("center2","innerHTML",$text1);
  $objResponse->assign("center3","innerHTML","<table></table>");
  $objResponse->assign("center4","innerHTML","<table></table>");
  $objResponse->assign("center5","innerHTML","<table></table>");
  $objResponse->assign("center6","innerHTML","<table></table>");
  return $objResponse;
}
?>
```

На шаге 3 мы попадаем в форму заполнения анкетных данных физического (рис. 6.5) или юридического лица. xAjax-функция `Reg3()` выполняет перенаправление на выбор нужных функций (листинг 6.11, файл hosting\my_prg_reg\form_reg3.php). Рассмотрим функции для шага 3 и регистрации пользователя — физического лица. Есть набор полей, обязательных для заполнения. Для физического лица это фамилия, имя и адрес электронной почты. При изменении значений этих полей,

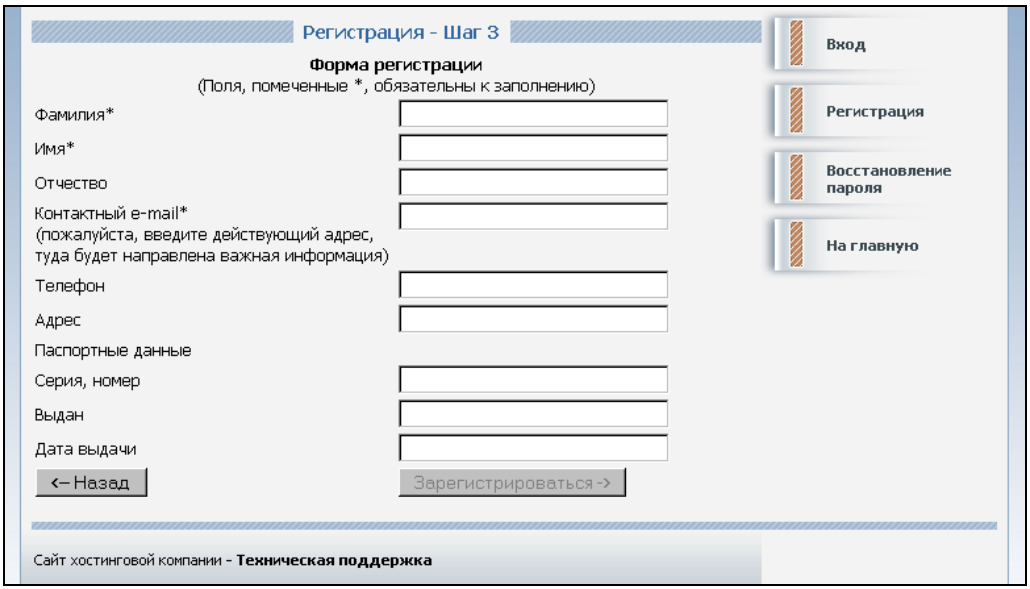


Рис 6.5. Форма заполнения анкетных данных для физического лица

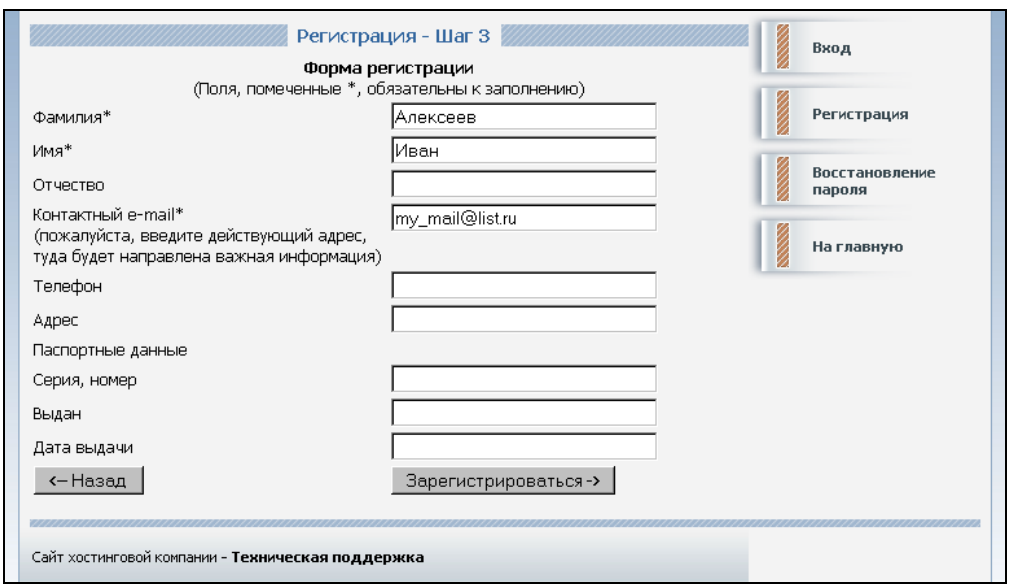


Рис. 6.6. Правильное заполнение обязательных полей. Активация кнопки регистрации

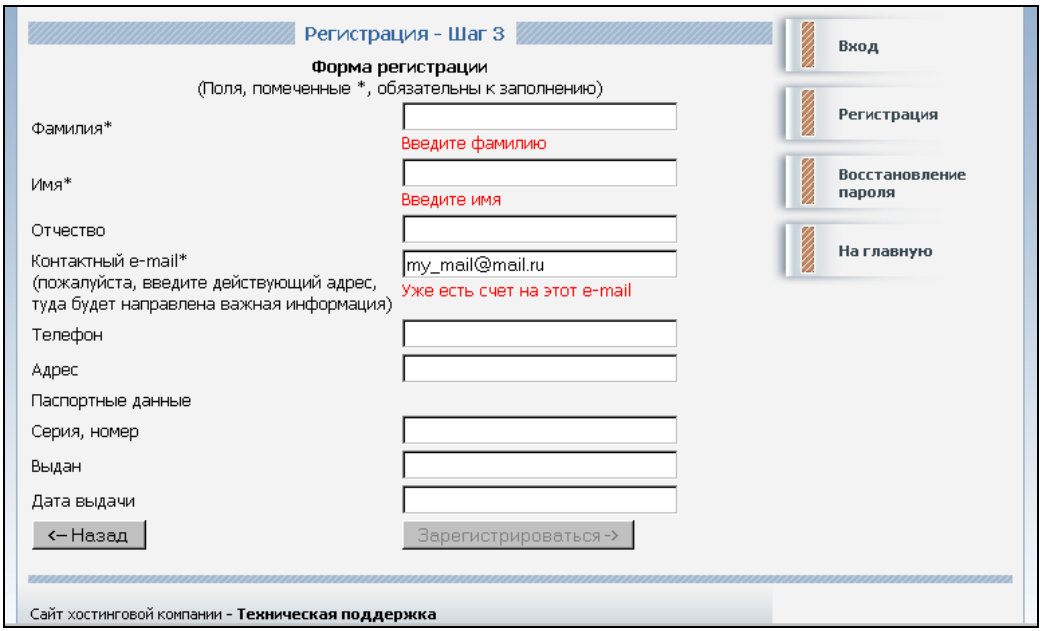


Рис. 6.7. Сообщения при неверном заполнении данных

по событию `onchange`, вызывается xAjax-функция `Control_Reg33()`, которая проверяет правильность заполнения обязательных полей (для адреса электронной почты — уникальность). При правильном заполнении обязательных полей активируется кнопка **Зарегестрироваться** (рис. 6.6), при неправильном — выводятся сообщения об ошибке (рис. 6.7).

Листинг 6.11

```
<?php
// Регистрация пользователя ШАГ 3.
// Анкета
function Form_Reg3($Id)
{ $objResponse = new xajaxResponse();
  $text1="";
  // переход на формирование нужной анкеты
  if($Id[type]==2)  $text1.=function_form_reg32($Id);
  else              $text1.=function_form_reg33($Id);
  // вывод контента
  $objResponse->assign("zag1","innerHTML",
  "<font class='blocktitle'> Регистрация – Шаг 3 </font>");
  $objResponse->assign("center1","innerHTML","<table></table>");
  $objResponse->assign("center2","innerHTML",$text1);
  $objResponse->assign("center3","innerHTML","<table></table>");
  $objResponse->assign("center4","innerHTML","<table></table>");
```

```
  $objResponse->assign("center5","innerHTML","<table></table>");
  $objResponse->assign("center6","innerHTML","<table></table>");
  // календарь
  $script="calendar1=new Epoch('epoch_popup','popup',
           document.getElementById('pasport_data'));";
  if($Id[type]==3) $objResponse->script($script);
  return $objResponse;
}
?>
```

За формирование контента отвечает функция из файла hosting\my_prg_reg\function_form_reg33.php, содержимое которого представлено в листинге 6.12.

Листинг 6.12

```
<?php
// Вывод формы регистрации пользователя — физического лица
function function_form_reg33($Id)
{ require_once("my.php");
  $text1="<center>";
  $text1.="<form id='Form_Reg33' action='javascript:void(null);'
    onsubmit='xajax.$(\"Button_Form_Reg33\").disabled=true;
    xajax.$(\"Button_Form_Reg33\").value=\"Подождите...\";
    xajax_Reg_User33(xajax.getFormValues(\"Form_Reg33\"));'>";
  $text1.="<table width=100%>";
  $text1.="<caption><b>Форма регистрации </b><br>
           (Поля, помеченные *, обязательны к заполнению)</caption>";
  // скрытые поля
  $text1.="<input name=type2 type='hidden' value='".$Id[type2]."'>";
  $text1.="<input name=type type='hidden' value='".$Id[type]."'>";
  $text1.="<tr><td width=50%>Фамилия*</td>
           <td width=50%><input type='text' name='name1' id='name1'
             size='30' maxlength='30' value='' onchange='
             xajax_Control_Reg33(xajax.getFormValues(\"Form_Reg33\"));'>
             <div id='div_reg_name1'></div></td></tr>";
  $text1.="<tr><td width=50%>Имя*</td>
           <td width=50%><input type='text' name='name2' id='name2'
             size='30' maxlength='30' value='' onchange='
             xajax_Control_Reg33(xajax.getFormValues(\"Form_Reg33\"));'>
             <div id='div_reg_name2'></div></td></tr>";
  ... ... ...
  $text1.="<tr><td width=50%>Дата выдачи</td>
           <td width=50%><input type='text' name='pasport_data'
           id='pasport_data' size='30' maxlength='10' value='' >
           </td></tr>";
  $text1.="<tr><td><input type='button' onclick='
           xajax_Form_Reg22(xajax.getFormValues(\"Form_Reg33\"));'
```

```
          value='<-- Назад ' ></td>
          <td><input type='submit' id='Button_Form_Reg33'
          value='Зарегистрироваться ->' disabled=true>
          </td></tr>";
  $text1.="</table>";
  $text1.="</form>";
  return $text1;
}
?>
```

xAjax-функция контроля правильности заполнения полей `Control_Reg33()` находится в файле hosting\my_prg_reg\control_reg33.php, содержимое которого представлено в листинге 6.13.

Листинг 6.13

```
<?php
// Проверка правильности заполнения полей
// при регистрации пользователя (физическое лицо)
function Control_Reg33($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  $count=0;
  // Фамилия
  if(strlen($Id[name1])<2)
    $objResponse->assign("div_reg_name1","innerHTML","<font color='red'>
         Введите фамилию</font>");
  else
  { $objResponse->assign("div_reg_name1","innerHTML","<table></table>");
    $count++;
  }
  // Имя
  if(strlen($Id[name2])<2)
    $objResponse->assign("div_reg_name2","innerHTML","<font color='red'>
         Введите имя</font>");
  else
  { $objResponse->assign("div_reg_name2","innerHTML","<table></table>");
    $count++;
  }
  // e-mail
  if(strlen($Id[email])<1)
    $objResponse->assign("div_reg_email","innerHTML","<font color='red'>
      Введите e-mail</font>");
  elseif(ereg("^([a-z,0-9,-_\.]{2,20})([\@]{1})([a-z,0-9,
         -_]{2,20})([\.]{1})([a-z,]{1,3})$",$Id[email]))
  { // поиск в базе email
    $query12="SELECT id FROM my_users_info2 WHERE email='".$Id[email]."'";
```

```
    $query13="SELECT id FROM my_users_info3 WHERE email='".$Id[email]."' ";
    $rez12=mysql_query($query12);
    $rez13=mysql_query($query13);
    if(mysql_num_rows($rez12)>0 || mysql_num_rows($rez13)>0)
    { $objResponse->assign("div_reg_email","innerHTML",
        "<font color='red'>Уже есть счет на этот e-mail</font>");
    }
    else
    { $objResponse->assign("div_reg_email","innerHTML","<table></table>");
      $count++;
    }
  }
  else
    $objResponse->assign("div_reg_email","innerHTML",
       "<font color='red'>Неверный формат email</font>");
  if($count==3)
    $objResponse->assign("Button_Form_Reg33","disabled",false);
  else
    $objResponse->assign("Button_Form_Reg33","disabled",true);
  return $objResponse;
  }
?>
```

При нажатии кнопки **Зарегистрироваться** вызывается xAjax-функция `Reg_User33()`. Функция записывает полученные данные в базу данных, генерирует пароль, отправляет письмо с паролем на введенный e-mail и выводит сообщение об успешной активации на страницу. При этом вход в панель ISPmanager будет доступна только после выбора и оплаты тарифа. Функция `Reg_User33()` находится в файле hosting\my_prg_reg\reg_user33.php, содержимое которого представлено в листинге 6.14.

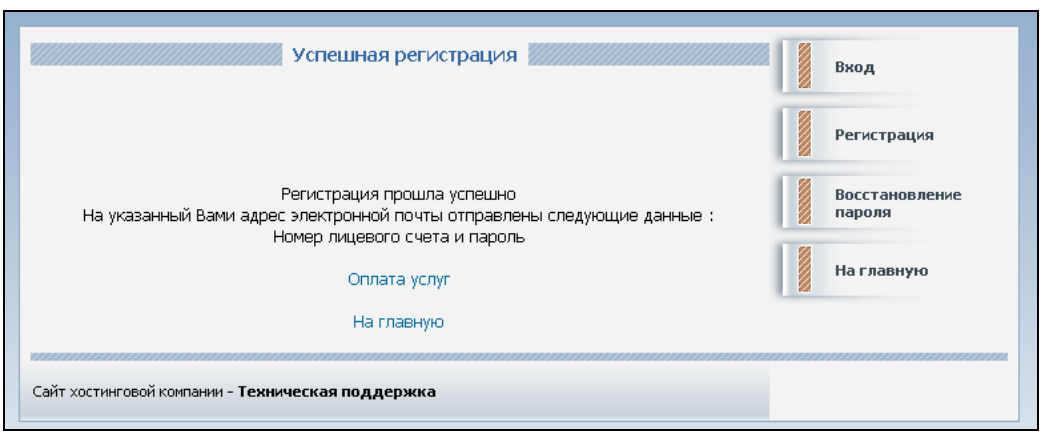


Рис 6.8. Сообщение об успешной регистрации

Листинг 6.14

```
<?php
// Регистрация пользователя (физ. лицо)
function Reg_User33($Id)
{ $objResponse = new xajaxResponse();
  // подключение к базе
  require_once("mybaza.php");require_once("my.php");
  // проверка данных
  $type2=$Id[type2];
  $name1=proverka1(utftowin($Id[name1]));
  $name2=proverka1(utftowin($Id[name2]));
  ... ... ...
  $data=date('Y-m-d H:i:s');
  // занесение данных в my_users_info3
  $query1="INSERT INTO my_users_info3 SET name1='".$name1."',
          name2='".$name2."',name3='".$name3."',email='".$email."',
          phone='".$phone."',address='".$address."',
          pasport_number='".$pasport_number."',
          pasport_place='".$pasport_place."',
          pasport_data='".$pasport_data."'
          ";
  $rez1=mysql_query($query1);
  ... ... ...
  $id_info=mysql_insert_id();
  // создание лицевого счета
  $query21="INSERT INTO my_l_schet SET name='0' ";
  $rez21=mysql_query($query21);
  $id_l_schet=mysql_insert_id();
  if($id_l_schet>0)
  { $query22="UPDATE my_l_schet SET
              name='".sprintf("%06s",$id_l_schet)."'
              WHERE id='".$id_l_schet."' ";
    $rez22=mysql_query($query22);
  ... ... ...
  }
  else
  { $text1="Ошибка MySQL (reg33-2) — Обратитесь к администратору";
    $objResponse->assign("center2","innerHTML",$text1);
    return $objResponse;
  }
  // занесение данных в my_users
  $password=create_password();
  $query3="INSERT INTO my_users SET type='3',type2='".$type2."',
          id_info='".$id_info."',id_l_schet='".$id_l_schet."',
          password='".$password."',visible='yes',
          data_reg='".$data."' ";
```

```
  $rez3=mysql_query($query3);
  ... ... ...
  // отправка на e-mail
  ... ... ...
  mail($to,$subject,$body,$headers);
  // подготовка контента для вывода
  ... ... ...
  // вывод контента
  $objResponse->assign("center2","innerHTML",$text1);
  $objResponse->assign("zag1","innerHTML",
    "<font class='blocktitle'> Успешная регистрация </font>");
  return $objResponse;
}
?>
```

Для юридического лица подбор функций аналогичен. Мы их рассматривать не будем. Файлы с этими функциями находятся на диске в папке glava_06\hosting\my_prg_reg.

6.5. Вход в систему, восстановление пароля

При входе на главную страницу личного кабинета либо при выборе пункта меню **Вход** формируется форма входа (см. рис. 6.1). Необходимо ввести номер лицевого счета и пароль. Переход к форме восстановления пароля (рис. 6.9) по ссылке **Забыли пароль?**. xAjax-функция `Form_Password_Mail()` (листинг 6.15) создает форму восстановления пароля. При этом кнопка **Получить** неактивна до тех пор, пока не будет введен адрес электронной почты. Если такого адреса нет в базе, выводится соответствующее сообщение (рис. 6.10). Проверку e-mail производит xAjax-функция `Control_Get_Password()` (листинг 6.16).

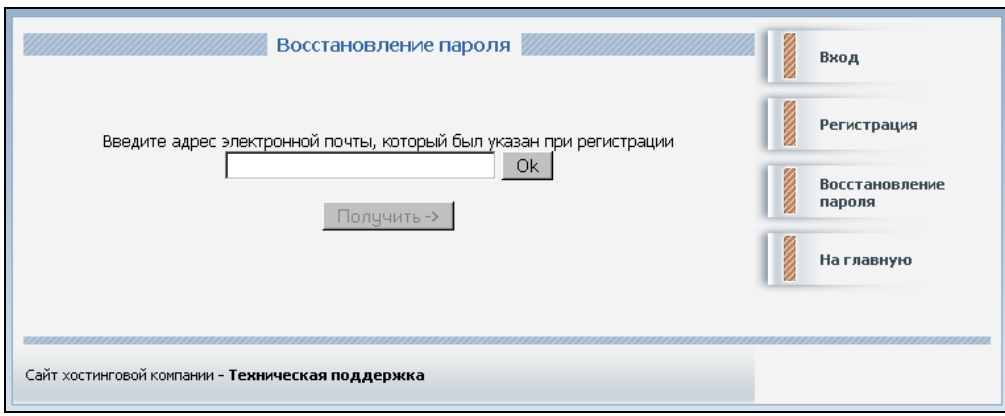


Рис. 6.9. Форма восстановления пароля

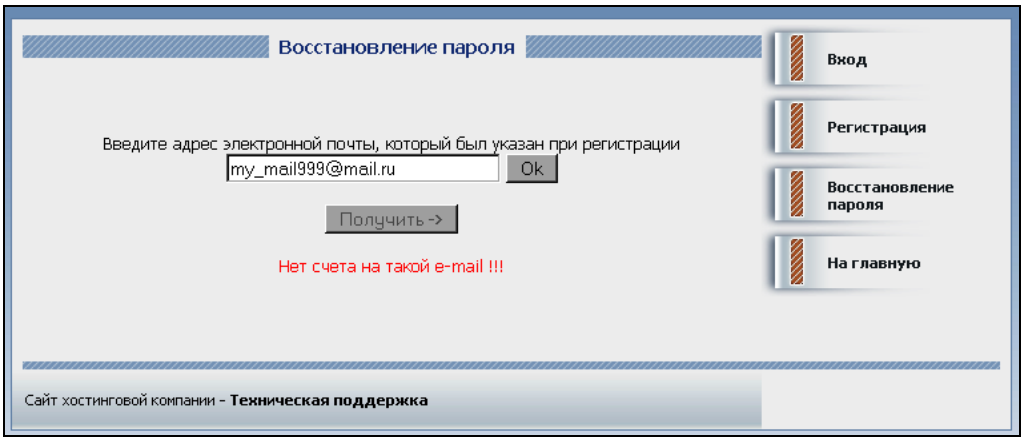


Рис. 6.10. Форма восстановления пароля — ошибка при вводе e-mail

Листинг 6.15

```
<?php
// Восстановление пароля и номера счета
// по email – создание формы
function Form_Password_Mail()
{ $objResponse = new xajaxResponse();
  // формирование формы
  $text1="<center><br><br><br>
       <form name='Form_Password_Mail' id='Form_Password_Mail'
       action='javascript:void(null);' onsubmit='
       xajax.$(\"Button_Form_Password_Mail\").disabled=true;
       xajax.$(\"Button_Form_Password_Mail\").value=\"Подождите...\";
       xajax_Get_Password_Mail(xajax.getFormValues(
       \"Form_Password_Mail\"));'>
       Введите адрес электронной почты, который был указан при регистрации
       <input type='text' id='email' name='email' value='' size=30
       maxlength=30 onchange='xajax_Control_Get_Password(this.value);'>
       <input type='button' value='Ok'>
       <input type='submit' id='Button_Form_Password_Mail'
              value='Получить ->' disabled=true>
       </form>
       <div id='div_password_mail'></div>
       </center>";
  // вывод контента
  $objResponse->assign("zag1","innerHTML",
  "<font class='blocktitle'> Восстановление пароля </font>");
  $objResponse->assign("center1","innerHTML","<table></table>");
  $objResponse->assign("center2","innerHTML",$text1);
  $objResponse->assign("center3","innerHTML","<table></table>");
  $objResponse->assign("center4","innerHTML","<table></table>");
```

```
  $objResponse->assign("center5","innerHTML","<table></table>");
  $objResponse->assign("center6","innerHTML","<table></table>");
  return $objResponse;
}
?>
```

Листинг 6.16

```
<?php
// Проверка правильности заполнения e-mail при восстановлении пароля
function Control_Get_Password($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  $count=0;
  // email
  if(strlen($Id)<1)
    $objResponse->assign("div_password_mail","innerHTML","<font
    color='red'>Введите e-mail</font>");
  elseif(ereg("^([a-z,0-9,-_\.]{2,20})([\@]{1})([a-z,0-9,-
_]{2,20})([\.]{1})([a-z,]{1,3})$",$Id))
  { // поиск email-адреса в базе
    $query12="SELECT id FROM my_users_info2 WHERE email='".$Id."' ";
    $query13="SELECT id FROM my_users_info3 WHERE email='".$Id."' ";
    $rez12=mysql_query($query12);
    $rez13=mysql_query($query13);
    if(mysql_num_rows($rez12)>0 || mysql_num_rows($rez13)>0)
    { $objResponse->assign("div_password_mail",
                           "innerHTML","<table></table>");
      $count++;
    }
    else
    { $objResponse->assign("div_password_mail","innerHTML",
                  "<font color='red'>
                  Нет счета на такой e-mail!!!</font>");
    }
  }
  else
    $objResponse->assign("div_password_mail","innerHTML",
                "<font color='red'>
                Неверный формат e-mail</font>");
  // активация или дезактивация кнопки
  if($count==1)
    $objResponse->assign("Button_Form_Password_Mail","disabled",false);
  else
    $objResponse->assign("Button_Form_Password_Mail","disabled",true);
  return $objResponse;
}
?>
```

При нажатии кнопки **Получить** вызывается xAjax-функция `Get_Password_Mail()`, которая получает данные лицевого счета и пароля из таблицы `my_users_info2` или таблицы `my_users_info3` и отправляет данные письмом на введенный электронный адрес. На страницу выводится соответствующее сообщение (рис. 6.11). Функция `Get_Password_Mail()` находится в файле hosting\my_prg_vhod\get_password_mail.php, содержимое которого представлено в листинге 6.17.

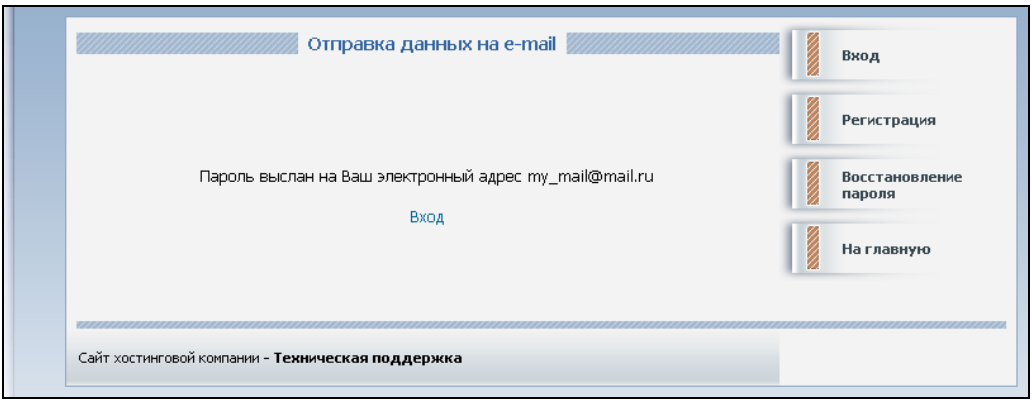


Рис. 6.11. Результат восстановления пароля

Листинг 6.17

```
<?php
// Отправка пароля на e-mail
function Get_Password_Mail($Id)
{ $objResponse = new xajaxResponse();

  require_once("mybaza.php");
  require_once("my.php");
  $query1="SELECT my_users.password,my_users.id_l_schet
           FROM my_users,my_users_info2
           WHERE my_users.id_info=my_users_info2.id &&
           my_users_info2.email='".$Id[email]."' && my_users.type='2' ";
  $rez1=mysql_query($query1);
  if(mysql_num_rows($rez1)>0) // юридическое лицо
  { $row1=mysql_fetch_row($rez1);
    $password=$row1[0];
    $id_l_schet=$row1[1];
  }
  else // физическое лицо
  { $query2="SELECT my_users.password,my_users.id_l_schet
             FROM my_users,my_users_info3
             WHERE my_users.id_info=my_users_info3.id &&
             my_users_info3.email='".$Id[email]."' && my_users.type='3' ";
    $rez2=mysql_query($query2);
    $row2=mysql_fetch_row($rez2);
```

```
    $password=$row2[0];
    $id_l_schet=$row2[1];
  }
// отправка на e-mail
  $to=$Id[email];
  $subject='Восстановление пароля ';
  $body.='Ваш лицевой счет <b>'.sprintf("%06s",$id_l_schet).'<br>';
  $body.='Ваш пароль <br>'.$password.'<br>';
  $body.='<br>';
  $body.='С уважением,    '.SITE.'<br>';
  $body.='Адрес для обратной связи — <a href=
       "mailto:'.EMAILADMIN.'">'.EMAILADMIN.'</a><br>';
       $headers="Content-type: text/html; charset=windows-1251;";
  mail($to,$subject,$body,$headers);
  // формирование контента
  $text1="<center>";
  $text1.="
  <br><br><br><br><br>
  Пароль выслан на Ваш электронный адрес ".$Id[email]."
  <br><br>
  <a href='javascript:void();' onclick='xajax_Form_Vhod();'>Вход</a>
  <br><br>";
  $text1.="<center>";
  // вывод контента
  $objResponse->assign("center2","innerHTML",$text1);
  $objResponse->assign("zag1","innerHTML",
    "<font class='blocktitle'> Отправка данных на e-mail </font>");
  return $objResponse;
}
?>
```

Опять возвращаемся к форме входа в личный кабинет (см. рис. 6.1). Необходимо ввести номер лицевого счета и пароль. При нажатии кнопки **Вход** вызывается xAjax-функция `Vhod()`, которая проверяет наличие в базе пользователя с такими данными и в случае успеха выводит список тарифных планов, доступных для заказа данным пользователем (рис. 6.12). Функция `Vhod()` расположена в файле hosting\my_prg_vhod\vhod.php, содержимое которого представлено в листинге 6.18.

Листинг 6.18

```
<?php
// Вход — проверка по номеру счета и паролю
function Vhod($Id)
{ $objResponse = new xajaxResponse();
  ... ... ...
  $query1="SELECT id FROM my_l_schet WHERE name='".$l_schet."' ";
  $rez1=mysql_query($query1);
```

```
if(mysql_num_rows($rez1)<1)
{... ... ...
return $objResponse;}
$id_l_schet=mysql_result($rez1,0);
$query2="SELECT * FROM my_users WHERE id_l_schet='".$id_l_schet."'
        && password='".$password."' ";
$rez2=mysql_query($query2);
if(mysql_num_rows($rez2)<1)
{... ... ...
  return $objResponse;}
// авторизация — успешно
$txt1="<div class='bghr' align='left'><font class='blocktitle'>
       Личный кабинет </font></div>";
$objResponse->assign("zag0","innerHTML",$txt1);
$row2=mysql_fetch_assoc($rez2);
$_SESSION[id_user]=$row2[id]; $_SESSION[type]=$row2[type];
$_SESSION[type2]=$row2[type2]; $_SESSION[id_info]=$row2[id_info];
// информация о клиенте
$user_info="<table  width=100%><tr>";
if($_SESSION[type2]=='1')
  $user_info.="<b>Хостинг</b></tr><tr>";
elseif($_SESSION[type2]=='2')
  $user_info.="<b>Реселлер</b></tr><tr>";
else
  ;
$user_info.="<td valign=top width=40%>Лицевой счет
     <b>".$l_schet."</b><br>";
$user_info.="Наименование клиента<br><b>";
// юридическое лицо
if($_SESSION[type]==2)
{ $query3="SELECT name FROM my_users_info2 WHERE
         id='".$_SESSION[id_info]."' ";
  $user_info.=mysql_result(mysql_query($query3),0);
}
// физическое лицо
elseif($_SESSION[type]==3)
{ $query3="SELECT name1,name2,name3 FROM my_users_info3 WHERE
           id='".$_SESSION[id_info]."' ";
  $user_info.=mysql_result(mysql_query($query3),0,"name1")." ";
  $user_info.=mysql_result(mysql_query($query3),0,"name2")." ";
  $user_info.=mysql_result(mysql_query($query3),0,"name3")." ";
}
... ... ...
// администратор
elseif($_SESSION[type]==8)
{ $query3="SELECT name1,name2,name3 FROM my_users_info3 WHERE
  id='".$_SESSION[id_info]."' ";
  $user_info="Модератор <br>";
```

```
    $user_info.=mysql_result(mysql_query($query3),0,"name1")." ";
    $user_info.=mysql_result(mysql_query($query3),0,"name2")." ";
    $user_info.=mysql_result(mysql_query($query3),0,"name3")." ";
  }
  else ;
  $user_info.="</b></td>";
  // текущий тарифный план
  ... ... ...
  // и следующий тарифный план
  ... ... ...
  $objResponse->assign("user_info","innerHTML",$user_info);
  // меню
  $new_menu=function_create_menu($_SESSION[type]);
  $objResponse->assign("submenu","innerHTML",$new_menu);
  // выбор списка доступных тарифных планов
  switch($_SESSION[type2])
  {
  ... ... ...
  }
  // вывод контента
  ... ... ...
}
?>
```

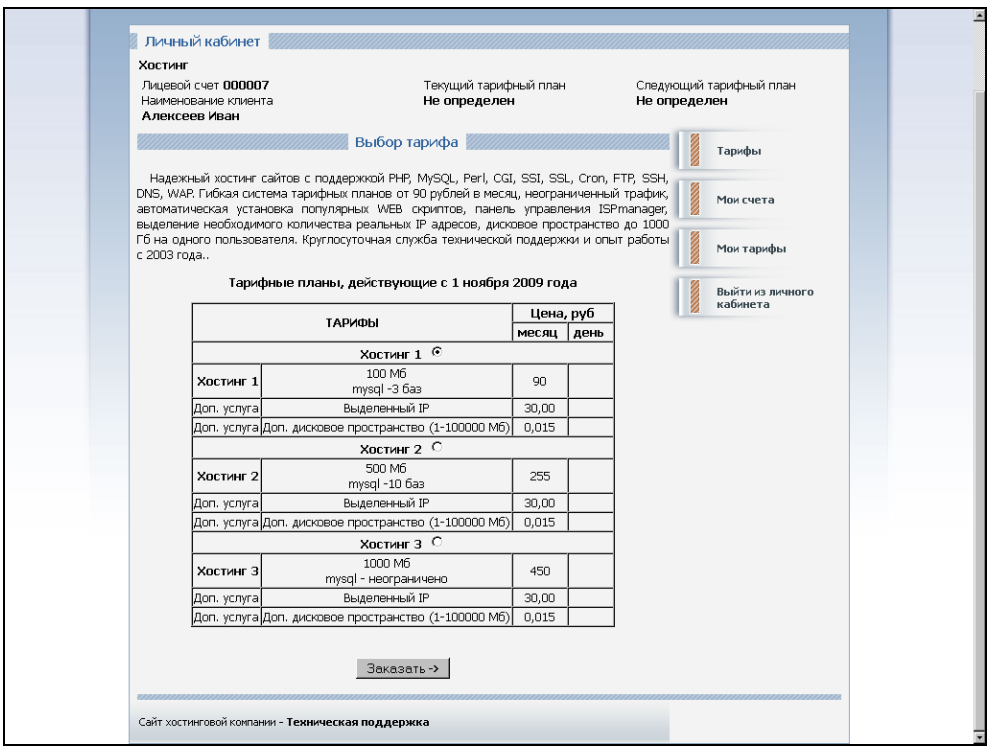


Рис. 6.12. При входе в личный кабинет

Файлы с процедурами данного блока находятся на прилагаемом диске в папке glava_06\ hosting\my_prg_vhod.

6.6. Выбор тарифного плана

Для разных пользователей (физическое лицо/юридическое, хостинг/реселлер, VIP) существует свой набор тарифов. При входе в личный кабинет (см. рис. 6.12) или при выборе пункта меню **Тариф** производится выбор из базы и вывод списка возможных тарифов для пользователя. Это делает xAjax-функция `View_Tarifs()` (листинг 6.19).

Листинг 6.19

```
<?php
// Просмотр и выбор тарифного плана
// $Id — type
function View_Tarifs()
{ $objResponse = new xajaxResponse();
  switch($_SESSION[type2])
  { // юридическое лицо
    case 1: $text1=function_view_tarifs2();     break;
    // физическое лицо
    case 2: $text1=function_view_tarifs3();     break;
    // администратор
    case 8: $text1=function_view_tarifs8();     break;
    default:$text1="SESSION=".$_SESSION[type];  break;
  }
  // вывод контента
  $objResponse->assign("zag1","innerHTML",
  "<font class='blocktitle'> Выбор тарифа </font>");
  $objResponse->assign("center1","innerHTML","<table></table>");
  $objResponse->assign("center2","innerHTML",$text1);
  $objResponse->assign("center3","innerHTML","<table></table>");
  $objResponse->assign("center4","innerHTML","<table></table>");
  $objResponse->assign("center5","innerHTML","<table></table>");
  $objResponse->assign("center6","innerHTML","<table></table>");
  return $objResponse;
}
?>
```

Само формирование контента для вывода происходит в функциях:

- `function_view_tarifs2()` — для пользователей;
- `function_view_tarifs8()` — для администратора (вывод всех тарифов).

Содержимое функции `function_view_tarifs2()` представлено в листинге 6.20 (файл hosting\ my_prg_tarifs\function_view_tarifs2.php).

Листинг 6.20

```
<?php
// Выбор тарифного плана
function function_view_tarifs2()
{ require_once("mybaza.php");
  $text1="<p align='justify'>   Надежный хостинг сайтов
          .........панель управления...... </p>";
  $text1.="<p align='center'><strong>Тарифные планы</strong></p>";
  $text1.="<form name='Form_Tarifs' id='Form_Tarifs'
         action='javascript:void(null);' onsubmit='
         xajax.$(\"Button_Form_Tarifs\").disabled=true;
       xajax.$(\"Button_Form_Tarifs\").value=\"Подождите...\";
       xajax_Create_Zakaz1(xajax.getFormValues(\"Form_Tarifs\"));'>";
  ... ... ...
  $text11="<tr><td align='center' rowspan='2' colspan='2'>
           <strong>ТАРИФНЫЕ ПЛАНЫ</strong></td>";
  ... ... ...
  // заполнение таблиц
  $query1="SELECT * FROM my_tarifs WHERE activ='yes' && id_tarif='0' &&
          type_users LIKE '%".$_SESSION[type]."%' &&
          type=".$_SESSION[type2]." ORDER BY sort ASC ";
  $rez1=mysql_query($query1);
  $checked="checked";
  while($row1=mysql_fetch_assoc($rez1))
  { $text11.="<tr align='center'>
            <td colspan='6'><strong>".$row1[name]."</strong>
            <input type=radio name=tarif ".$checked."
             value='".$row1[id]."'></td></tr>";
    $checked="";
    $text11.="<tr> <td><strong> ".$row1[name]."</strong></td>";
    $text11.="<td align='center'>
             ".str_replace(";","<br>",$row1[info])."</td>";
    $pay1=explode(";",$row1[pay1]);
    $pay2=explode(";",$row1[pay2]);
    if($pay1[1]=='-')
      $text11.="<td> - </td><td> - </td>";
    else
    { $text11.="<td align='center'>".$pay1[1]."</td>";
      $text11.="<td align='center'>".$pay2[1]."</td>";
      $checked="";
    }
    $text11.="</tr>";
    // дополнительные услуги
    $query2="SELECT * FROM my_tarifs WHERE activ='yes' &&
          id_tarif='".$row1[id]."' ORDER BY sort ASC ";
```

```
    $rez2=mysql_query($query2);
    while($row2=mysql_fetch_assoc($rez2))
    {
    ... ... ...
    }
  }
  $text1.=$text11.$text12."</table>";
  $text1.="<input type='hidden' name='type' value='".$_SESSION[type]."'>";
  $text1.="<input type='hidden' name='type2' value='".$_SESSION[type2]."'>";
  $text1.="<input type='hidden' name='id_user'
           value='".$_SESSION[id_user]."'>";
  $text1.="<center><br><br><input type='submit' id='Button_Form_Tarifs'
           value='Заказать ->'></center>";
  return $text1;
}
?>
```

Для выбора тарифного плана необходимо выбрать тарифный план и нажать кнопку **Заказать**.

6.7. Заказ тарифного плана. Формирование счета

При выборе тарифного плана вызывается xAjax-функция `Create_Zakaz1()`, переходим на форму выбора дополнительных услуг и периода действия тарифного плана. Набор услуг для каждого тарифа может быть разным. По умолчанию устанавливается период 30 дней без дополнительных услуг (рис. 6.13). Кроме того, введена возможность единовременного бесплатного пробного периода (10 дней). Если пользователь еще не пользовался данной возможностью, выводится соответствующий флажок. Содержимое функции `Create_Zakaz1()` представлено в листинге 6.21.

Листинг 6.21

```
<?php
// Создать счет – шаг 1.
//   Подтвердить создание счета
function Create_Zakaz1($Id)
{ $objResponse = new xajaxResponse();
  require_once("mybaza.php");
  $type=$Id[type];$id_user=$Id[id_user]; $id_tarif=$Id[tarif];
  // получение данных для формы подтверждения
  $query1="SELECT name,info,pay2 FROM my_tarifs WHERE id='".$id_tarif."' ";
  $rez1=mysql_query($query1);
  $arrsumma=explode(";",mysql_result($rez1,0,"pay2"));
  $arrsumma[1]=str_replace(",",".",$arrsumma[1]);
  $summa=$arrsumma[1]*30;
```

```
// формирование контента
$text1<form name='Form_Zakaz' id='Form_Zakaz'
     action='javascript:void(null);'
     onsubmit='xajax.$(\"Button_Form_Zakaz\").disabled=true;
     xajax.$(\"Button_Form_Zakaz\").value=\"Подождите...\";
     xajax_Create_Zakaz2(xajax.getFormValues(\"Form_Zakaz\"));'>
     <input type='hidden' name='type' value='".$type."' >
     <input type='hidden' name='id_user' value='".$id_user."' >
     <input type='hidden' name='id_tarif' value='".$id_tarif."' >
     <input type='hidden' name='pay2' value='".$arrsumma[1]."' >
     <input type='hidden' name='summa' value='".$summa."' >
     <br>
     Вы выбрали тариф <br>
     <b>".mysql_result($rez1,0,"name")."</b><br>
     ".str_replace(";","<br>",mysql_result($rez1,0,"info"))."<br>";
$text1.="Кол-во дней <input type='text' name='days' value='30' size=3
        maxlength=3 onchange='xajax_Calculator1(
          xajax.getFormValues(\"Form_Zakaz\"))'>";
 // получение дополнительных услуг
 $query2="SELECT * FROM my_tarifs WHERE activ='yes' &&
         id_tarif='".$id_tarif."' ORDER BY sort ASC ";
 $rez2=mysql_query($query2);
 $text1.="<br><b>Дополнительные  услуги</b><br>";
  ... ... ...
 $text1.="<center><br><br><br><div id='calc1_days'>
     Срок <b>5</b> дней</div>
     <div id='calc1_summa'>Сумма <b>".$summa."</b> руб</div><br><br>
     <br><br>";
 $query3="SELECT oplata_podkl FROM my_users WHERE id='".$id_user."' ";
 $rez3=mysql_query($query3);
 $prob=mysql_result($rez3,0);
 // возможность бесплатного пробного периода
 if($prob=='no')
   $text1.="Вы можете выбрать бесплатный пробный период (10 дней)
     <br>
     <input type='checkbox' name='prob' value='1'
     onchange='xajax_Calculator1(xajax.getFormValues(\"Form_Zakaz\"))'>
     <br><br>";
     $text1.="<input type='submit' id='Button_Form_Zakaz'
     value='Подтвердить ->'>
     </form>";
... ... ...
// вывод контента
$objResponse->assign("zag1","innerHTML",
"<font class='blocktitle'> Подтверждение выбора тарифа </font>");
$objResponse->assign("center1","innerHTML","<table></table>");
$objResponse->assign("center2","innerHTML",$text1);
```

```
  $objResponse->assign("center3","innerHTML","<table></table>");
  $objResponse->assign("center4","innerHTML","<table></table>");
  $objResponse->assign("center5","innerHTML","<table></table>");
  $objResponse->assign("center6","innerHTML","<table></table>");
  return $objResponse;
}
?>
```

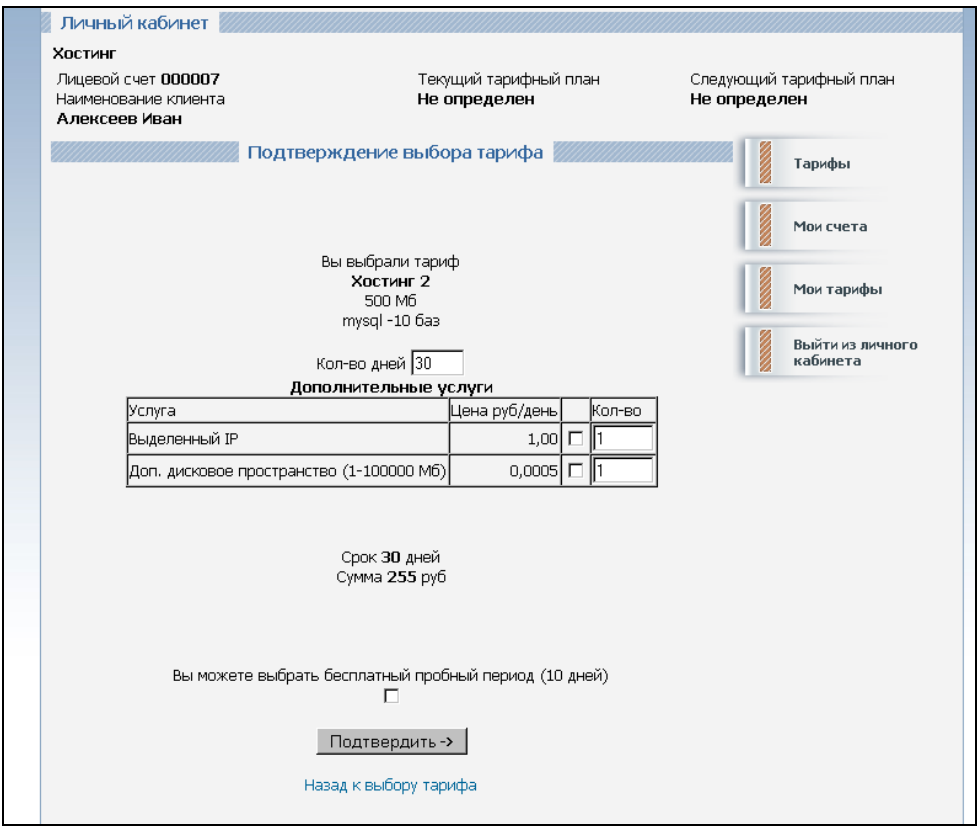


Рис. 6.13. Форма выбора дополнительных услуг

При изменении параметров тарифного плана (период действия или выбор дополнительных услуг) вызывается xAjax-функция `Calculator1()`, которая, получая новые данные формы, пересчитывает сумму оплаты (рис. 6.14). При выборе периода менее 5 дней устанавливается минимальный — 5 дней. Зарегистрированный пользователь может в любое время, но только 1 раз, выбрать бесплатный пробный период продолжительностью 10 дней, при этом калькулятор выставляет сумму счета 0 рублей.

При подтверждении выбранных услуг вызывается xAjax-функция `Create_Zakaz2()`, которая создает новый неоплаченный счет для пользователя и отправляет уведомление на почту пользователя. В контент выводится сообщение о создании нового

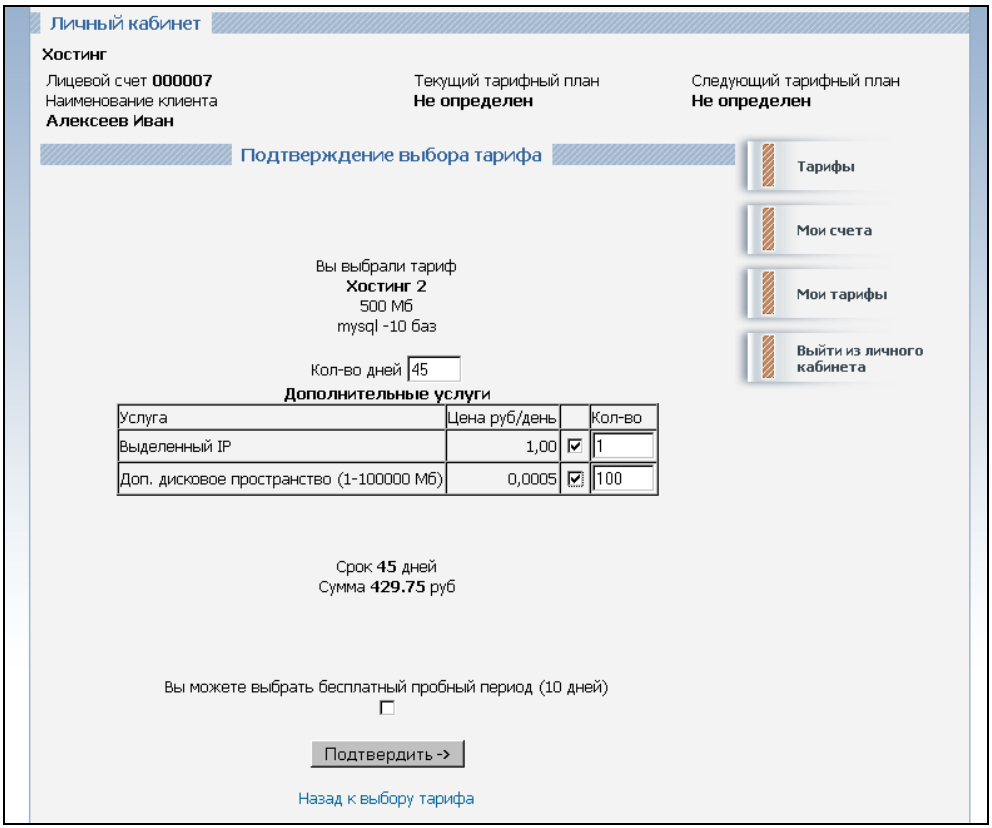


Рис. 6.14. Пересчет суммы оплаты за услуги

счета и кнопка о переходе к оплате (рис. 6.15). Функция `Create_Zakaz2()` находится в файле hosting\my_prg_zakaz\create_zakaz2.php, содержимое которого представлено в листинге 6.22.

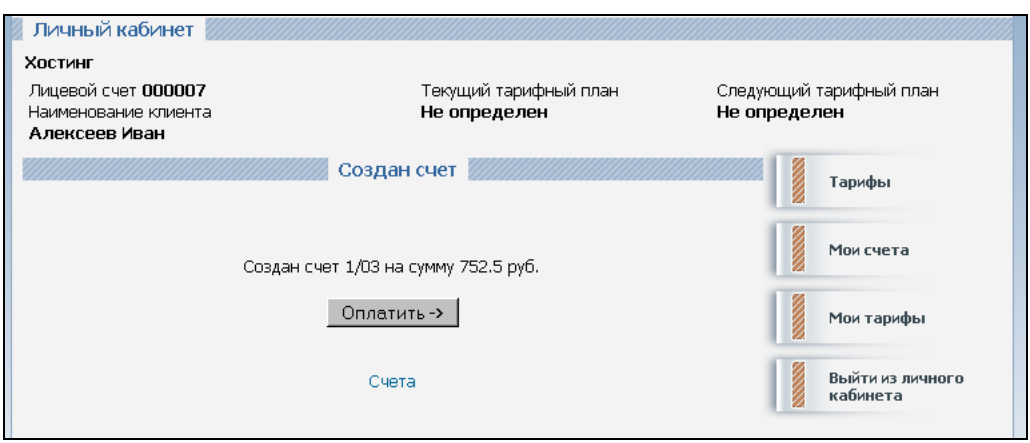


Рис. 6.15. Сообщение о создании счета на оплату

Листинг 6.22

```
<?php
// Создание счета в таблице my_schets
function Create_Zakaz2($Id)
{ $objResponse = new xajaxResponse();
  // подключение к базе данных
  require_once("mybaza.php");
  // подготовка данных для записи
  $type=$Id[type];$id_user=$Id[id_user];$id_tarif=$Id[id_tarif];
  $days_id_tarif=$Id[days];$summa=$Id[summa];$oplata='no';
  $data_oplata='0000-00-00 00:00:00';
  if($Id[prob]=='1')
  { $oplata='yes';
    $data_oplata=date('Y-m-d H:i:s'); }
  // дополнительные услуги
  $query2="SELECT * FROM my_tarifs WHERE activ='yes' &&
           id_tarif='".$id_tarif."' ORDER BY sort ASC ";
  $rez2=mysql_query($query2);$i=0;
  $dop_uslugi=";";$info1="";
  while($row2=mysql_fetch_assoc($rez2))
  { $i++;
    if($Id[usluga.$i]>0)
    { $dop_uslugi.=$row2[id]."-".$Id[kol.$i].";";
      $info1.=" ".$row2[name]." — ".$Id[kol.$i].";";
    }
    else  $dop_uslugi.=$row2[id]."-0;";
  }
  // вычисление номера счета
  $number=date('/m');
  $query1="SELECT COUNT(id) FROM my_schets WHERE number LIKE '%".$number."' ";
  $rez1=mysql_query($query1);
  $number="".(mysql_result($rez1,0)+1).$number;
  // создание счета
  $query1="SELECT name,info,pay1 FROM my_tarifs WHERE id='".$id_tarif."' ";
  $rez1=mysql_query($query1);
  $info="Заказ тарифного плана ".mysql_result($rez1,0,"name")." на
        ".$days_id_tarif." дней";
  if($Id[prob]=='1')
    $info.=" — ПРОБНЫЙ ПЕРИОД";
  if(strlen($info1)>0)
    $info.=" + Доп. услуги : ".$info1;
  $query2="INSERT INTO my_schets SET number='".$number."',
          id_users='".$id_user."',id_tarif='".$id_tarif."',
          days_id_tarif='".$days_id_tarif."',summa='".$summa."',
           dop_uslugi='".$dop_uslugi."',info='".$info."',oplata='".$oplata."',
          data='".date('Y-m-d H:i:s')."', data_oplata='".$data_oplata."'";
```

```
  $rez2=mysql_query($query2);
  //***** ошибка создания счета
  if(!$rez2)
  ... ... ...
  //** ok
  $id_schet=mysql_insert_id();
  // форма перехода на оплату
  ... ... ...
  // отправка на e-mail
  ... ... ...
  // вывод контента
  $objResponse->assign("zag1","innerHTML",
  "<font class='blocktitle'> Создан счет </font>");
  $objResponse->assign("center1","innerHTML","<table></table>");
  $objResponse->assign("center2","innerHTML",$text1);
  $objResponse->assign("center3","innerHTML","<table></table>");
  $objResponse->assign("center4","innerHTML","<table></table>");
  $objResponse->assign("center5","innerHTML","<table></table>");
  $objResponse->assign("center6","innerHTML","<table></table>");
  return $objResponse;
}
?>
```

Оплатить счет можно и позже, все созданные счета можно просмотреть по ссылке **Счета** (рис. 6.16) либо выбрав пункт меню **Мои счета**.

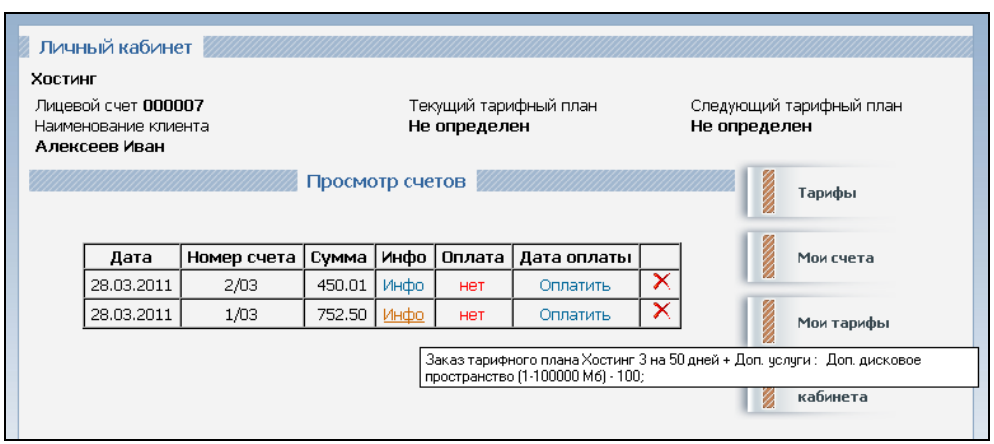


Рис. 6.16. Все счета пользователя

6.8. Счета пользователя

Переход к просмотру всех выписанных счетов осуществляется по выбору пункта меню **Мои счета**. Вызывается xAjax-функция `View_My_Schets()`, которая выводит список всех счетов пользователя в табличном виде. Функция `View_My_Schets()` на-

ходится в файле hosting\my_prg_zakaz\view_my_schets.php, содержимое которого представлено в листинге 6.23. Формированием контента занимается функция `function_view_my_schets()` (листинг 6.24). Из таблицы можно увидеть информацию о тарифе, об оплате. Для неоплаченных счетов есть ссылка **Оплатить** для перехода на форму оплаты (см. рис. 6.15).

Листинг 6.23

```
// Просмотр своих счетов
function View_My_Schets()
{ $objResponse = new xajaxResponse();
  // создание контента
  $text1=function_view_my_schets($_SESSION[id_user]);
  // вывод контента
  $objResponse->assign("zag1","innerHTML",
    "<font class='blocktitle'> Просмотр счетов </font>");
  $objResponse->assign("center1","innerHTML","<table></table>");
  $objResponse->assign("center2","innerHTML",$text1[0]);
  $objResponse->assign("center3","innerHTML",$text1[1]);
  $objResponse->assign("center4","innerHTML","<table></table>");
  $objResponse->assign("center5","innerHTML","<table></table>");
  $objResponse->assign("center6","innerHTML","<table></table>");
  return $objResponse;
}
?>
```

Листинг 6.24

```
<?php
// Просмотр своих счетов
function function_view_my_schets($Id)
{ $text=array();
  require_once("mybaza.php");
  // выбор счетов
  $query1="SELECT * FROM my_schets WHERE id_users='".$_SESSION[id_user]."'
          && oplata<>'del' ORDER BY data DESC ";
  $rez1=mysql_query($query1);
  if(mysql_num_rows($rez1)>0)
  { // заголовок таблицы
    $text1="<br><br><center>";
    $text1.="<table cellspacing='0' cellpadding='1' align='center'
             border='1'>";
    $text1.="<tr>
            <td align='center'><strong> Дата </strong></td>
            ... ... ...
            </tr>";
```

```
    while($row1=mysql_fetch_assoc($rez1))
    { // строки таблицы
      $text1.="<tr>";
      $data=date('d.m.Y',strtotime($row1[data]));
      $text1.="<td align='center'> ".$data." </td>";
      $text1.="<td align='center'> ".$row1[number]." </td>";
      $text1.="<td align='right'> ".$row1[summa]." </td>";
      $text1.="<td align='center'> <a
               title='".$row1[info]."'>Инфо</a> </td>";
      if($row1[oplata]=='yes')
      { $text1.="<td align='center'> да </td>";
        $data_oplata=date('d.m.Y',strtotime($row1[data_oplata]));
        $text1.="<td align='center'> ".$data_oplata." </td>";
        $text1.="<td align='center'>  </td>";
      }
      else
      { $text1.="<td align='center'> <font
                 color='red'>нет</font> </td>";
        $text1.="<td align='center'> <a href='javascript:void();'
                 onclick='xajax_Oplata_Schet1(".$row1[id].");'>
                 Оплатить</a> </td>";
        $text1.="<td align='center'> <a href='javascript:void();'
                 onclick='xajax_Delete_My_Schet(".$row1[id].");'
                 title='Удалить'>
                 <img border=no src='images/delete.png'></a> </td>";
      }
      $text1.="</tr>";
    }
    $text1.="</table>";
    $text1.="</center><br><br>";
  }
  else
  { $text1="<br><br><br><br><center>Нет счетов</center><br><br><br><br>"; }
  $text[0]=$text1;$text[1]=$text2;
  return $text;
}
?>
```

При оплате счета пользователю предлагается несколько вариантов оплаты. Для физического лица возможно оплатить платежкой через Сбербанк (по ссылке формируется извещение (форма ПД-4)) либо электронными деньгами через сервис Robokassa. Если была оплата не online (через Сбербанк или по счет-фактуре), администратор должен подтвердить факт оплаты *(см. разд. 6.12)*. После подтверждения факта оплаты происходит запись в таблицу `my_users` данных текущего, либо будущего, плана, запись данных в таблицу `my_history` и, в случае активации текущего тарифного плана, вызов процедуры, устанавливающей параметры через API ISPmanager *(см. разд. 6.14)*.

6.9. Просмотр, изменение тарифных планов

Посмотреть все свои тарифные планы пользователь может по ссылке **Мои тарифы** из профиля пользователя (рис. 6.17). Вызывается xAjax-функция `View_All_Tarif_User()`, которая выводит не только все действующие и следующие тарифные планы, но и уже закрытые. Это возможно благодаря тому, что данные берутся из таблицы `my_history`, куда записываются все изменения тарифных планов. Содержимое функции `View_All_Tarif_User()` представлено в листинге 6.25.

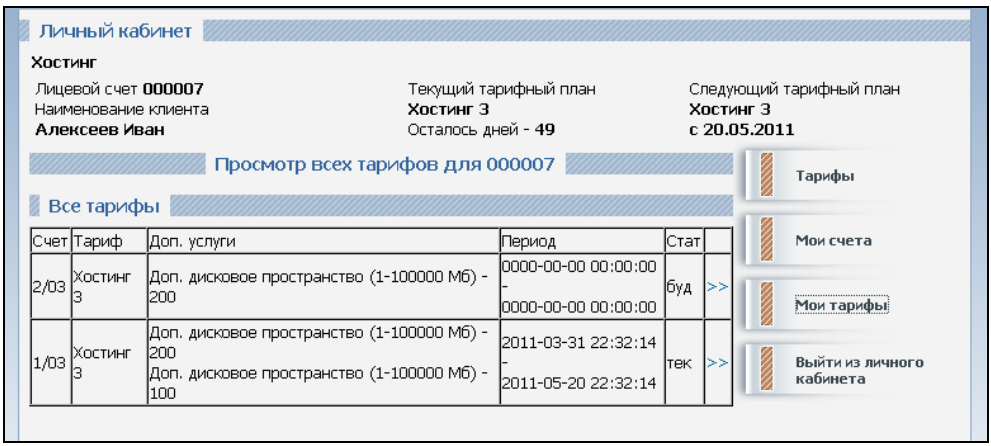


Рис. 6.17. Все тарифные планы пользователя

Листинг 6.25

```
<?php
// Просмотр всех заказанных и оплаченных тарифов пользователя
function View_All_Tarif_User($Id)
{ $objResponse = new xajaxResponse();
  // подключение к базе данных
  require_once("mybaza.php");
  $query0="SELECT * FROM my_users WHERE id='".$Id."' ";
  $rez0=mysql_query($query0);
  $row0=mysql_fetch_assoc($rez0);
  $l_schet=sprintf("%06s",$row0[id_l_schet]);
  // получение всех тарифных планов пользователя
  $query1="SELECT DISTINCT(id_schet) FROM my_history WHERE
      id_user='".$Id."' ORDER BY id DESC ";
  $rez1=mysql_query($query1);
  // шапка таблицы
  $text1="<table cellspacing='0' cellpadding='1' align='center' border='1'>";
  $text1.="<tr><td>Счет</td>";
  ... ... ...
  $text1.="<td>Стат</td>";
```

```
  while($row1=mysql_fetch_assoc($rez1))
  { // получение данных о тарифном плане
    $query2="SELECT * FROM my_history WHERE
      id_schet='".$row1[id_schet]."' ORDER BY id DESC";
    $rez2=mysql_query($query2);
    $row2=mysql_fetch_assoc($rez2);
    $query31="SELECT number FROM my_schets WHERE id='".$row1[id_schet]."' ";
    $rez31=mysql_query($query31);
    $schet=mysql_result($rez31,0);
    $text1.="<tr><td>".$schet."</td>";
    $query32="SELECT name FROM my_tarifs WHERE id='".$row2[id_tarif]."' ";
    $rez32=mysql_query($query32);
    $tarif=mysql_result($rez32,0);
    $text1.="<td>".$tarif."</td>";
    $arrdopuslugi=explode(";",$row2[dop_uslugi]);
    // дополнительные услуги
    for($i=1;$i<count($arrdopuslugi)-1;$i++)
    { $arr2=explode("-",$arrdopuslugi[$i]);
      if($arr2[1]!='0')
      { $query33="SELECT name FROM my_tarifs WHERE id='".$arr2[0]."' ";
        $rez33=mysql_query($query33);
        $text11.=mysql_result($rez33,0)." — ".$arr2[1]."<br>";
      }
    }
    $text1.="<td>".$text11."</td>";
    $text1.="<td>".$row2[data_start]." -<br> ".$row2[data_end]."</td>";
    // статус тарифного плана
    if($row2[visible]=='yes')    $text1.="<td>тек</td>";
    elseif($row2[visible]=='no') $text1.="<td>буд</td>";
    else                         $text1.="<td>закр</td>";
    $text1.="<td><a href='javascript:void()'
             onclick='xajax_View_Client_Tarif(".$row1[id_schet].")'>>>
             </a></td></tr>";
  }
  // заголовок
  $text2="<div class='bghr' align='left'><font class='blocktitle'>
      Все тарифы </font></div>";
  $objResponse->assign("center5","innerHTML",$text2);
  $objResponse->assign("center6","innerHTML",$text1);
  if($_SESSION[type]<'8')
  { $objResponse->assign("zag1","innerHTML",
    "<font class='blocktitle'> Просмотр всех тарифов для ".$l_schet."
    </font>");
  ... ... ...
  }
  return $objResponse;
}
?>
```

Тарифные планы, действующие и следующие, можно редактировать. Вызываются xAjax-функции: `View_Client_Tarif()` (для закрытых тарифных планов) или `Edit_Client_Tarif()` (для действующих и следующих), создающие форму просмотра/редактирования тарифного плана (рис. 6.18). Содержимое функции `Edit_Client_Tarif()` представлено в листинге 6.26. Функция `View_Client_Tarif()` находится в файле hosting\my_prg_tarifs\view_client_tarif.php на прилагаемом компакт-диске.

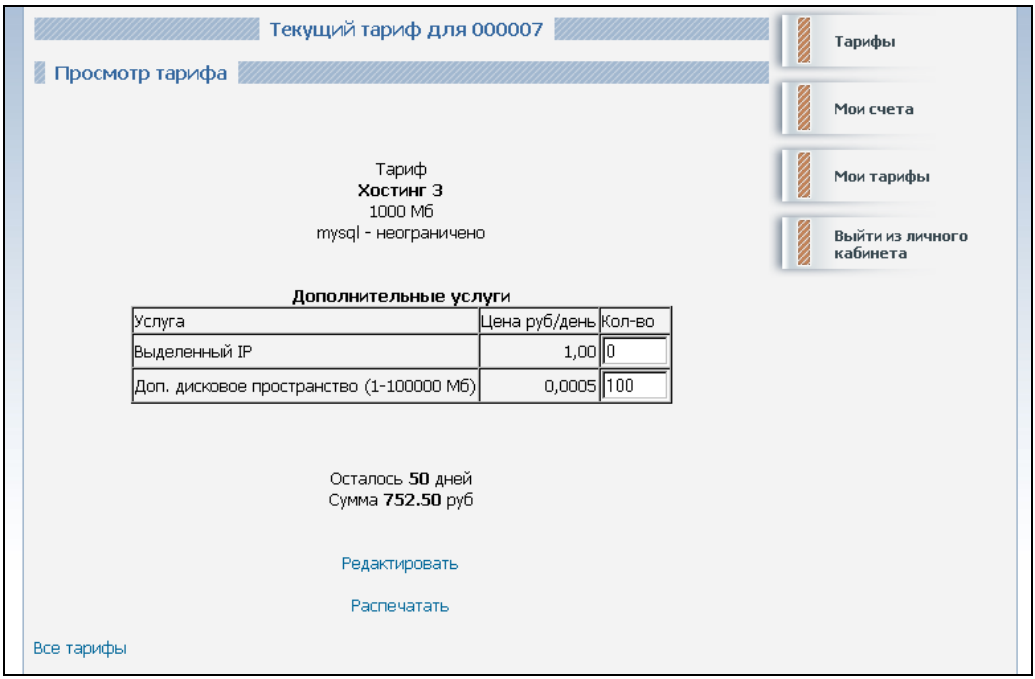


Рис. 6.18. Форма просмотра/редактирования тарифного плана пользователя

Листинг 6.26

```
<?php
// Редактирование тарифа $Id — id_user
function Edit_Client_Tarif($Id)
{ $objResponse = new xajaxResponse();
  // подключение к БД
  require_once("mybaza.php");
  $query00="SELECT * FROM my_schets WHERE id='".$Id."' ";
  $rez00=mysql_query($query00); $row00=mysql_fetch_assoc($rez00);
  $id_user=mysql_result($rez00,0,"id_users");
  $query0="SELECT * FROM my_users WHERE id='".$id_user."' ";
  $rez0=mysql_query($query0); $row0=mysql_fetch_assoc($rez0);
  $l_schet=sprintf("%06s",$row0[id_l_schet]); $id_tarif=$row00[id_tarif];
```

```
// форма подтверждения
$query1="SELECT name,info,pay2 FROM my_tarifs WHERE id='".$id_tarif."' ";
$rez1=mysql_query($query1);
$pay2=$row11[pay2]; $arrpay2=explode(";",$pay2);
$query11="SELECT * FROM my_history WHERE id_schet='".$Id."'
          ORDER BY id DESC LIMIT 0, 1 ";
$rez11=mysql_query($query11); $row11=mysql_fetch_assoc($rez11);
$uslugi_id_tarif=$row11[dop_uslugi];
$arrdopuslugi=explode(";",$uslugi_id_tarif);
$summa=$row11[summa]; $days=$row11[days];
// формирование контента
$text1<form name='Form_Edit_Client_Tarif' id='Form_Edit_Client_Tarif'
     action='javascript:void(null);'onsubmit='
     xajax.$(\"Button_Form_Edit_Client_Tarif\").disabled=true;
     xajax.$(\"Button_Form_Edit_Client_Tarif\").value=\"Подождите...\";
    xajax_Add_Edit_Client_Tarif
    (xajax.getFormValues(\"Form_Edit_Client_Tarif\"));'>
    <input type='hidden' name='data_start'
    value='".$row11[data_start]."' >
    <input type='hidden' name='data_end' value='".$row11[data_end]."' >
    <input type='hidden' name='visible' value='".$row11[visible]."' >
    ... ... ...
    Вы выбрали тариф <br>
    <b>".mysql_result($rez1,0,"name")."</b><br>
     ".str_replace(";","<br>",mysql_result($rez1,0,"info"))."<br>";
 $text1.="Кол-во дней <input type='text' name='days' value='".$days."'
         size=3 maxlength=3
         onchange='xajax_Calculator2(xajax.getFormValues(
           \"Form_Edit_Client_Tarif\"))'>";
 $query2="SELECT * FROM my_tarifs WHERE activ='yes' &&
        id_tarif='".$id_tarif."' ORDER BY sort ASC ";
 $rez2=mysql_query($query2);
 $text1.="<br><b>Дополнительные  услуги</b><br>";
 $text1.="<table cellspacing='0' cellpadding='1' align='center'
          border='1'><tr><td>Услуга</td><td>Цена руб/день</td>
              <td></td><td>Кол-во</td></tr>";
 while($row2=mysql_fetch_assoc($rez2))
 { $i++;$text1.="<tr>";
   $arrsum=explode(";",$row2[pay2]); $sum=$arrsum[1];
   $check=$arrdopuslugi[$i]; $arrcheck=explode("-",$check);
   if($arrcheck[1]=='0')
     {$kol=1;$checked="";}
   else
     {$kol=$arrcheck[1];$checked="checked";}
   $text1.="<td>".$row2[name]."</td><td align=right>".$sum."</td>
            <td><input type=checkbox name='usluga".$i."'
            value='".$row2[id]."' ".$checked."
```

```
            onchange='xajax_Calculator2(xajax.getFormValues(
            \"Form_Edit_Client_Tarif\"))'></td>
            <td><input type=text name='kol".$i."' value='".$kol."'
            size=4 maxlength=6
            onchange='xajax_Calculator2(xajax.getFormValues(
           \"Form_Edit_Client_Tarif\"))'></td></tr>";
  }
  $text1.="</table><center><br><br><br><div id='calc1_days'>
           Осталось <b>".$days."</b> дней</div>
          <div id='calc1_summa'>Сумма
          <b>".sprintf("%10.2f",$summa)."</b> руб</div>";
  $query3="SELECT oplata_podkl FROM my_users WHERE id='".$id_user."' ";
  $rez3=mysql_query($query3); $prob=mysql_result($rez3,0);
  $text1.="<input type='submit' id='Button_Form_Edit_Client_Tarif'
           value='Подтвердить ->'></form>
          <a href='javascript:void();'
          onclick='xajax_View_Client_Tarif(".$Id.");'>
          Отменить</a></center>";
 // вывод контента
   ... ... ...
 return $objResponse;
}
?>
```

При изменении значений в полях вызывается xAjax-функция `Calculator1()`, которую мы рассмотрели в *разд. 6.7*. После сохранения изменений тарифного плана (xAjax-функция `Add_Edit_Client_Tarif()`) происходит запись в таблицу `my_users` данных текущего либо будущего плана, запись данных в таблицу `my_history` и, в случае изменения текущего тарифного плана, вызов процедуры, устанавливающей новые параметры через API ISPmanager *(см. разд. 6.14)*.

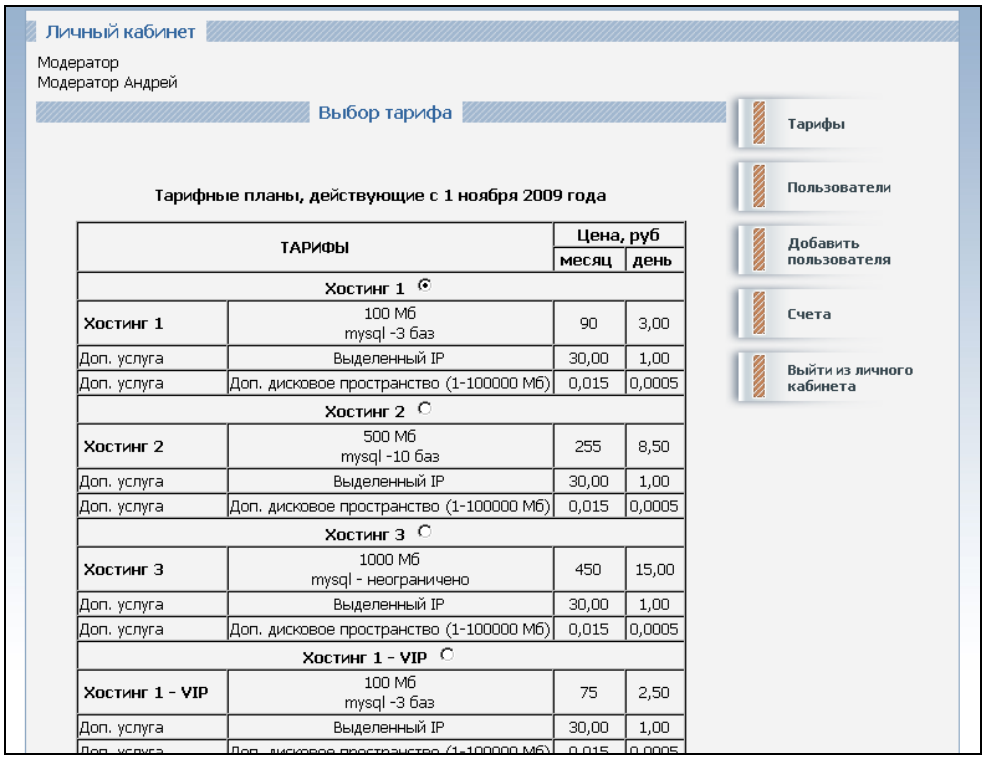
Рассмотрим теперь функции профиля администратора.

6.10. Меню администратора

Вход в панель администратора осуществляется с формы входа пользователя (см. рис. 6.1). Если в таблице `my_users` для пользователя поле `type2=8`, вызывается профиль администратора. Меню для профиля администратора представлено на рис. 6.19.

Пункты главного меню для профиля администратора следующие:

- **Тарифы** — список всех тарифных планов;
- **Пользователи** — просмотр по фильтру и редактирование всех пользователей, их профилей и тарифных планов;
- **Добавить пользователя** — добавление нового пользователя;



ТАРИФЫ		Цена, руб	
		месяц	день
Хостинг 1			
Хостинг 1	100 Мб mysql -3 баз	90	3,00
Доп. услуга	Выделенный IP	30,00	1,00
Доп. услуга	Доп. дисковое пространство (1-100000 Мб)	0,015	0,0005
Хостинг 2			
Хостинг 2	500 Мб mysql -10 баз	255	8,50
Доп. услуга	Выделенный IP	30,00	1,00
Доп. услуга	Доп. дисковое пространство (1-100000 Мб)	0,015	0,0005
Хостинг 3			
Хостинг 3	1000 Мб mysql - неограничено	450	15,00
Доп. услуга	Выделенный IP	30,00	1,00
Доп. услуга	Доп. дисковое пространство (1-100000 Мб)	0,015	0,0005
Хостинг 1 - VIP			
Хостинг 1 - VIP	100 Мб mysql -3 баз	75	2,50
Доп. услуга	Выделенный IP	30,00	1,00

Рис. 6.19. Меню профиля администратора *(справа)*

- **Счета** — просмотр по фильтру всех созданных счетов;
- **Выйти из личного кабинета** — выход из профиля на форму авторизации.

Пункт **Тарифы**, думаю, понятен. Это просмотр всех тарифных планов. **Добавить пользователя** — аналогично регистрации пользователя. Рассмотрим пункт **Счета**.

6.11. Просмотр счетов

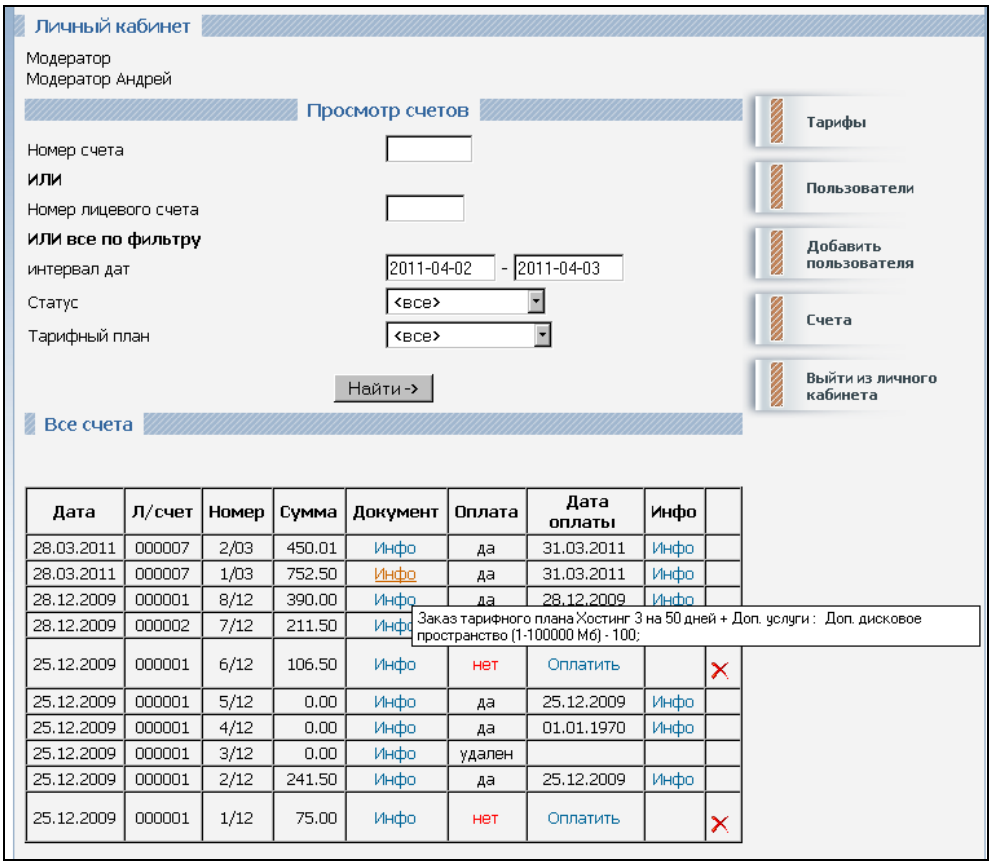
При выборе пункта главного меню **Счета** администратор попадает на страницу просмотра всех счетов (рис. 6.20).

На странице присутствует форма поиска (фильтр) нужных счетов. Возможен поиск по номеру счета, по лицевому счету, в интервале дат, по статусу счета (оплаченные, неоплаченные, удаленный) и по тарифному плану. Функция `function_form_search_all_schets()` формирует контент формы поиска. Ее содержимое представлено в листинге 6.27.

Листинг 6.27

```
<?php
// Форма поиска счетов
function function_form_search_all_schets()
```

```
{ require_once("my.php");require_once("mybaza.php");
  // формирование контента
  $text1.="<form id='Form_Search_All_Schets'
           action='javascript:void(null);' onsubmit='
           xajax.$(\"Button_Form_Search_All_Schets\").disabled=true;
           xajax.$(\"Button_Form_Search_All_Schets\").value=\"Подождите...\";
           xajax_View_Search_All_Schets(xajax.getFormValues(
           \"Form_Search_All_Schets\"));'>";
  $text1.="<table width=100%>";
  $text1.="<tr><td width=50%>Номер счета</td>";
  $text1.="<td width=50% align='left'>
          <input type='text' name='number' value='' size=7 maxlength=7>
          <input type='hidden' id='pagesearch' name='pagesearch'
          value='1'></td></tr>";
  $text1.="<tr><td width=50%><b>ИЛИ</b></td></td></tr>";
  $text1.="<tr><td width=50%>Номер лицевого счета</td>";
  $text1.="<td width=50% align='left'>
          <input type='text' name='l_schet' value='' size=6 maxlength=6>
          </td></tr>";
  $text1.="<tr><td width=50%><b>ИЛИ все по фильтру</b></td></td></tr>";
  $text1.="<tr><td width=50%>интервал дат</td>";
  $text1.="<td width=50% align='left'>
          <input type='text' name='datazakaz1' id='datazakaz1'
           value='".date('Y-m-d')."' size=10 maxlength=10> -
          <input type='text' name='datazakaz2' id='datazakaz2'
           value='".date('Y-m-d',strtotime('now +1 day'))."' size=10
           maxlength=10></td></tr>";
  $text1.="<tr><td width=50%>Статус</td>";
  $text1.="<td width=50% align='left'><select name=oplata><option
            value='0' selected><все>
           <option value='yes'>оплаченные<option value='no'>неоплаченные
           <option value='del'>удаленные</select></td></tr>";
  $text1.="<tr><td width=50%>Тарифный план</td>";
  $text1.="<td width=50% align='left'><select name=id_tarif>
           <option value='0' selected><все>";
  $query1="SELECT id, name FROM my_tarifs WHERE activ='yes' && id<100
           ORDER BY id ASC";
  $rez1=mysql_query($query1);
  while($row1=mysql_fetch_assoc($rez1))
  { $text1.="<option value='".$row1[id]."'>".$row1[name]; }
  $text1.="</select></td></tr></table><center><br>
           <input type='submit' id='Button_Form_Search_All_Schets'
           value='Найти ->'></center></form>";
  return $text1;
}
?>
```



Дата	Л/счет	Номер	Сумма	Документ	Оплата	Дата оплаты	Инфо	
28.03.2011	000007	2/03	450.01	Инфо	да	31.03.2011	Инфо	
28.03.2011	000007	1/03	752.50	Инфо	да	31.03.2011	Инфо	
28.12.2009	000001	8/12	390.00	Инфо	да	28.12.2009	Инфо	
28.12.2009	000002	7/12	211.50	Инфо				
25.12.2009	000001	6/12	106.50	Инфо	нет	Оплатить		
25.12.2009	000001	5/12	0.00	Инфо	да	25.12.2009	Инфо	
25.12.2009	000001	4/12	0.00	Инфо	да	01.01.1970	Инфо	
25.12.2009	000001	3/12	0.00	Инфо	удален			
25.12.2009	000001	2/12	241.50	Инфо	да	25.12.2009	Инфо	
25.12.2009	000001	1/12	75.00	Инфо	нет	Оплатить		

Рис. 6.20. Просмотр счетов из профиля администратора

xAjax-функция представления результатов поиска счетов `View_Search_All_Schets()` выводит результат постранично в табличной форме. Для каждого счета выдается следующая информация:

- дата создания счета;
- номер счета;
- номер лицевого счета пользователя, создавшего счет;
- сумма;
- ссылка, позволяющая просмотреть (распечатать) счет или квитанцию, и всплывающая подсказка о заказанном в счете тарифном плане и дополнительных услугах;
- статус оплаты (оплачен, не оплачен, удален);
- дата оплаты счета или ссылка на подтверждение оплаты;
- информация по оплате (всплывающая подсказка);
- ссылка на удаление счета.

Содержимое функции `View_Search_All_Schets()` представлено в листинге 6.28. Содержимое функции `function_view_search_all_schets()`, формирующей контент результатов поиска, — в листинге 6.29.

Листинг 6.28

```
<?php
// Просмотр всех счетов по фильтру (администратор)
function View_Search_All_Schets($Id)
{ $objResponse = new xajaxResponse();
  // формирование контента результатов поиска
  $text3=function_view_search_all_schets($Id);
  // вывод контента
  $text2="<div class='bghr' align='left'><font class='blocktitle'>
  Счета по фильтру </font></div>";
  $objResponse->assign("center2","innerHTML",$text2);
  $objResponse->assign("center3","innerHTML",$text3[0]);
  $objResponse->assign("center4","innerHTML",$text3[1]);
  $objResponse->assign("center5","innerHTML","<table></table>");
  $objResponse->assign("center6","innerHTML","<table></table>");
  $objResponse->assign("Button_Form_Search_All_Schets","value","Найти ->");
  $objResponse->assign("Button_Form_Search_All_Schets","disabled", false);
  return $objResponse;
}
?>
```

Листинг 6.29

```
<?php
// Просмотр всех счетов (модератор)
function function_view_search_all_schets($Id)
{ $text=array();
  require_once("my.php");
  require_once("mybaza.php");
  // формирование запроса в зависимости от данных формы поиска
  $query0="SELECT my_schets.id FROM my_schets WHERE my_schets.id>0 ";
  if(strlen($Id[number])>0)
  { $query0.="&& my_schets.number='".$Id[number]."' "; }
  elseif(strlen($Id[l_schet])>0)
  { $query0="SELECT my_schets.id FROM my_schets,my_users
             WHERE my_schets.id>0 ";
    $query0.=" && my_schets.id_users=my_users.id
               && my_users.id_l_schet='".$Id[l_schet]."' ";
  }
  else
  { $query0.="&& my_schets.data >= '".$Id[datazakaz1]."' &&
             my_schets.data <='".$Id[datazakaz2]."'  ";
```

```
    if($Id[oplata]<>'0')
    { $query0.="&& my_schets.oplata='".$Id[oplata]."' "; }
    if($Id[id_tarif]>0)
    { $query0.="&& my_schets.id_tarif='".$Id[id_tarif]."' "; }
  }
  // запрос к базе данных, получение количества страниц результата
  $rez0=mysql_query($query0); $count=mysql_num_rows($rez0);
  $pages=ceil($count/NN1);
  $page=min($Id[pagesearch],$pages);$poz=($page-1)*NN1;
  if(mysql_num_rows($rez0)>0)
  { // получение результатов для выбранной страницы
    $query0.=" ORDER BY my_schets.data DESC LIMIT ".$poz.", ".NN1."";
    $rez0=mysql_query($query0);
    // заголовок таблицы
    ... ... ...
    // данные строк таблицы результатов
    while($row0=mysql_fetch_row($rez0))
    { $query1="SELECT * FROM my_schets WHERE id='".$row0[0]."' ";
      $rez1=mysql_query($query1);
      $row1=mysql_fetch_assoc($rez1);
      ... ... ...
      $text1.="<td align='center'> <a href='javascript:void();'
                onclick='xajax_Oplata_Schet_Moderator1(".$row1[id].");'>
                Оплатить</a> </td>";
      $text1.="<td align='center'>  </td>";
      $text1.="<td align='center'> <a href='javascript:void();'
                onclick='xajax_Delete_All_Schet(".$row1[id].");'
                title='Удалить'>
                <img border=no src='images/delete.png'></a> </td>";
      }
      else
      { $text1.="<td align='center'> удален </td>";
        $text1.="<td align='center'>  </td>";
        $text1.="<td align='center'>  </td>";
        $text1.="<td align='center'>  </td>";
      }
      $text1.="</tr>";
    }
    $text1.="</table></center><br><br>";
    // список ссылок перехода по страницам
    ... ... ...
  }
  else
  { $text1="<br><br><br><br><center>Поиск не дал результата</center>";
    $text2="";}
  $text[0]=$text1;$text[1]=$text2;
  return $text;
}
?>
```

6.12. Подтверждение оплаты счета администратором

При оплате через Сбербанк или счет (для юридического лица) необходимо подтверждение оплаты, которое совершает модератор. Оплата модератором происходит из просмотра всех счетов пользователей (см. рис. 6.20). При щелчке по ссылке **Оплатить** происходит вызов xAjax-функции `Oplata_Schet_Moderator1()`, создающей форму подтверждения (рис. 6.21), где можно внести дополнительную информацию об оплате. Содержимое xAjax-функции `Oplata_Schet_Moderator1()` представлено в листинге 6.30.

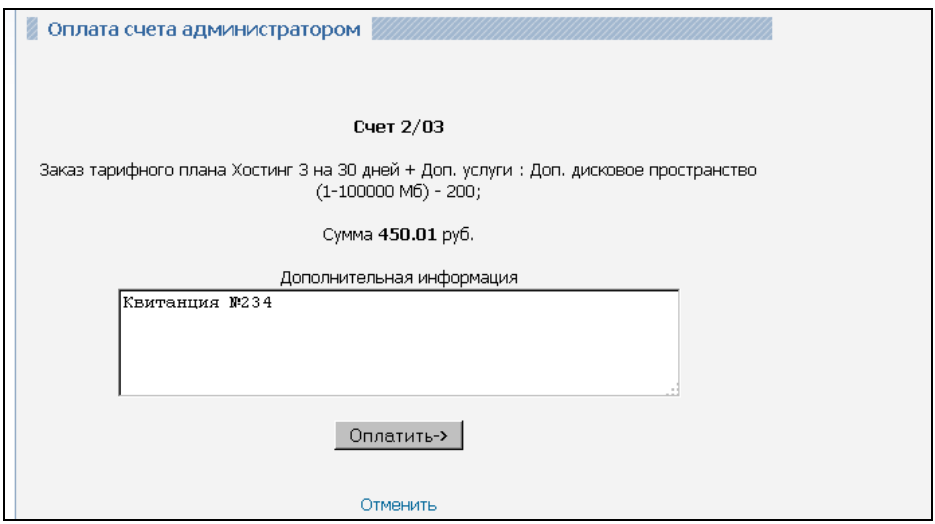


Рис. 6.21. Подтверждение оплаты счета администратором

Листинг 6.30

```
<?php
// Оплата счета администратором
function Oplata_Schet_Moderator1($Id)
{ $objResponse = new xajaxResponse();
  // подключение к базе данных
  require_once("mybaza.php");
  // получение данных счета
  $query1="SELECT * FROM my_schets WHERE id='".$Id."' ";
  $rez1=mysql_query($query1);
  $row1=mysql_fetch_assoc($rez1);
  // создание формы
  $text2="<center><br><br>
  <form name='Form_Oplata_Schet_Moderator'
     id='Form_Oplata_Schet_Moderator'
```

```
    action='javascript:void(null);' onsubmit='
    xajax.$(\"Button_Form_Oplata_Schet_Moderator\")
    .disabled=true;
    xajax.$(\"Button_Form_Oplata_Schet_Moderator\")
    .value=\"Подождите...\";
    xajax_Oplata_Schet_Moderator2(xajax.getFormValues(
    \"Form_Oplata_Schet_Moderator\"));'>
    <input type='hidden' name='id_schet' value='".$Id."'>
    <br>
    <b>Счет ".$row1[number]."</b><br><br>
    ".$row1[info]."<br><br>
    Сумма  <b>".$row1[summa]."</b> руб.<br>
    <input type='hidden' name='summa_oplata'
    value='".$row1[summa]."' size=10 maxlength=10>
    <br>
    Дополнительная информация
    <textarea name='info_oplata' value='Информация по оплате'
    cols=50 rows=4>
    Информация по оплате</textarea>
    <br><br>
    <input type='submit' id='Button_Form_Oplata_Schet_Moderator'
    value='Оплатить->'>
    </form>
    <br><br>
    <a href='javascript:void();' onclick='
    document.getElementById(\"center5\").innerHTML=\"\";
    document.getElementById(\"center6\").innerHTML=\"\";
    '>Отменить</a>
    <br><br><br><br>
    </center>";
  // вывод контента
  $text1="<div class='bghr' align='left'><font class='blocktitle'>
          Оплата счета администратором</font></div>";
  $objResponse->assign("center5","innerHTML",$text1);
  $objResponse->assign("center6","innerHTML",$text2);
  $objResponse->script("document.getElementById('center5')
              .scrollIntoView();");
  return $objResponse;
}
?>
```

При нажатии кнопки **Оплатить** происходит вызов xAjax-функции `Oplata_Schet_Moderator2()`, которая выполняет следующие действия:

- проверяет наличие у пользователя тарифного плана следующего периода (если есть — возврат);
- если есть текущий тарифный план, записывает в таблицу `my_users` данные как тарифный план следующего периода;

- если нет тарифных планов, записывает в таблицу `my_users` данные как текущий тарифный план и вызывает программу API ISPmanager *(см. разд. 6.14)*;
- делает записи в таблицу `my_history`;
- выводит заново форму поиска и обновленную таблицу счетов пользователей.

xAjax-функция `Oplata_Schet_Moderator1()` представлена в листинге 6.31.

Листинг 6.31

```
// Оплата счета администратором
function Oplata_Schet_Moderator2($Id)
{ $objResponse = new xajaxResponse();
  // подключение к базе данных
  require_once("mybaza.php");
  // если есть не только текущий, но и следующий — возврат
  $query0="SELECT my_users.id_tarif_sled FROM my_users,my_schets
           WHERE my_schets.id='".$Id[id_schet]."' &&
           my_schets.id_users=my_users.id  ";
  $rez0=mysql_query($query0);
  ... ... ...
  //  установить счет на оплату — оплаченный
  $query1="UPDATE my_schets SET summa_oplata='".$Id[summa_oplata]."',
           oplata='yes',data_oplata='".date('Y-m-d H:i:s')."',
           info_oplata='".utftowin($Id[info_oplata])."'
           WHERE id='".$Id[id_schet]."' ";
  $rez1=mysql_query($query1);
  ... ... ...
  else
  { // изменить информацию о пользователе
    $query2="SELECT * FROM my_schets WHERE id='".$Id[id_schet]."' ";
    $rez2=mysql_query($query2);
    $row2=mysql_fetch_assoc($rez2);
    $query4="SELECT * FROM my_users WHERE id='".$row2[id_users]."' ";
    $rez4=mysql_query($query4); $row4=mysql_fetch_assoc($rez4);
    if($row4[id_tarif]>0) // текущий есть — установить следующий
    { $data_start=$row4[data_end];
      $query3="UPDATE my_users SET id_tarif_sled='".$row2[id_tarif]."',
               schet_id_tarif_sled='".$Id[id_schet]."',oplata_podkl='yes',
               days_id_tarif_sled='".$row2[days_id_tarif]."',
               uslugi_id_tarif_sled='".$row2[dop_uslugi]."'
               WHERE id='".$row2[id_users]."' ";
         $rez3=mysql_query($query3);
         ... ... ...
      // запись в таблицу my_history
      $text2=function_create_history2($row4[id]);
    }
```

```
    else  // установить текущий
    { $data_start=date('Y-m-d H:i:s'); $data_end=date('Y-m-d H:i:s',
      strtotime('now+'.$row2[days_id_tarif].' days'));
      $query3="UPDATE my_users SET id_tarif='".$row2[id_tarif]."',
              schet_id_tarif='".$Id[id_schet]."',oplata_podkl='yes',
              days_id_tarif='".$row2[days_id_tarif]."',
              uslugi_id_tarif='".$row2[dop_uslugi]."',
              data_start='".$data_start."', data_end='".$data_end."',
              oplata_period='yes' WHERE id='".$row2[id_users]."' ";
         $rez3=mysql_query($query3);
      ... ... ...
      // запись в таблицу my_history
      $text2=function_create_history($row4[id]);
      // установка параметров через API ISPmanager
      $text2=function_api($row4[id]);
    }
    // отправка на e-mail
    ... ... ...
  }
  // контент формы поиска
  $text1=function_form_search_all_schets();
  // контент таблицы всех счетов
  $text3=function_view_all_schets(1);
  // вывод контента
  $objResponse->assign("zag1","innerHTML",
     "<font class='blocktitle'> Просмотр счетов </font>");
  $objResponse->assign("center1","innerHTML",$text1);
  $text2="<div class='bghr' align='left'><font class='blocktitle'>
      Все счета </font></div>";
  $objResponse->assign("center2","innerHTML",$text2);
  $text2="<div class='bghr' align='left'><font class='blocktitle'>
      Все счета </font></div>";
  $objResponse->assign("center3","innerHTML",$text3[0]);
  $objResponse->assign("center4","innerHTML",$text3[1]);
  $objResponse->assign("center5","innerHTML","<table></table>");
  $objResponse->assign("center6","innerHTML","<table></table>");
  return $objResponse;
}
?>
```

6.13. Просмотр и редактирование профилей пользователей и их тарифных планов

При выборе пункта главного меню **Пользователи** администратор попадает на страницу просмотра всех счетов (рис. 6.22).

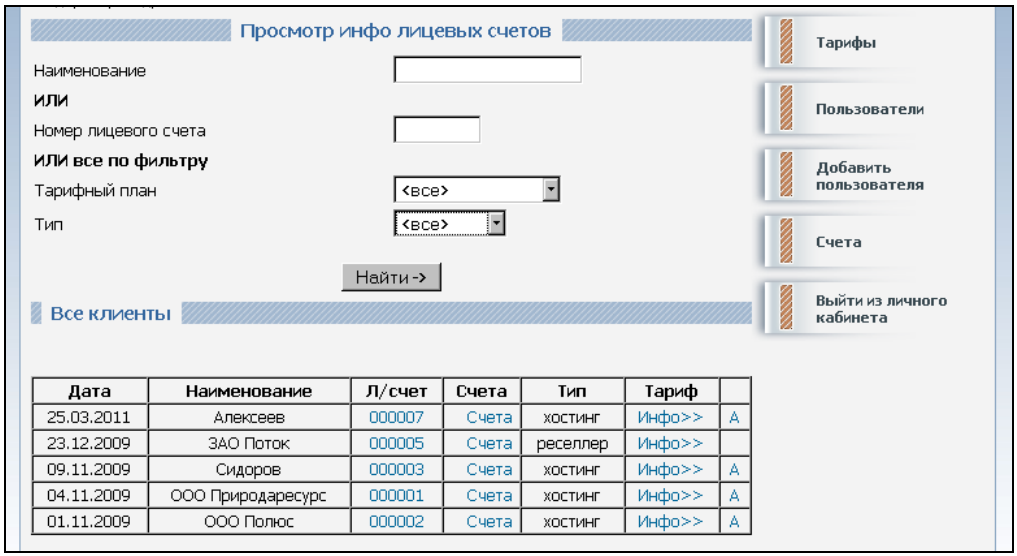


Рис. 6.22. Просмотр всех пользователей из профиля администратора

На странице присутствует форма поиска (фильтр) пользователей. Возможен поиск пользователей по наименованию, номеру лицевого счета, по типу (хостинг, реселлер) и по заказанному пользователем тарифному плану. Функция `function_form_search_all_users()` формирует контент формы поиска. Ее содержимое представлено в листинге 6.32.

Листинг 6.32

```
<?php
// Форма поиска пользователей
function function_form_search_all_users()
{ require_once("my.php");
  // подключение к базе данных
  require_once("mybaza.php");
  // формирование контента формы
  $text1.="<form id='Form_Search_All_Users'
           action='javascript:void(null);' onsubmit='
           xajax.$(\"Button_Form_Search_All_Users\").disabled=true;
      xajax.$(\"Button_Form_Search_All_Users\").value=\"Подождите...\";
           xajax_View_Search_All_Users(xajax.getFormValues(
           \"Form_Search_All_Users\"));'>";
  $text1.="<table width=100%>";
  $text1.="<tr><td width=50%>Наименование</td>";
  $text1.="<td width=50% align='left'>
          <input type='text' name='name' value='' size=20 maxlength=20>
          <input type='hidden' id='pagesearch' name='pagesearch'
          value='1'></td></tr>";
```

```
  $text1.="<tr><td width=50%><b>ИЛИ</b></td></td></tr>";
  $text1.="<tr><td width=50%>Номер лицевого счета</td>";
  $text1.="<td width=50% align='left'>
          <input type='text' name='l_schet' value='' size=7 maxlength=7>
          </td></tr>";
  $text1.="<tr><td width=50%><b>ИЛИ все по фильтру</b></td></td></tr>";
  $text1.="<tr><td width=50%>Тарифный план</td>";
  $text1.="<td width=50% align='left'><select name=id_tarif><option
           value='0' selected><все>";
  $query1="SELECT id,name FROM my_tarifs WHERE activ='yes' && id<100
           ORDER BY id ASC";
  $rez1=mysql_query($query1);
  while($row1=mysql_fetch_assoc($rez1))
  { $text1.="<option value='".$row1[id]."'>".$row1[name]; }
  $text1.="</select></td></tr>";
  $text1.="<tr><td width=50%>Тип</td>";
  $text1.="<td width=50% align='left'><select name=type2>
           <option value='0' selected><все>
           <option value='1'> хостинг<option value='2'> реселлер
           </select></td></tr</table>";
  $text1.="<center><br>
           <input type='submit' id='Button_Form_Search_All_Users'
           value='Найти ->'></center></form>";
  return $text1;
}
?>
```

xAjax-функция представления результатов поиска счетов `View_Search_All_Users()` выводит результат постранично в табличной форме. Для каждого пользователя выдается следующая информация:

- дата регистрации пользователя;
- наименование пользователя;
- номер лицевого счета пользователя, по ссылке переходим на просмотр (редактирование) данных пользователя;
- ссылка для просмотра всех счетов пользователя;
- тип профиля пользователя (хостинг, реселлер);
- ссылка для просмотра всех тарифных планов пользователя (рис. 6.23) и информация о действующем тарифе (всплывающая подсказка);
- ссылка на активацию тарифного плана (если произошел сбой при автоматическом запуске тарифа, существует эта возможность принудительно заново запустить активацию тарифного плана через вызов функций API ISPmanager).

Содержимое функции `View_Search_All_Users()` представлено в листинге 6.33. Содержимое функции `function_view_search_all_users()`, формирующей контент результатов поиска, — в листинге 6.34.

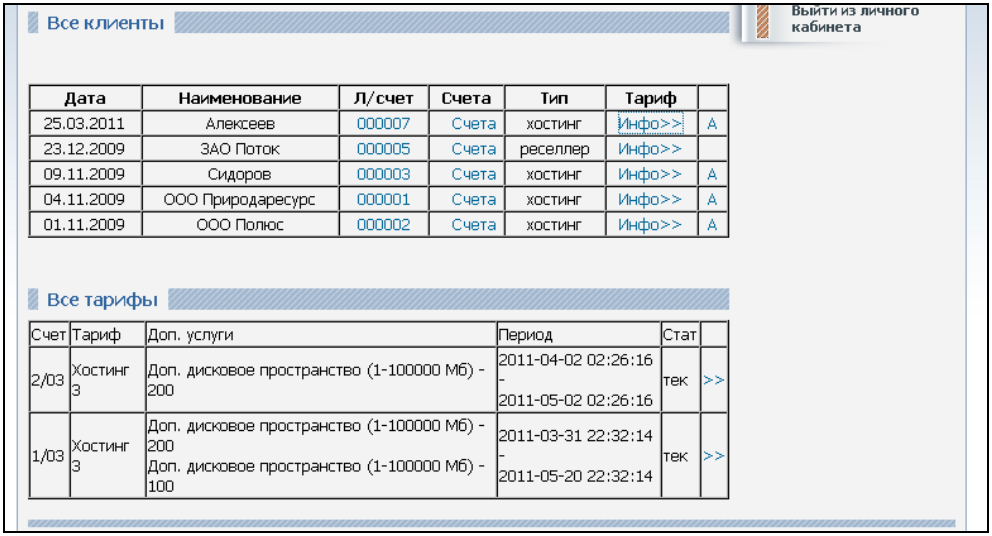


Рис. 6.23. Просмотр всех тарифных планов выбранного пользователя

Листинг 6.33

```
<?php
// Просмотр всех пользователей по фильтру (администратор)
function View_Search_All_Users($Id)
{ $objResponse = new xajaxResponse();
  // формирование контента результатов поиска
  $text3=function_view_search_all_users($Id);

  // вывод контента
  $text2="<div class='bghr' align='left'><font class='blocktitle'>
      Фильтр пользователей </font></div>";
  $objResponse->assign("center2","innerHTML",$text2);
  $objResponse->assign("center3","innerHTML",$text3[0]);
  $objResponse->assign("center4","innerHTML",$text3[1]);
  $objResponse->assign("center5","innerHTML","<table></table>");
  $objResponse->assign("center6","innerHTML","<table></table>");
  $objResponse->assign("Button_Form_Search_All_Users","value","Найти ->");
  $objResponse->assign("Button_Form_Search_All_Users","disabled", false);
  return $objResponse();
}
?>
```

Листинг 6.34

```
<?php
// Просмотр всех пользователей по фильтру (администратор)
function function_view_search_all_users($Id)
{ $text=array();
```

```
// подключение к базе данных
require_once("mybaza.php");require_once("my.php");
// формирование запроса в зависимости от данных формы поиска
$query0="SELECT DISTINCT(my_users.id)
         FROM my_users WHERE my_users.type<'8' ";
if(strlen($Id[name])>0)
{ $str1="FROM my_users";
  $str2="FROM my_users,my_users_info1,my_users_info2,my_users_info3";
  $query0=str_replace($str1,$str2,$query0);
  $query0.="&& ((my_users_info1.phone LIKE '%".utftowin($Id[name])."%'
       && my_users.id_info=my_users_info1.id && my_users.type='1') ";
  $query0.=" || (my_users_info2.name LIKE '%".utftowin($Id[name])."%'
    && my_users.id_info=my_users_info2.id && my_users.type='2') ";
  $query0.=" || (my_users_info3.name1 LIKE '%".utftowin($Id[name])."%'
    && my_users.id_info=my_users_info3.id && my_users.type='3')) ";
}
if(strlen($Id[l_schet])>0)
{ $query0.="&& my_users.id_l_schet='".$Id[l_schet]."' "; }
else
{ if($Id[type2]>0)
  { $query0.="&& my_users.type2='".$Id[type2]."' "; }
  if($Id[id_tarif]>0)
  { $query0.="&& my_users.id_tarif='".$Id[id_tarif]."' "; }
}
// получение количества страниц результата
$rez0=mysql_query($query0); $count=mysql_num_rows($rez0);
$pages=ceil($count/NN2);
$page=min($Id[pagesearch],$pages);$poz=($page-1)*NN2;
if(mysql_num_rows($rez0)>0)
{ // получение результатов для выбранной страницы
  $query0.=" ORDER BY my_users.data_reg DESC LIMIT ".$poz.", ".NN2."";
  $rez0=mysql_query($query0);
  // заголовок таблицы
  ... ... ...
  // данные строк таблицы результатов
  while($row0=mysql_fetch_row($rez0))
  { $query1="SELECT * FROM my_users WHERE id='".$row0[0]."' ";
    $rez1=mysql_query($query1);
    $row1=mysql_fetch_assoc($rez1);
    // вывод данных таблицы
    ... ... ...
    // ссылка на просмотр тарифов
    $text1.="<td align='center'> <a href='javascript:void();'
      onclick='xajax_View_All_Tarif_User(".$row1[id].");'
      title='".$info."'>Инфо>></a> </td>";
    // ссылка активации
    if($row1[id_tarif]>0)
```

```
        $text1.="<td align='center'> <a title='Активировать'
         href='javascript:void()'
         onclick='xajax_Activ_SSH(".$row1[id].")'>A</a> </td>";
    ... ... ...
    }
    $text1.="</table></center><br><br>";
    // список ссылок перехода по страницам
    ... ... ...
  }
    ... ... ...
  $text[0]=$text1; $text[1]=$text2;
  return $text;
}
?>
```

6.14. Функция активации тарифа с использованием API ISPmanager

Созданный нами личный кабинет — это всего лишь веб-оболочка профиля клиента. Для создания реального профиля на веб-сервере необходимо передать эти данные в программу упраления хостингом, в нашем случае — панель управления хостингом ISPmanager. Для этого будем использовать обращение к функциям API ISPmanager, которое мы подробно рассмотрели в *главе 5*.

При регистрации клиента не происходит создания профиля в ISPmanager. Профиль в панели создается при первичной оплате тарифного плана. Также выполняется создание и активация тарифного плана. Обращение к функциям API ISPmanager необходимо при изменении текущего тарифного плана, окончании времени его действия, автоматической активации тарифного плана следующего периода, ручном запуске активации тарифного плана из профиля администратора. Для всех этих случаев мы создадим одну функцию `function_api()`, в качестве аргумента будем передавать идентификатор тарифного плана, который нам нужно активировать.

Рассотрим алгоритм действия функции `function_api()`. Функция выполняет следующие действия.

По входящему `id_user` получаем параметры данного пользователя, в частности номер лицевого счета. Получая запросом к API список всех пользователей из панели управления *(см. разд. 5.1.5)*:

`http://ISP_SERVER/manager/ispmgr?authinfo=`*логин*`:`*пароль*`&out=text&func=user`

или

`http://ISP_SERVER/manager/ispmgr?authinfo=`*логин*`:`*пароль*`&out=text&func=reseller`

проверяем существование пользователя `nf_лицевой счет`. Если пользователь не существует — создаем, если не активирован — активируем, иначе — изменяем данные. Данные тарифного плана берутся из таблицы `my_tarifs`.

Далее — дополнительные услуги. Возможен заказ двух дополнительных услуг:

- IP-адреса;
- дополнительного дискового пространства.

Получаем список всех доступных IP-адресов и список IP-адресов пользователя *(см. разд. 5.1.22)*:

```
http://ISP_SERVER/manager/ispmgr?authinfo=логин:пароль&out=text&func=iplist
```

Из возвращенного сервером ответа получаем список IP-адресов пользователя.

Если количество IP-адресов отличается от полученного, тогда либо добавляем (привязываем) дополнительные IP к пользователю (из списка свободных):

```
http://ISP_SERVER/manager/ispmgr?authinfo=логин:пароль&out=text."&func=iplist.edit&sok=yes&elid=адрес&name=адрес&stat=assigned&owner=владелец
```

либо освобождаем:

```
http://ISP_SERVER/manager/ispmgr?authinfo=логин:пароль&out=text."&func=iplist.edit&sok=yes&elid=адрес&name=адрес&stat=free
```

Дополнительное дисковое пространство добавляем к положенному по тарифному плану и устанавливаем новое значение для дискового пространства.

Содержимое функции `function_api()`, расположенной в файле hostin\my_prg_api\function_api.php, представлено в листинге 6.35.

Листинг 6.35

```
<?php
// Посылка команд по API
// $Id -id_user
function function_isp($Id)
{ // подключить файл настроек, подключить к базе данных
  require_once("my.php");require_once("mybaza.php");
  // запрос на получение данных о тарифном плане пользователя
  $query1="SELECT id_l_schet,password,id_tarif,type2,uslugi_id_tarif
    FROM my_users WHERE id='".$Id."' ";
  $rez1=mysql_query($query1);
  $row1=mysql_fetch_assoc($rez1);
  $name=sprintf("%06s",$row1[id_l_schet]);
  $password=$row1[password];
  $type2=$row1[type2];
  if($type2=='2') $type2='reseller';
  else            $type2='user';
  // получение данных для дополнительных услуг
  $query2="SELECT ssh,info FROM my_tarifs WHERE id='".$row1[id_tarif]."' ";
  $rez2=mysql_query($query2);
  $command=mysql_result($rez2,0,"ssh");
  $info=mysql_result($rez2,0,"info");
  $diskarr=explode(";",$info);
```

```
$disksize=str_replace("Мб","",$diskarr[0]);
$disksize=str_replace(" ","",$disksize);
// шаблон запроса
$str='http://'.ISP_SERVER.'/manager/ispmgr?authinfo=
     '.ISP_LOGIN.':'.ISP_PASS.'&out=text';
// Получить список пользователей.
// Проверить: существует/не существует, отключен/включен
$str11=$str.'&func='.$type2;
$fh = fopen($str11, "r");
... ... ...
}
// получить список всех IP
$str12=$str.'&func=iplist';
... ... ...
// получить список IP пользователя
$str13=$str.'&func=iplist';
... ... ...
// не существует — создать, не активирован — активировать, иначе — изменить
$str2=$str."&func=".$type2.".edit&elid=nf_".$name;
... ... ...
if($ok=='no')
{
... ... ...
  $str2=$str."&func=".$type2.".edit&sok=yes&name=nf_".$name.
    "&passwd=".$password."&ip=".$ip."&disklimit=".$disksize.$command;
... ... ...
}
elseif($status=='no')
{
  ... ... ...
  $str2=$str."&func=".$type2.".edit&sok=yes&elid=nf_".$name.
  "&name=nf_".$name."&ip=".$ip."&disklimit=".$disksize.$command;
  ... ... ...
}
else
{
   $str2=$str."&func=".$type2.".edit&sok=yes&elid=nf_".$name.
     "&name=nf_".$name."&ip=".$ip."&disklimit=".$disksize.$command;
  ... ... ...
}
// дополнительные услуги
$arr_uslugi_id_tarif=explode(";",$row1[uslugi_id_tarif]);
for($i=1;$i<count($arr_uslugi_id_tarif);$i++)
{ $arr2=explode("-",$arr_uslugi_id_tarif[$i]);
  $query2="SELECT ssh FROM my_tarifs WHERE id='".$arr2[0]."' ";
  $rez2=mysql_query($query2); $ssh=mysql_result($rez2,0,"ssh");
  $str3.=" ".$ssh;
```

```
    switch($ssh)
    { case 'ip':
            if($count_ip_user<$arr2[1])
            { for($j=0;$j<$arr2[1]-$count_ip_user;$j++)
              { $ip_add=$arr_ip_free[$j];
                $str4=$str."&func=iplist.edit&sok=yes&
       elid=".$ip_add."&name=".$ip_add." &stat=assigned&owner=nf_".$name;
                ... ... ...
              }
            }
            if($count_ip_user>$arr2[1])
            { for($j=0;$j<$count_ip_user-$arr2[1];$j++)
              { $ip_add=$arr_ip_user[$j];
                $str4=$str."&func=iplist.edit&sok=yes
                &elid=".$ip_add."&name=".$ip_add."&stat=free";
                ... ... ...
              }
            }
            else ;
            break;
      case 'disklimit':
            $disklimit=$arr2[1]+$disksize;
            $str3=$str."&func=".$type2.".edit&sok=yes&elid=nf_".$name."
        &name=nf_".$name."&ip=".$ip." &disklimit=".$disklimit.$command;
            ... ... ...
      default:
            break;
    }
  }
  return $text1;
}
?>
```

6.15. Скрипты, запускаемые по cron. Деактивация аккаунта

Для автоматизации контроля за состоянием тарифных планов пользователей (деактивация тарифного плана, переключение на следующий заказанный тарифный план, отправка уведомлений о сроке окончания тарифного плана) необходимо предусмотреть ежедневный запуск программ по cron. У нас предусмотрены 3 скрипта, запускаемых по cron. Все они располагаются в папке cron нашего сайта. Список скриптов следующий:

- change_tarif.php — отключение тарифного плана по окончании срока действия и, при наличии оплаченного плана следующего периода, активация следующего тарифного плана;

- to_mail.php — отправка на электронный почтовый адрес пользователя уведомлений за несколько дней до окончания срока действия тарифного плана;
- cron_history.php — создание в таблице `my_history` базы данных записи списывания средств за пользование тарифным планом за день.

Рассмотрим скрипт change_tarif.php. Он осуществляет поиск по таблице `my_users` записей с действующим тарифным планом. Если срок действия истек, то обнуляются данные в таблице для текущего тарифного плана и вызывается функция `function_api_out()` для отправки команды отключения пользователя. Далее проверяется наличие оплаченного тарифного плана следующего периода. Если он есть, переписываются его данные в поля для текущего тарифного плана. Поля для тарифного плана следующего периода обнуляются. Осуществляется отправка команд по API для активации нового тарифного плана пользователя. Содержимое скрипта change_tarif.php представлено в листинге 6.36.

Листинг 6.36

```
#!/usr/bin/php
<?php
$data_tek=date('Y-m-d H:i:s');
// подключение к базе данных
require_once("../mybaza.php");
// подключение функции отправки команд по API
require_once("../my_prg_api/function_api.php");
// поиск закончившихся тарифных планов
$query1="SELECT * FROM my_users WHERE id_tarif>0 ";
$rez1=mysql_query($query1);
while($row1=mysql_fetch_assoc($rez1))
{ // проверка окончания действия тарифа
  if($row1[data_end]<$data_tek)
  { // обнулить id_tarif и отправить отключение по API
    $query2="UPDATE my_users SET id_tarif='0',oplata_period='no'
        WHERE id='".$row1[id]."' ";
    $rez2=mysql_query($query2);
    // отключение по API
    echo function_api_out($row1[id]);
    // если оплачен следующий период — изменить
    //  id_tarif=id_tarif_sled
    //  id_tarif_sled=0
    //  data_start,data_end
    if($row1[id_tarif_sled]>0)
    { $id_tarif=$row1[id_tarif_sled];
      $data_start=$row1[data_end];
      $data_end=date('Y-m-d H:i:s',strtotime($data_start." + 30 days"));
      $query3="UPDATE my_users SET
         id_tarif='".$id_tarif."',id_tarif_sled='0',
         data_start='".$data_start."',data_end='".$data_end."',
```

```
          oplata_period='yes'
          WHERE id='".$row1[id]."' ";
       $rez3=mysql_query($query3);
       // подключение следующего тарифного плана по API
       echo function_api($row1[id]);
     }
   }
}
```

Функция `function_api_out()` осуществляет отправку команд API для отключения пользователя. Содержимое функции `function_api_out()`, расположенной в файле hosting\my_prg_api\function_api.php, представлено в листинге 6.37.

Листинг 6.37

```
// Отключение пользователя по API
// $Id -id_user
function function_api_out($Id)
{ require_once("my.php");
  require_once("mybaza.php");
  $query1="SELECT id_l_schet,password,id_tarif,type2,uslugi_id_tarif
          FROM my_users WHERE id='".$Id."' ";
  $rez1=mysql_query($query1);
  $row1=mysql_fetch_assoc($rez1);
  $name=sprintf("%06s",$row1[id_l_schet]);
  $password=$row1[password];
  $type2=$row1[type2];
  if($type2=='2') $type2='reseller';
  else            $type2='user';
  // формирование запроса
  $str='http://'.ISP_SERVER.'/manager/ispmgr?authinfo='.ISP_LOGIN.':
  '.ISP_PASS.'&out=text';
  $str2=$str."&func=".$type2.".disable&elid=nf_".$name;
  // отправка команды
  $fh = fopen($str11, "r");
  fclose($fh);
  return;
}
?>
```

Все файлы данного проекта вы можете найти на прилагаемом к книге компакт-диске в папке glava_06.

ГЛАВА 7

Google,Twitter и другие сервисы

Количество веб-сервисов, имеющих свои API, огромно. Рассмотрим некоторые из них, наиболее популярные и известные, и обсудим примеры их использования.

7.1. API сервисов Google

Мало кто сомневается в том, что Google доминирует в мире информационных интернет-сервисов. На долю компании приходится 57% оборота мирового рынка поиска. Но Google — это не только поиск. Кроме обычного поиска, Google предлагает множество сервисов и инструментов для различных нужд. Большинство из них — веб-приложения, требующие от пользователя только наличия браузера, в котором они работают, и интернет-подключения. Это позволяет использовать данные в любой точке планеты и не быть привязанным к одному компьютеру. Google, являясь одним из крупнейших разработчиков веб-приложений, совсем не собирается скрывать свои достижения и наиболее успешные разработки от сторонних программистов, а, наоборот, старается поделиться исходным кодом. Кроме уже готовых приложений компания предоставляет множество API, с помощью которых любой веб-разработчик может усовершенствовать свой сайт, связать его с какой-либо службой Google.

На момент написания книги доступно более 60 публичных API-сервисов Google. Приведу только некоторые из них.

- **API Google AdWords** — сервис, взаимодействующий с рекламной системой Google — AdWords. Цели программирования с использованием AdWords API — это автоматизация определения ключевых слов и рекламных сообщений для сайта, интеграция с базами данных, разработка дополнительных инструментов для работы с аккаунтом рекламной системы. Для работы с API можно применять любые из языков: Java, .NET, Perl, PHP, Python, OCAML, Ruby или XML. Работа сервиса сводится к обработке запросов, отправляемых клиентским приложением.
- **Ajax API Google** — предоставляет пользователям интерфейс для доступа к сервисам Google через JavaScript.

- **Ajax API для Google Переводчик** — позволяет осуществлять перевод текста с одного языка на другой с помощью JavaScript
- **AJAX API поиска Google** — позволяет использовать результаты поисковой системы Google в своих проектах.
- **API Карт Google** — позволяет интегрировать интерактивные карты Google с данными на своем сайте.
- **API Карт Google для Flash** — этот API позволяет разработчикам Flex вставлять Карты Google в приложения Flash. Подобно версии для JavaScript, этот API ActionScript предоставляет ряд программ для управления содержанием карт и добавления его через различные службы, позволяя разработчикам создавать надежные, интерактивные приложения карт прямо на своих веб-сайтах.
- **API O3D (лаборатория Google)** — это Web API с открытым исходным кодом для создания мощных, интерактивных приложений трехмерной графики в браузере.
- **API данных веб-альбомов Picasa** — позволяет включить веб-альбомы Picasa в свои приложения или веб-сайт.
- **API Google Chart** — API диаграмм Google, позволяет динамически встраивать статические диаграммы (png-рисунок) в веб-страницу.
- **API визуализаций Google** — API диаграмм Google, позволяет динамически встраивать интерактивные диаграммы в веб-страницу.
- **API Google Планета Земля** — подключаемый модуль Google Earth и JavaScript API позволяют добавить на веб-страницу службу Google Earth. Таким образом, на вашем сайте появится настоящая трехмерная модель целой планеты.
- **API YouTube** — позволяет интегрировать видео YouTube в свой веб-сайт.

Каждый сервис и API к нему заслуживают целой главы, а то и книги. Рассмотрим в качестве примера только некоторые из них.

7.2. Google Ajax API

Итак, Google Ajax API предоставляет пользователям интерфейс для доступа к Google-сервисам через JavaScript. Отличительной особенностью такого подхода является то, что для работы со всеми этими удобствами нет необходимости писать обработчики на PHP и других серверных языках (скрипты которых обрабатываются на стороне сервера). Все, что нужно, — это просто использовать необходимые методы уже готовых JavaScript-классов.

Все Google-сервисы сосредоточены в нескольких модулях (модуль поиска, модуль карт и т. д.). Кроме этого, компания Google учла широкое распространение таких js-фрейморков как jQuery, dojo, prototype, mootools и т. д., и добавила возможность их использования как отдельных модулей. Если вы используете библиотеку jQuery у себя на сайте, то все, что вам нужно, — это загрузить ее с Google методом

```
google.load('jquery', '1.4.1');
```

И дальше уже работать с ней как с ее обычной версией. Но обо всем по порядку. Для начала работы с Google Ajax API вам необходимо сделать две вещи:

- ❒ получить специальный ключ;
- ❒ подключить библиотеку загрузчика на странице вашего сайта.

Для получения ключа к API Карт требуется аккаунт Google. Получить собственный аккаунт можно по адресу **https://www.google.com/accounts/ManageAccount**. На данной странице следует выбрать ссылку **Создайте аккаунт прямо сейчас** (рис. 7.1). И пройти процедуру регистрации. Для получения ключа перейдите по адресу **http://code.google.com/intl/ru-RU/apis/loader/signup.html**. На странице (рис. 7.2) необходимо ввести адрес своего ресурса, с которого будет осуществлять-

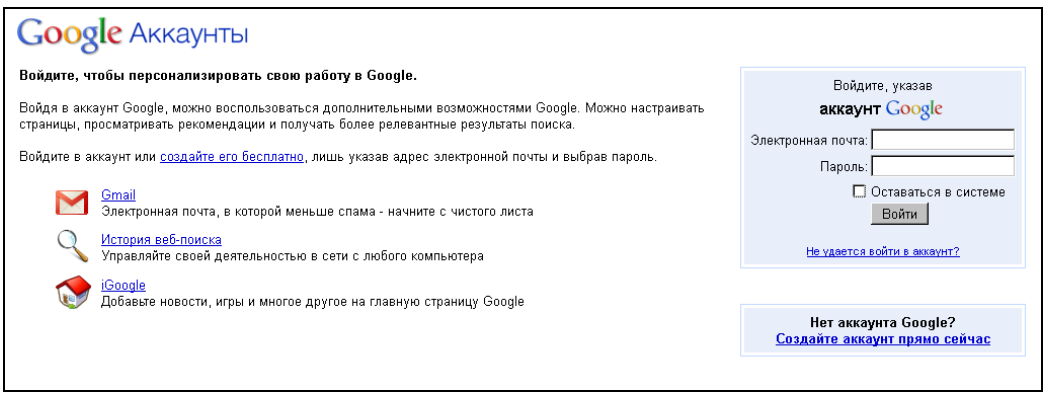


Рис. 7.1. Страница, приглашающая создать аккаунт

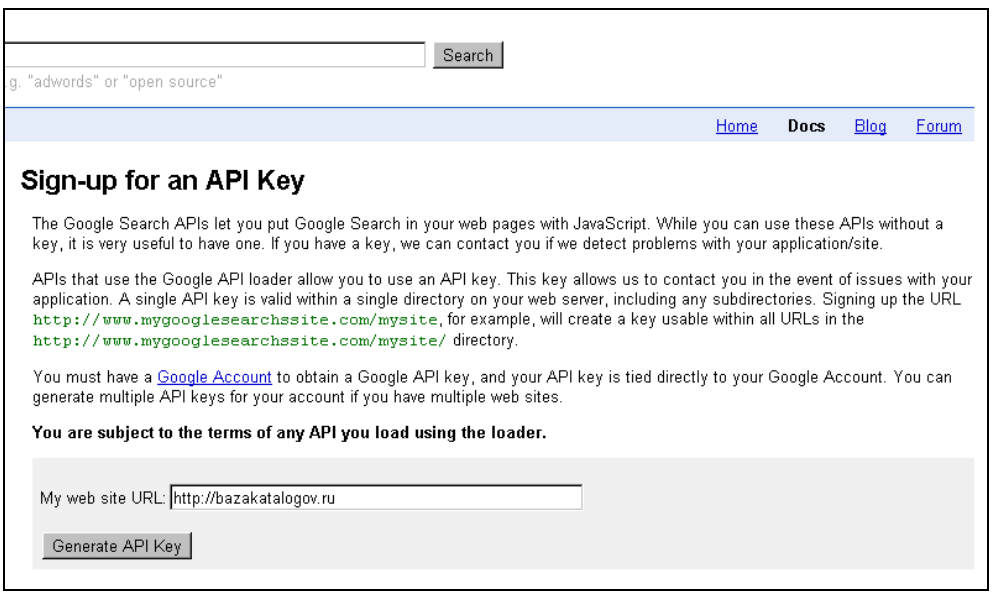


Рис. 7.2. Форма получения ключа

ся обращение к загрузчику. Лучше всего в качестве адреса сайта указывать только его домен (например, **mysayt.ru**). Тогда ключ, зарегистрированный на отдельный домен, будет действителен для всех URL внутри этого домена и для специальных поддоменов (например, для **www**). Ключ, зарегистрированный на домен http://mysayt.ru/, будет действителен для:

- **http://mysayt.ru/**;
- **http://www.mysayt.ru/**;
- **http://www.mysayt.ru/page/**;
- **http://host1.mysayt.ru/**;
- **http://host2.mysayt.ru/page/**.

Для работы на локальном компьютере с именем localhost необходимо зарегистрировать ключ для сайта с адресом **http://localhost**.

Для получения ключа необходимо нажать кнопку **Generate API Key**. В результате будет сформирован ключ (рис. 7.3).

Рис. 7.3. Ключ получен

После получения персонального API-ключа на странице нужно разместить следующий код:

```
<script type="text/JavaScript"
        src=http://www.google.com/jsapi?key=YOUR_KEY></script>
```

где *YOUR_KEY* — это API-ключ. После выполнения этого кода станет доступным объект Google, с помощью которого можно подключать любые доступные модули, например модуль поиска или модуль карт:

```
<script type="text/JavaScript">
  google.load("maps","2"); google.load("search,"1");
</script>
```

Для подключения модулей используется метод `load` с двумя параметрами. Первый — имя модуля, а второй — версия модуля. Общий синтаксис этого метода выглядит так:

```
google.load(moduleName, moduleVersion, optionalSettings);
```

Здесь:

- *moduleName* — имя модуля;
- *moduleVersion* — версия модуля;
- *optionalSettings* — возможные опции для модулей:
  - `callback` — функция, которую нужно запустить сразу после загрузки модуля;
  - `language` — язык для локализации API и интерфейса элементов управления. Формат записи языка — ISO639 (`en_EN` или `ru_RU` и т. д.);
  - `nocss` — `true`/`false`, указывает, загружать ли css-файл со стандартным оформлением элементов управления. Можно указать `false`, если хотите использовать собственное оформление;
  - `packages` — массив строк, указывающий, какие связанные с модулем пакеты так же необходимо загрузить, например можно загрузить пакеты "piechart" и "table" вместе с модулем Visualization;
  - `base_domain` — имя домена, с которого загружать библиотеку. Например, можно загрузить библиотеку с "ditu.google.cn" с модулей maps для использования китайской версии Maps API;
  - `other_params` — другие параметры, специфичные для каждого модуля.

Например, так можно загрузить китайскую версию Google Maps 2:

```
google.load("maps", "2", {"language" : "ja_JP"});
```

Имена библиотек также являются пространствами имен. Например, при загрузке модуля maps все методы и поля этого модуля будут доступны, как методы и поля google.maps. Вторым параметром `google.load` передается версия модуля. Версии всех модулей состоят их двух частей: полной версии и номера релиза в виде ВЕРСИЯ.РЕЛИЗ. Каждый раз, когда появляется новая версия модуля, номер релиза увеличивается на один. Если версия модуля search была 2.23, и его обновили, то его версия станет 2.24. При этом версия 2.23 будет называться стабильной (stable), а версия 2.24 — тестовой (test).

Теперь несколько слов о синтаксисе указаний версий. Например, если вы указали версию 2 во втором параметре для модуля search, то будет загружена последняя stable-версия этого модуля. Если нужно загрузить test-версию, то ее нужно будет указать полностью (2.24, например) или указать 2.*x*, что будет эквивалентно. Для загрузки нужных модулей нужно вызывать `google.load` с соответствующими параметрами, но это можно делать вместе с подключением загрузчика. Нужные модули можно подгружать тогда, когда в них есть необходимость, например в ответ на какие-то действия пользователя (листинг 7.1).

Листинг 7.1

```
function mapsLoaded()
{ var map = new google.maps.Map2(document.getElementById("map"));
  map.setCenter(new google.maps.LatLng(37.4419, -122.1419), 13);
}
function loadMaps()
  google.load("maps", "2", {"callback" : mapsLoaded});
```

Сам загрузчик тоже может быть загружен динамически (листинг 7.2).

Листинг 7.2

```
function mapsLoaded()
{ var map = new google.maps.Map2(document.getElementById("map"));
  map.setCenter(new google.maps.LatLng(37.4419, -122.1419), 13);
}
function loadMaps()
{ google.load("maps", "2", {"callback" : mapsLoaded});
}
function initLoader()
{ var script = document.createElement("script");
  script.src = "http://www.google.com/jsapi?key= YOUR_KEY&callback=loadMaps";
  script.type = "text/JavaScript";
  document.getElementsByTagName("head")[0].appendChild(script);
}
```

Когда ваша страница обращается к загрузчику, то он пытается определить ваше местоположение по IP, от которого пришел запрос. Данные о местоположении сохраняются в свойстве `google.loader.ClientLocation`. В случае, если местоположение определить не удалось, это свойство будет содержать `null`. Само свойство `ClientLocation` содержит следующие поля:

- `ClientLocation.latitude` — широта, ассоциированная с IP клиента;
- `ClientLocation.longitude` — долгота, ассоциированная с IP клиента;
- `ClientLocation.address.city` — город;
- `ClientLocation.address.country` — название государства;
- `ClientLocation.address.country_code` — код государства в формате ISO 3166-1;
- `ClientLocation.address.region` — регион для заданного IP (специфичный для каждого государства).

7.3. Ajax API для Google Переводчика

В Google очень давно существует сервис перевода текстов с одного языка на другой. Кроме этого сервиса Google создал возможность доступа к сервису переводов с приложений на JavaScript и открыл соответствующее API — Ajax API Google

Translate. API включает в себя функции автоматического определения языка исходного текста и собственно сам перевод. Сам перевод выполняется на сервере, клиентский код отсылает текст для распознавания языка и перевода и получает ответ, выводя его пользователю. Фоновой функции определения языка для перевода нет, поэтому всегда нужно сначала определить язык своего текста (если вы его не задаете вручную), а потом уже посылать на перевод. Об ограничениях на количество запросов ничего не известно, а вот ограничение на длину текста в одном запросе есть — 500 символов.

Создадим пример использования данного API — чтение RSS-ленты газеты "Times", вывод новостей на языке оригинала (английский). При выделении текста будет появляться окно с переводом. В Интернете данный пример можно посмотреть по адресу **http://examples-api.bazakatalogov.ru/google/js_language/index.php** (рис. 7.4). На компакт-диске пример находится в папке glava_07\google_js_language. Получаем XML-файл RSS-ленты, выводим на страницу в виде списка новостей на языке оригинала. Перевод необходимого фрагмента будем производить с помощью JavaScript-кода. Выводить текст перевода будем в окно, появляющееся в месте выделенного текста — элемент `div` с `id=translation`:

```
#translation { position:absolute; background:yellow;
               max-width:300px; height:avto }
```

При загрузке страницы подгружаем модуль Google Translate:

```
google.load("language", "1");
```

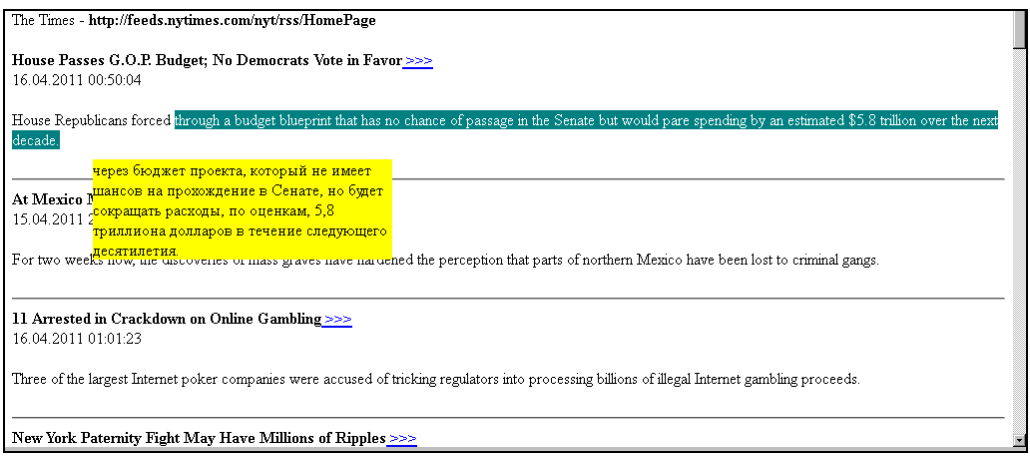


Рис. 7.4. RSS-лента с возможностью перевода выделенного текста

Для документа определим обработчик `onclick`, который будет возвращать фрагмент выделенного текста и делать видимым окно перевода в нужных координатах либо, при отсутствии выделения, скрывать окно.

Собственно, сам перевод выполняется вызовом API-функции `google.language.translate()`, которой передается строка с текстом, код языка, на который переводим, и код языка оригинала (который мы получаем вызовом API-функции

`google.language.detect()`). Библиотека берет на себя все низкоуровневые функции, и мы получаем только объект `result`, который и содержит возвращенные данные. Проверяем свойство `result.translation`: если он не равно `false`, значит, перевод успешный и в нем содержится переведенная строка текста. Переведенный текст заносим в окно перевода (`div id=translation`).

Содержимое файла index.php приведено в листинге 7.3. Обратите внимание на js-функцию `GetSelectedText()`, из которой понятно различие механизма определения выделенного текста в теле документа и в полях `input` и `textarea`. Поля `input` и `textarea` приведены для примера. Скопируйте в них текст и с помощью выделения части текста получите перевод (рис. 7.5).

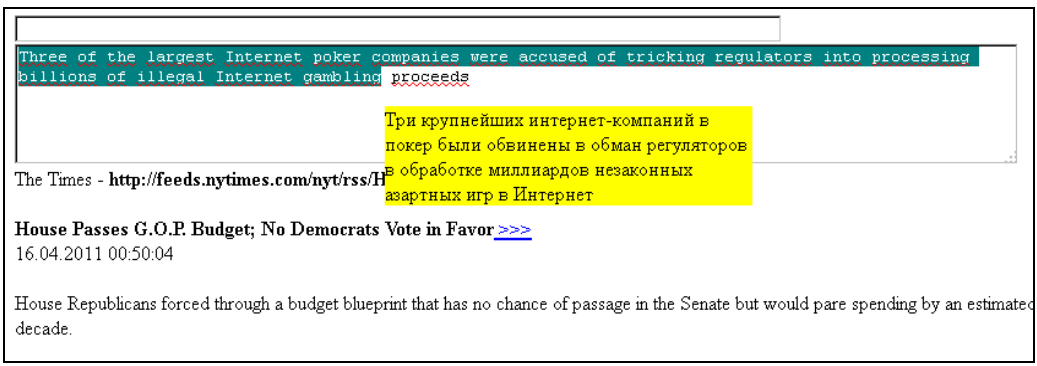


Рис. 7.5. Перевод выделенного текста в поле `textarea`

Листинг 7.3

```
<html>
<head>
<script type="text/JavaScript" src="http://www.google.com/jsapi"></script>
<style type="text/css">
#translation { position:absolute; background:yellow; max-width:300px;
               height:avto }
</style>
<script type="text/JavaScript">
  google.load("language", "1");
  function translate(txt)
  { google.language.detect(txt, function(result)
    { if (!result.error && result.language)
      { google.language.translate(txt, result.language, "ru",
        function(result)
        { var translated = document.getElementById("translation");
          if (result.translation)
          { translated.innerHTML = result.translation;
            translated.style.display="block"; }
        });
```

```
      }
    });
  }
  function GetSelectedText ()
  { var selText = "";
    if (window.getSelection)
    { // Firefox, Opera, Google Chrome, Safari
      if (document.activeElement &&
          (document.activeElement.tagName.toLowerCase() == "textarea" ||
           document.activeElement.tagName.toLowerCase() == "input"))
      { var text = document.activeElement.value;
        selText = text.substring(document.activeElement.selectionStart,
                                 document.activeElement.selectionEnd); }
      else
      { var selRange=window.getSelection();selText = selRange.toString();}
    }
    else
    { if (document.selection.createRange)
      { // Internet Explorer
        var range=document.selection.createRange();selText = range.text; }
    }
    return selText;
  }
  document.onclick = function (e)
  { var translated = document.getElementById("translation");
    var dy=(!e)?document.documentElement.scrollTop:window.pageYOffset;
    var yyy=(!e)?(event.clientY+dy+10):(e.pageY+20);
    var xxx=(!e)?(event.clientX-1):(e.pageX-1);
    translated.style.left=""+xxx+"px"; translated.style.top=yyy+"px";
    selText = GetSelectedText ();      translated.style.display="none";
    if(selText.length>0) translate(selText);
  }
</script>
</head>
<body>
  <div id="translation"></div>
  <input type=text size=100><br>
  <textarea cols=100 rows=5></textarea><br>
  <?php
    $records=array();
    $url="http://feeds.nytimes.com/nyt/rss/HomePage";
    echo "The Times — <b>".$url."</b><br><br>";
    $xml = simplexml_load_file($url);
    $records=$xml->xpath('channel/item');
    if (empty($records))
    { echo "Пустой результат "; }
```

```
    else
    { foreach ($records as $record)
      { echo "<b>".$record->title."<a href='".$record->link."'> >>></a>
        </b><br>".date('d.m.Y H:i:s',strtotime($record->pubDate))."<br><br>".
             $record->description."<br>"."<hr>";}
    }
  ?>
</body>
</html>
```

7.4. Ajax API поиска Google

Поиск по сайту — функция, безусловно, очень важная, особенно для больших ресурсов, на которых найти нужную информацию бывает не так просто. Крупные поисковые сервисы (в частности Google) предоставляют API на базе JavaScript и Ajax, которые можно легко подключить к любому сайту. API Search Google позволяет разработчикам как внедрять в свои страницы поиск, в том числе по видео, новостям и картам, так и строить свои приложения.

Рассмотрим, как внедрить поиск на наш сайт. Сначала необходимо получить API-ключ. Процесс получения ключа описан в *разд. 7.2*. Далее берем код из документации для Ajax API Search Google для быстрого старта (листинг 7.4), помещаем на нашем сайте и запускаем скрипт (рис. 7.6). Не забываем вписать вместо *API_KEY* свой ключ. Получаем стандартную форму поиска Google, за которой кроется вся мощь поисковой системы Google.

Листинг 7.4

```
<html>
<head>
  <script src="https://www.google.com/jsapi? key=API_KEY"
          type="text/JavaScript"></script>
  <script type="text/JavaScript">
    google.load("search", "1");
    // Call this function when the page has been loaded
    function initialize() {
      var searchControl = new google.search.SearchControl();
      searchControl.addSearcher(new google.search.WebSearch());
      searchControl.addSearcher(new google.search.NewsSearch());
      searchControl.draw(document.getElementById("searchcontrol"));
    }
    google.setOnLoadCallback(initialize);
  </script>
</head>
<body>
  <div id="searchcontrol"></div>
</body>
</html>
```

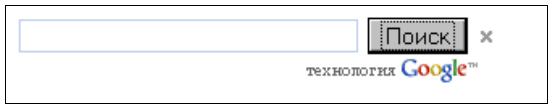


Рис. 7.6. Внедрение формы поиска на сайт

Наберите в поиске слово, нажмите кнопку **Поиск** (Search), и пусть Google работает. Вы быстро увидите несколько результатов, как показано на рис. 7.7. Все неплохо, но есть существенные недостатки — данный способ формирует избыточную HTML-разметку, которая к тому же "обвешана" дополнительными CSS-правилами, которые подойдут не каждому сайту.

Рис. 7.7. Результаты, выдаваемые внедренной формой поиска

Поэтому создадим форму поиска с использованием этого же API, но на более низком уровне, что позволит сформировать любую HTML-разметку и украшать ее собственными стилями. Пример созданной формы можно посмотреть в сети по адресу **http://examples-api.bazakatalogov.ru/google/search/**. Сначала произведем разметку страницы поиска. Создадим три блока:

- ❒ блок формы поиска;
- ❒ блок вывода результатов;
- ❒ блок навигации по страницам поиска.

Не забываем вывести и логотип Google (см. пользовательское соглашение). Код разметки страницы представлен в листинге 7.5.

Листинг 7.5

```
<body>
  <h3><b>Пример из книги (глава 7 — Ajax API Search Google) </b></h3>
  <div id=options_search>
    <form id="form_search" action='JavaScript:void()'
          onsubmit='Go_Search(this.text1.value);'>
      <input type="text" name=text1 size=50 value="" />
      <input type="submit" id="button_search" value="Искать" />
    </form>
    Используется лучший поисковик -
    <a href="http://google.ru"><img  src="http://www.google.com/uds/
            css/small-logo.png" alt="Google" /></a>
  </div>
  <h4>Результаты поиска:</h4>
  <div id="result_search">
  </div>
  <div id="pages_search">
  </div>
</body>
```

Подключение любых API от Google производится с помощью специального загрузчика API. Поэтому первым делом в секции `<head>` страницы подключаем скрипт загрузчика:

```
<script src="https://www.google.com/jsapi?key=API_KEY" type="text/JavaScript" >
</script>
```

где вместо *API_KEY* записываем свой ключ.

Далее загружаем API Search Google. Вызов загрузчика выполняется с дополнительными параметрами, это позволяет избежать загрузки ненужных фрагментов JavaScript и CSS, соответственно, немного увеличивается производительность:

```
var webSearch;
google.load('search', '1', {'nooldnames':true, 'language':'ru-RU',
            'nocss': true});
```

Любые действия с API поиска можно производить только после полной загрузки API и страницы, поэтому нужно зарегистрировать обработчик соответствующего события:

```
google.setOnLoadCallback(PageLoad);
```

Содержимое обработчика полной загрузки страницы и API-функции `PageLoad()` представлено в листинге 7.6.

Листинг 7.6

```
function PageLoad()
{ // создать объект веб-поиска
  webSearch = new google.search.WebSearch();
```

```
  // задать размер результатов
  //webSearch.setResultSetSize(google.search.Search.LARGE_RESULTSET);
  webSearch.setResultSetSize(google.search.Search.SMALL_RESULTSET);
  // запретить автоматическую генерацию результатов поиска
  webSearch.setNoHtmlGeneration();
  // обработчик события завершения поиска
  webSearch.setSearchCompleteCallback(this, WebSearchComplete);
}
```

При нажатии кнопки формы **Искать** по событию `onsubmit` выполняется функция `Go_Search()`, которая получает в качестве аргумента содержимое текстового поля формы поиска и запускает поиск (листинг 7.7).

Листинг 7.7

```
// запуск поиска
function Go_Search(txt)
{ if(txt.length<1) return;
  document.getElementById("button_search").disabled=true;
  document.getElementById("button_search").value="Идет поиск";
  webSearch.execute(txt);
  return;
}
```

Обработчик события завершения поиска — функция `WebSearchComplete()`. После завершения поиска результаты доступны в свойствах объекта поиска `results` и `cursor` в виде JSON-объектов. Соответственно, вывод результатов на страницу — это уже дело техники (листинг 7.8).

Листинг 7.8

```
function WebSearchComplete()
{ // активировать кнопку формы
  document.getElementById("button_search").value="Искать";
  document.getElementById("button_search").disabled=false;
  if(webSearch.results.length<1)  // нет результата
  { document.getElementById("result_search").innerHTML="По вашему запросу
          ничего не найдено";
    document.getElementById("pages_search").innerHTML="";
  }
  // сформировать контент из результата
  var content="";
  for(var i=0;i<webSearch.results.length;i++)
  { // заголовок
    content+="<b>"+webSearch.results[i].titleNoFormatting+"</b><br>";
    // содержание
    content+=webSearch.results[i].content+"<br>";
```

```
    // ссылка
    content+="<a href='"+webSearch.results[i].url+"' target=blank>
     Перейти >>> <a><br>";
    content+="<hr>";
  }
  document.getElementById("result_search").innerHTML=content;
  // указатель страниц
  pages="";
  if(webSearch.cursor.pages.length<2)  // 1 страница
  { document.getElementById("pages_search").innerHTML="";
    return;
  }
  if(webSearch.cursor.currentPageIndex!=0)
  { pages+=" <a href='JavaScript:void()' ";
    pages+="onclick='Go_Search_Page(";
    pages+=webSearch.cursor.currentPageIndex-1;
    pages+=")'><<<</a> ";
  }
  for(var i=0;i<webSearch.cursor.pages.length;i++)
  { if(i==webSearch.cursor.currentPageIndex)
      pages+=" "+(i+1)+" ";
    else
    { pages+=" <a href='JavaScript:void()' ";
      pages+="onclick='Go_Search_Page("+i+")'>"+(i+1)+"</a> ";
    }
  }
  if(webSearch.cursor.currentPageIndex<webSearch.cursor.pages.length-1)
  { pages+=" <a href='JavaScript:void()' ";
    pages+="onclick='Go_Search_Page(";
    pages+=webSearch.cursor.currentPageIndex+1;
    pages+=")'>>>></a> ";
  }
  // вывод указателя страниц
  document.getElementById("pages_search").innerHTML=pages;
}
```

При выводе указателя страниц формируем запуск функции `Go_Search_Page()`, которая запускает вывод результатов поиска для выбранной страницы (листинг 7.9).

Листинг 7.9

```
// переход по страницам
function Go_Search_Page(page)
{ document.getElementById("button_search").disabled=true;
  document.getElementById("button_search").value="Идет поиск";
  webSearch.gotoPage(page);
  return;
}
```

Полный код примера представлен на прилагаемом компакт-диске в папке glava_07\google_search. Вид страницы с результатами поиска представлен на рис. 7.8.

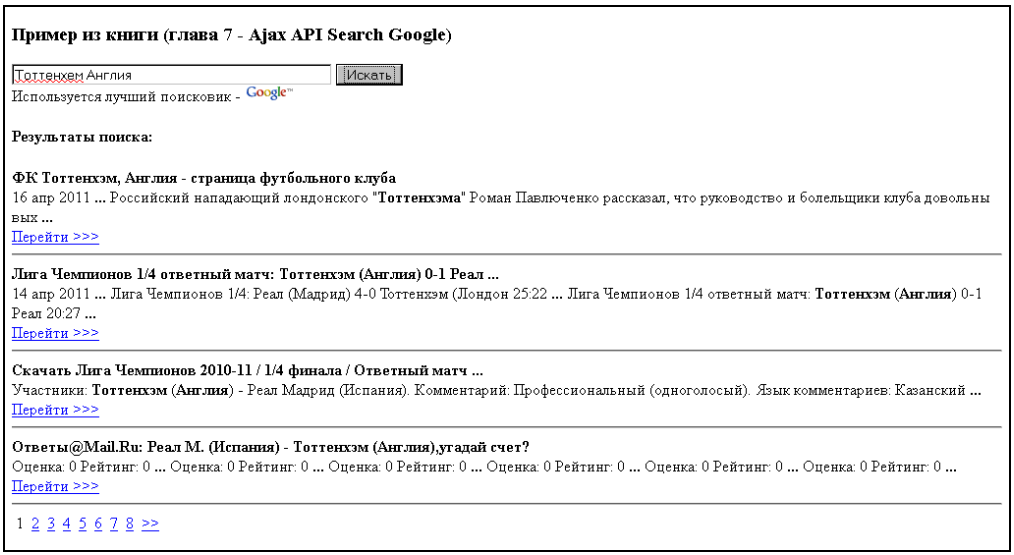


Рис. 7.8. Страница поиска для примера использования Ajax API Search Google

7.5. API Google Chart

Лучшей рекламой для этого API служит фраза "рисунок заменяет тысячу слов". Что более наглядно — просмотр длинного списка чисел или график, объединяющий все цифры? Диаграммы и графики предоставляют краткий обзор большого количества информации. При помощи диаграммы человек может быстро заметить любой тренд, сравнить различные результаты либо выявить закономерность.

Инструменты API Google Chart обеспечивают легкий способ добавления диаграмм на любую веб-страницу. Графики могут быть статическими или интерактивными *(см. разд. 7.6)*. Программный интерфейс построения статических диаграмм достаточно прост: все параметры отрисовки диаграммы и данные для визуализации передаются на сервер Google в виде GET-запроса (проще говоря, специального URL), а сервер в ответ отдает png-картинку с нарисованным графиком. По скорости процесс сравним с загрузкой статических файлов. Кроме того, правила его использования не накладывают никаких жестких ограничений на количество обращений с одного хоста. API Google Chart бесплатный и не требует наличия никакой учетной записи и прохождения процесса регистрации. Все это позволяет очень легко интегрировать диаграммы в свои системы, да еще и бесплатно снять с собственного сервера дополнительную нагрузку по генерации графики.

Основной ссылкой на API Google Chart является **http://chart.apis.google.com/chart?**. Параметры, которые определяют отображение диаграммы, следуют после

символа **?**. Существует множество параметров, которые вы можете указать посредством строки запроса. Вот некоторые из них:

- **cht** — тип диаграммы, существует очень большое количество, например:
  - bhs, bvs, bhg, bvg, bvo — столбчатый (вертикальный, горизонтальный) график;
  - lc, ls, lxy — линейный график;
  - p, p3, pc — секторная диаграмма;
  - v — диаграмма Венна;
  - s — точечная диаграмма.

  Полный список типов диаграмм можно посмотреть по адресу

  http://code.google.com/intl/en/apis/chart/docs/gallery/chart_gall.html;
- **chs** — размер диаграммы в виде width×height, где width и height — количество пикселов в отношении ширины и высоты диаграммы. Максимальная высота и ширина диаграммы может быть 1000 пикселов, и в результате площадь диаграммы не должна превышать 300 000 пикселов;
- **chd** — данные для диаграммы. При использовании этого параметра необходимо указать формат данных. API Google Chart позволяет вам использовать различные кодировки данных. Простейшим способом будет использование текстовой кодировки, которая обозначается буквой t. Далее идет двоеточие (:), а за ним — список значений точек графика, разделенных запятой. Стандартная текстовая кодировка требует наличия числовых значений, отображаемых на графике, в виде чисел с плавающей точкой между нулем (0.0) и сотней (100.0). Для того чтобы правильно масштабировать данные, необходимо преобразовать каждое значение в процентное отношение к значению самому большому значению. То есть самое большое значение может быть 100.0, а остальные точки будут выражены в процентном отношении к этому значению: 50.0 — это половина самого большого значения, 25.0 для 25% от самого большого и т. д. Чтобы обработать диаграмму со значениями 10, 20 и 8, посылаем chd=t:50,100,40. Обратите внимание на t:, которое указывает на то, что форматирование данных использует текстовую кодировку. В качестве альтернативы можно применить метод текстовой кодировки с масштабированием данных, который позволяет использовать значения с плавающей точкой, но как положительные, так и отрицательные;
- **chds** — масштабирование данных — от и до;
- **chtt** — заголовок диаграммы;
- **chdl** — текст для каждой серии, для отображения в сноски;
- **chdlp** — позиция для данных (left, right, top, bottom);
- **chco** — цвет для каждой серии;
- **chf** — цвет фона, фона графика, прозрачность.

Полный список параметров можно посмотреть на странице **http://code.google.com/intl/en/apis/chart/docs/chart_params.html**. Единственными обязательными пара-

метрами являются тип диаграммы (**cht**), размер диаграммы (**chs**) и данные диаграммы (**chd**).

Чтобы лучше разобраться в диаграммах, перейдем на страницу составления запросов и построения диаграмм Image Chart Editor, расположенную по адресу **http://imagecharteditor.appspot.com/**, где можно потренироваться в выборе параметров для составления диаграмм (рис. 7.9).

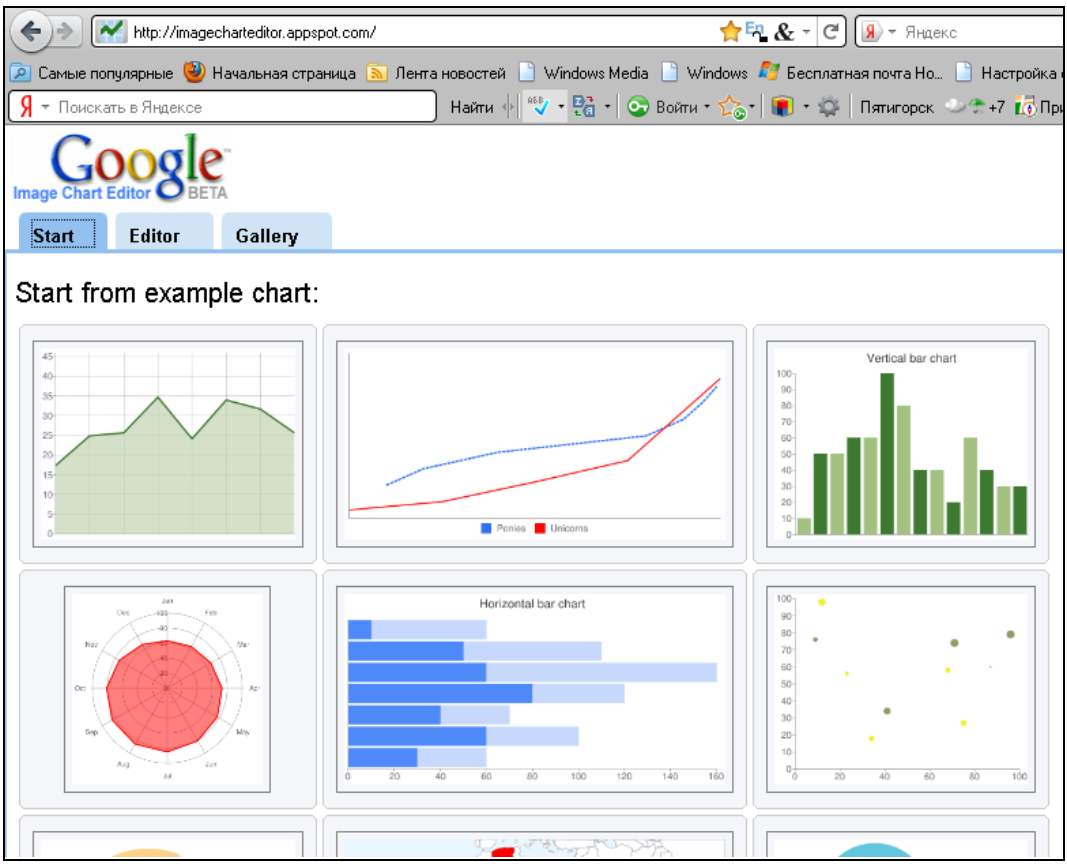


Рис. 7.9. Страница построения диаграмм Image Chart Editor

Переходим на вкладку **Editor** и, изменяя параметры, получаем диаграммы (рис. 7.10).

Теперь в качестве усвоения материала создадим небольшой пример. Будем строить диаграммы для сравнения количества районов, городов, прочих населенных пунктов в различных регионах Российской Федерации. В сети данный пример расположен по адресу **http://examples-api.bazakatalogov.ru/google/static_chart/**. Статистические данные будем брать из официальных баз КЛАДР. На компакт-диске в папке glava_07\google_static_chart находится дамп базы данных. Таблица с данными КЛАДР "перегонялась" в MySQL для другого примера, поэтому не будем обращать внимание на избыточное количество полей.

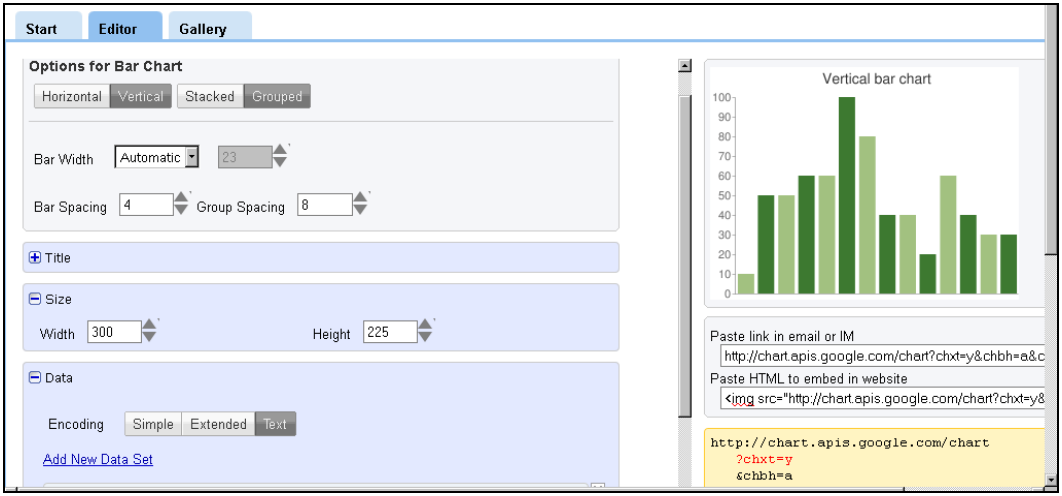


Рис. 7.10. Пример построения диаграммы в Image Chart Editor

Структура таблицы `kladr` следующая:

- `id` — уникальный идентификатор, дополненный одним кодом КЛАДР для каждой записи;
- `id_region` — взятые из кода данные полей региона для данной записи;
- `id_rayon` — взятые из кода данные полей района для данной записи;
- `id_town` — взятые из кода данные полей города для данной записи;
- `id_punkt` — взятые из кода данные полей негородского пункта для данной записи;
- `name` — наименование объекта КЛАДР;
- `socr` — дополнение наименования (обл., г., с., р-н и пр.).

Дамп для создания структуры представлен в листинге 7.10.

Листинг 7.10

```
CREATE TABLE `kladr` (
`id` bigint(12) NOT NULL AUTO_INCREMENT ,
`id_region` int(2) NOT NULL default '0',
`id_rayon`  int(3) NOT NULL default '0',
`id_town`   int(3) NOT NULL default '0',
`id_punkt`  int(3) NOT NULL default '0',
`name` varchar(100) CHARACTER SET cp1251 default NULL ,
`socr` varchar(10) CHARACTER SET cp1251 default NULL ,
`zindex` int(6) default NULL ,
UNIQUE KEY `id` (`id`)
) ENGINE = MYISAM AUTO_INCREMENT = 1 DEFAULT CHARSET = cp1251 COLLATE =
cp1251_bin
```

На главной странице примера (рис. 7.11) создадим форму выбора параметров для построения будущих диаграмм. Создадим возможность выбора следующих параметров:

- тип диаграммы;
- размер диаграммы;
- заголовок диаграммы;
- наличие блока легенды;
- расположение блока легенды.

Поле `select` выбора списка регионов и параметра сравнения (районы, города, прочие населенные пункты) имеет свойство `multiple` (множественный выбор). Список регионов берется из таблицы `kladr` нашей базы данных. Содержимое файла index.php, где формируется сама форма, представлено в листинге 7.11.

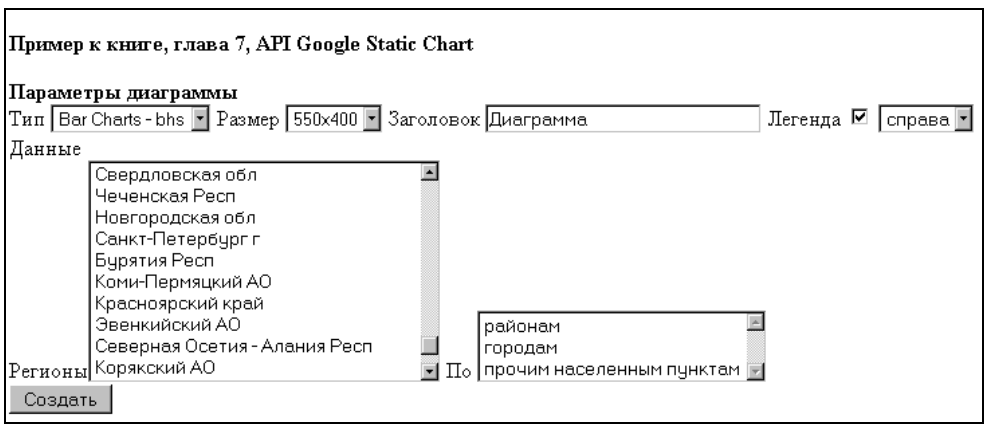


Рис. 7.11. Форма выбора параметров диаграммы

Листинг 7.11

```
<?
// подключение библиотеки xAjax
... ... ...
?>
<html xmlns="http://www.w3.org/1999/xhtml" lang="ru" xml:lang="ru">
<head>
<title>Пример к книге, глава 7, API Google Static Chart</title>
<?php $xajax->printJavaScript(''); ?>
</head>
<body >
  <p><b>Пример к книге, глава 7, API Google Static Chart</b></p>
  <!-- настройки  -->
  <div id=options>
    <form name=form_options id=form_options action='JavaScript:void();'
```

```
     onsubmit='xajax.$("button_options").disabled=true;
     xajax_Create_Static_Chart(xajax.getFormValues("form_options"));'>
    <b>Параметры диаграммы</b><br>
    Тип <select name=cht>
     <option value='bhs'> Bar Charts — bhs
     <option value='bvs'> Bar Charts — bvs
     <option value='bhg'> Bar Charts — bhg
     <option value='bvg'> Bar Charts — bvg
     <option value='bvo'> Bar Charts — bvo
    </select>
    Размер <select name=chs>
     <option value='550x400'>550x400
     <option value='600x425'>600x425
     <option value='650x450'>650x450
    </select>
    Заголовок
    <input type=text name=chtt size=30 maxlength=30 value='Диаграмма'>
    Легенда
    <input name=chdlp_ok type=checkbox checked>
    <select name=chdlp>
     <option value='r' selected>справа<option value='l'>слева
     <option value='t'>вверху<option value='b'>внизу
    </select><br>
    Данные<br>
    Регионы <select name=chd1[] size=10 multiple>
    <?php
    // получение содержимого для select
    $query1="SELECT id_region,name,socr FROM ".$db_table." WHERE
             id_rayon='0' && id_town='0' && id_punkt=0 ";
    $rez1=mysql_query($query1);
    while($row1=mysql_fetch_assoc($rez1))
    { $text1.="<option value='".$row1[id_region]."'>
            ".$row1[name]." ".$row1[socr];
    }
    echo $text1;
    ?>
    </select>
    По <select name=chd2[] size=3 multiple>
     <option value='1'>районам
     <option value='2'>городам
     <option value='3'>прочим населенным пунктам
    </select><br>
    <input type=submit id=button_options value="Создать"
      <?php echo $disabled; ?> >
  </form>
</div>
```

```
  <!-- запрос -->
  <div id=query style='background-color:#eeeeee'></div>
  <!-- диаграмма  -->
  <div id=img></div>
</body>
</html>
```

Данные формы отправляются в xAjax-функции `Create_Static_Chart()`, которая анализирует данные запроса, получает из БД необходимые данные для запроса, формирует URL запроса и отправляет запрос к API Google Static Chart. Ответ сервера — картинка в формате PNG выводится в блок `img` главной страницы. Для правильного отображения данных необходимо предусмотреть установку в запросе параметров масштаба, разметки одной из осей наименованиями регионов, автоматического подбора размеров диаграммы, установки параметров цвета для диаграммы. URL запроса выдается в блок `query`, наличие кракозябр происходит из необходимости отправляемые русскоязычные символы перекодировать в кодировку UTF-8. xAjax-функция `Create_Static_Chart()` расположена в файле create_static_chart.php, содержимое которого представлено в листинге 7.12.

Листинг 7.12

```
<?php
// Получение/построение диаграммы Google Static Chart
function Create_Static_Chart($Id)
{ $objResponse = new xajaxResponse();
  // проверка выбора данных
  ... ... ...
  // подготовка запроса
  $query='http://chart.apis.google.com/chart?';
  // тип диаграммы
  $query.='cht='.$Id[cht].'';
  // размер
  $query.='&chs='.$Id[chs].'';
  // заголовок
  $query.='&chtt='.iconv("windows-1251","utf-8",$Id[chtt]).'';
  // данные, легенда, цвета
  $chd='&chd=t:'; $chdl='&chdl=';
  $colors=array('ff0000','00ff00','0000ff','ff7700','0077ff',
                'ffff00','00ffff','ff00ff','0077ff','77AA55',);
  $chco='&chco='; $max=0;
  $data2=array("","id_rayon>0 && id_town=0 && id_punkt=0",
               "id_town>0 && id_punkt=0","id_punkt>0");
  $data2n=array("","районы","города","прочие н/п");
  // получаем данные из БД
  for($i2=0;$i2<count($Id[chd2]);$i2++)
  { $chxl='&chxl=1:';
```

```
    $chdl.=iconv("windows-1251","utf-8",$data2n[$Id[chd2][$i2]]).'|';
    for($i1=0;$i1<count($Id[chd1]);$i1++)
    { $query1="SELECT count(id) FROM ".$db_table." WHERE
          id_region='".$Id[chd1][$i1]."' && ".$data2[$Id[chd2][$i2]]." ";
      $rez1=mysql_query($query1); $count=mysql_result($rez1,0);
      $chd.=$count.",";   $max=max($max,$count);
      $query2="SELECT name FROM ".$db_table." WHERE
             id_region='".$Id[chd1][$i1]."' && id_rayon=0
             && id_town=0 && id_punkt=0 ";
      $rez2=mysql_query($query2); $name=mysql_result($rez2,0);
      $chxl.='|'.iconv("windows-1251","utf-8",$name);
    }
    $chd=substr($chd,0,strlen($chd)-1); $chd.="|";
    $chco.=$colors[$i2].",";
  }
  $chd=substr($chd,0,strlen($chd)-1);
  $chco=substr($chco,0,strlen($chco)-1);
  $chdl=substr($chdl,0,strlen($chdl)-1);
  $query.=$chd.$chco.$chxl;
  if(isset($Id[chdlp_ok]))           // легенда
    $query.=$chdl."&chdlp=".$Id[chdlp];
  // масштаб
  $query.='&chxr=0,0,'.$max; $query.='&chds=0,'.$max;
  // оси x,y
  if($Id[cht]=='bhs' || $Id[cht]=='bhg')
    $query.='&chxt=x,y';
  else
    $query.='&chxt=y,x';
  // автоматически на весь размер
  $query.='&chbh=a';
  // записать запрос в блок
  $objResponse->assign("query","innerHTML",$query);
  $width=substr($Id[chs],0,3);
  $height=substr($Id[chs],4,3);
  $content='<img src="'.$query.'" width="'.$width.'px"
                height="'.$height.'px">';
  // полученную картинку в блк диаграмм
  $objResponse->assign("img","innerHTML",$content);
  $objResponse->assign("button_options","disabled",false);
  return $objResponse;
}
?>
```

Примеры полученных диаграмм представлены на рис. 7.12—7.14.

Рис. 7.12. Пример диаграммы Bar chart, горизонтальная

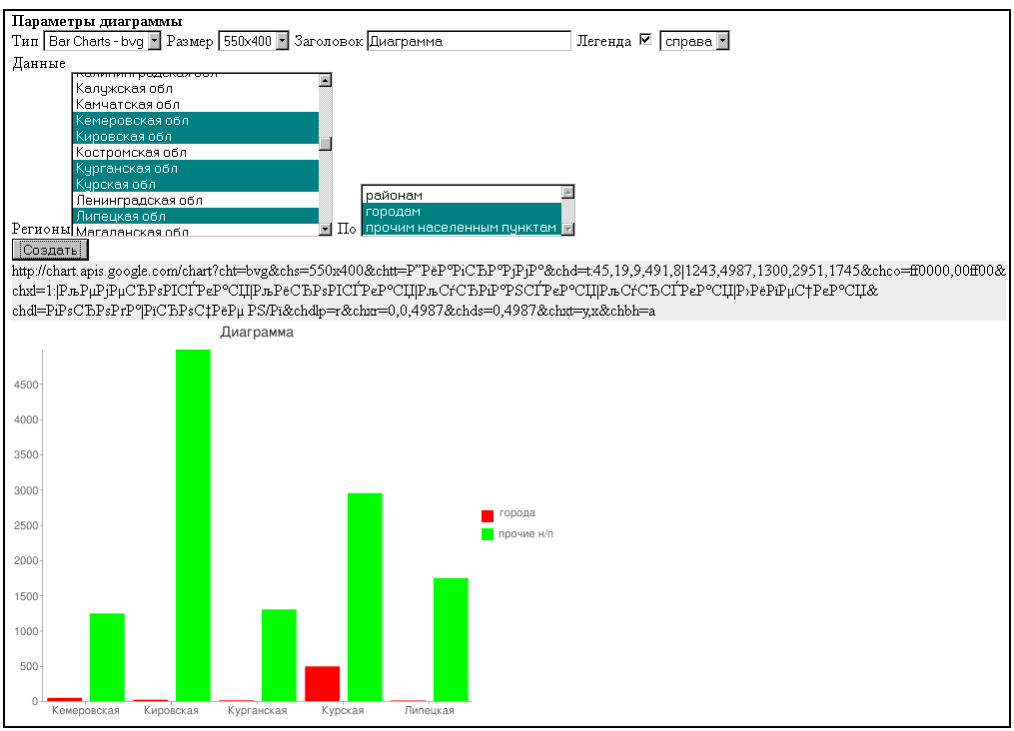


Рис. 7.13. Пример диаграммы Bar chart, вертикальная

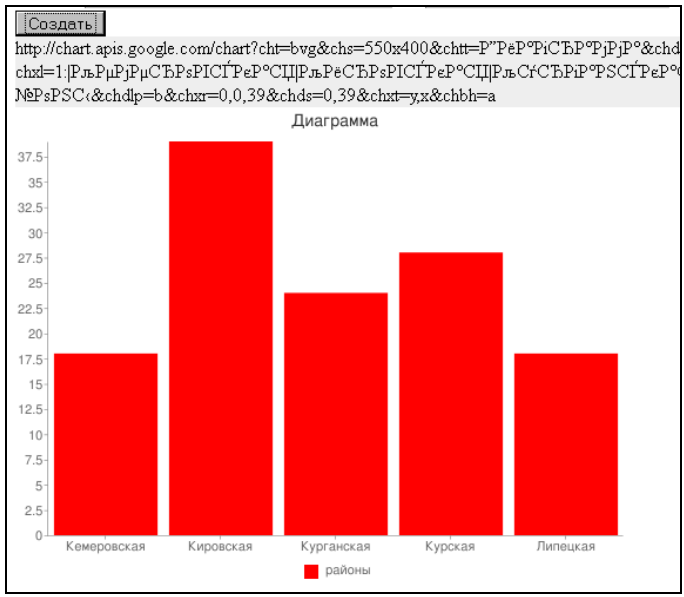


Рис. 7.14. Пример диаграммы Bar chart, вертикальная со сносками внизу

7.6. API визуализаций Google

API Google Visualization предоставляет возможности для создания интерактивных графиков и диаграмм в веб-страницах. Для этого необходимо использовать Google API. Сначала загружаем Ajax API Google, он обеспечивает основные функции для работы:

```
<script type="text/JavaScript" src="http://www.google.com/jsapi">
</script>
```

Затем загружаем API Google Visualization, при этом выбираем необходимые пакеты:

```
google.load('visualization', '1', {'packages':['corechart', 'table']});
```

в данном случае подгружаем пакеты corechart и table.

Далее обычно объявляется callback-функция окончания загрузки библиотек:

```
google.setOnLoadCallback (load_lib);
function load_lib()
{ alert("Библиотеки загружены!!!"); }
```

Теперь необходимы данные: нам нужно, чтобы они были представлены в строках и столбцах. Создаем табличный объект данных:

```
var data=new google.visualization.DataTable();
```

Определяем колонки для таблицы:

```
data.addColumn('string', 'Колонка 1');
data.addColumn('number', 'Колонка 2');
```

Добавляем строки:

```
data.addRows(3);
```

Заполняем строки данными:

```
data.setCell(0, 0, 'Поле1'); data.setCell(0, 1, 120);
data.setCell(1, 0, 'Поле2'); data.setCell(1, 1, 74);
data.setCell(2, 0, 'Поле3'); data.setCell(2, 1, 146);
```

Выбираем тип диаграммы:

```
var chart = new
google.visualization.ColumnChart(document.getElementById('chart'));
```

Записываем опции визуализации:

```
var options = {width: 400, height: 300, is3D: true, title: 'Заголовок'};
```

Рисуем диаграмму:

```
chart.draw(data,options);
```

Можно нарисовать и таблицу с данными:

```
table = new google.visualization.Table(document.getElementById('table'));
table.draw(data);
```

API Google Visualization позволяет создать обработчики событий, т. е. предоставить пользователю возможность реагировать на это событие и выполнять некоторые действия.

```
google.visualization.events.addListener(table,'select',function() {
    var row = table.getSelection()[0].row;
    alert('Значение='+data.getValue(row,0));
});
```

Теперь создадим небольшой проект. Будем строить диаграммы для сравнения количества районов, городов, прочих населенных пунктов в различных регионах Российской Федерации *(см. разд. 7.5)*, но для построения будем использовать API Google Visualization. Создадим два примера: в одном при загрузке API Google Visualization будем выбирать пакеты table и corechart, в другом — table и columnchart. В сети данные примеры расположены по адресам **http://examples-api.bazakatalogov.ru/google/visualizations_chart/index1.php** и **http://examples-api.bazakatalogov.ru/google/visualizations_chart/index2.php**. На прилагаемом компакт-диске файлы проекта расположены в папке glava_07\ google_visualizations_ chart.

Файл index1.php (или index2.php) загружает библиотеку xAjax, Google Ajax API, API Google Visualization с пакетами corechart и table (или columnchart и table), создает разметку страницы и помещает на страницу форму со списком регионов (рис. 7.15). Список регионов берется из базы данных. Содержимое файла index1.php представлено в листинге 7.13.

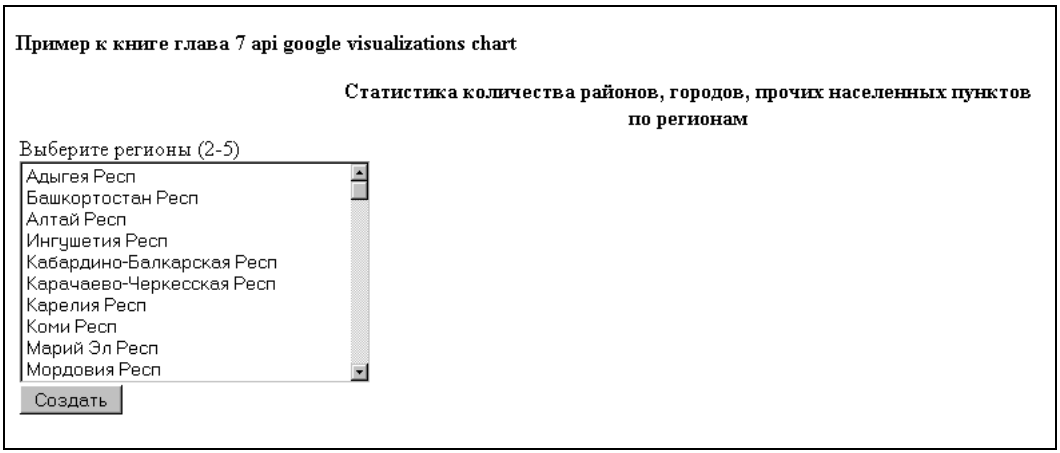


Рис. 7.15. Форма выбора регионов

Листинг 7.13

```
<?
// подключение библиотеки xAjax
... ... ...
?>
<html xmlns="http://www.w3.org/1999/xhtml" lang="ru" xml:lang="ru">
<head>
  <?php $xajax->printJavaScript(''); ?>
  <!--Загрузка Ajax API Google-->
  <script type="text/JavaScript" src="https://www.google.com/jsapi">
  </script>
  <script type="text/JavaScript">
    var data; var formatter; var view; var table;
    var chart1,chart2,chart3;
    // Загрузка API Google Visualization и пакетов
    google.load('visualization', '1', {'packages':[ 'columnchart', 'table']});
    // Callback-функция загрузки библиотек
    google.setOnLoadCallback(load_ok);
    function load_ok()
    { document.getElementById('button_options').disabled=false; }
  </script>
</head>
<body >
  <p><b>Пример к книге, глава 7, API Google Visualizations Chart</b></p>
  <center><b>Статистика количества районов, городов, прочих населенных
  пунктов<br> по регионам</b><br></center>
  <table><tr><td width=30% style='vertical-align:top'>
  <!-- настройки  -->
  <div id=options>
```

```
    <form name=form_options id=form_options action='JavaScript:void();'
       onsubmit='xajax.$("button_options").disabled=true;
       xajax_Create_Visualizations_Chart(xajax.getFormValues(
          "form_options"));'>
      Выберите регионы (2-5)<br>
      <select name=chd1[] size=10 multiple>
      <?php
        // получение содержимого для select-списка
        $query1="SELECT id_region,name,socr FROM ".$db_table." WHERE
                id_rayon='0' && id_town='0' && id_punkt=0 ";
        $rez1=mysql_query($query1);
        $text1="";
        while($row1=mysql_fetch_assoc($rez1))
        { $text1.="<option value='".$row1[id_region]."'>
               ".$row1[name]." ".$row1[socr];
        }
        echo $text1;
      ?>
      </select><br>
      <input type=submit id=button_options value="Создать" disabled
        <?php echo $disabled; ?> >
    </form>
  </div>
  <!-- скрипт -->
  <div id=script style='background-color:#eeeeee'></div>
  </td><td width=70% style='vertical-align:top'>
  <!-- вывод  -->
  <div id=table></div><div id=chart1></div><div id=chart2></div>
  <div id=chart3></div></td></tr></table>
</body>
</html>
```

Данные формы отправляются в xAjax-функцию `Create_Visualizations_Chart()`, которая анализирует данные запроса, получает из базы данных необходимые сведения для формирования таблицы и формирует js-код для построения диаграмм. Текст кода выводится в блок `id=script`, диаграммы — в блоки `id=table`, `chart1`, `chart2`, `chart3`. Вид выводимых диаграмм зависит от подключенных пакетов API Visualizations Chart (рис. 7.16 и 7.17). Обработчик события `sort` таблицы данных перерисовывает диаграмму в блоке `id=chart1` (рис. 7.18). xAjax-функция `Create_Visualizations_Chart()` расположена в файле create_visualizations_chart.php, содержимое которого представлено в листинге 7.14.

Листинг 7.14

```
<?php
// Получение данных для диаграммы, построение диаграммы
// Google Chart Visualizations
function Create_Visualizations_Chart($Id)
```

```
{ $objResponse = new xajaxResponse();
  ... ... ...
  // формирование скрипта
  $script1="data = new google.visualization.DataTable();
  // определение колонок для таблицы
  data.addColumn('string', 'Регион'); data.addColumn('number', 'Районы');
  data.addColumn('number', 'Города');
  data.addColumn('number', 'Прочие нас. пункты');
  data.addRows(".count($Id[chd1]).");";
  // получение данных
  for($i=0;$i<count($Id[chd1]);$i++)
  { $query0="SELECT name,socr FROM ".$db_table." WHERE id_region='
    ".$Id[chd1][$i]."' && id_rayon=0 && id_town=0 && id_punkt=0 ";
    $query1="SELECT count(id) FROM ".$db_table." WHERE id_region='
    ".$Id[chd1][$i]."' && id_rayon>0 && id_town=0 && id_punkt=0 ";
    $query2="SELECT count(id) FROM ".$db_table." WHERE
    id_region='".$Id[chd1][$i]."' && id_town>0 && id_punkt=0 ";
    $query3="SELECT count(id) FROM ".$db_table." WHERE
             id_region='".$Id[chd1][$i]."' && id_punkt>0 ";
    $d0=mysql_result(mysql_query($query0),0,"name")." ";
    $d0.=mysql_result(mysql_query($query0),0,"socr")." ";
    $d1=mysql_result(mysql_query($query1),0);
    $d2=mysql_result(mysql_query($query2),0);
    $d3=mysql_result(mysql_query($query3),0);
    $script1.="data.setCell(".$i.", 0, '".$d0."');";
    $script1.="data.setCell(".$i.", 1, ".$d1.");";
    $script1.="data.setCell(".$i.", 2, ".$d2.");";
    $script1.="data.setCell(".$i.", 3, ".$d3.");";
  }
  $script1.="
    formatter = new google.visualization.NumberFormat({fractionDigits:0});
  formatter.format(data,1);formatter.format(data,2);formatter.format(data,3);
  // выбор данных для визуализации
  view = new google.visualization.DataView(data);
  view.setColumns([0,1,2,3]);
  view1 = new google.visualization.DataView(data);
  view1.setColumns([0,1]);
  view2 = new google.visualization.DataView(data);
  view2.setColumns([0,2]);
  // Выбор типа диаграммы и рисование диаграммы.
  // Таблица
  table = new google.visualization.Table(document.getElementById('table'));
  table.draw(view);
  // column chart
  chart1 = new google.visualization.ColumnChart(
                document.getElementById('chart1'));
  chart1.draw(data, {width: 600, height: 300, is3D: true,  title:
                    'Районы, города, прочие н/п'});
```

```
  // pie-chart
  chart2 = new google.visualization.PieChart(
                  document.getElementById('chart2'));
  chart2.draw(view1, {width: 600, height: 300, is3D: true, title: 'Районы'});
  // bar chart
  chart3 = new google.visualization.BarChart(
                  document.getElementById('chart3'));
  chart3.draw(view2, {width: 600, height: 300, is3D: true, title: 'Города'});
  // обработчик события по сортировке столбца табл. данных
  // – перерисовка column chart
  google.visualization.events.addListener(table, 'sort',
      function(event) {
        data.sort([{column: event.column, desc: !event.ascending}]);
        chart1.draw(data,{width: 600, height: 300, is3D: true,  title:
           'Районы, города, прочие н/п'});
      });";
  // выдача скрипта на выполнение
  $objResponse->script($script1);
  $script1=str_replace(";",";<br>",$script1);
  // выдача кода в блок id=script
  $objResponse->assign("script","innerHTML",$script1);
  $objResponse->assign("button_options","disabled",false);
  return $objResponse;
}
?>
```

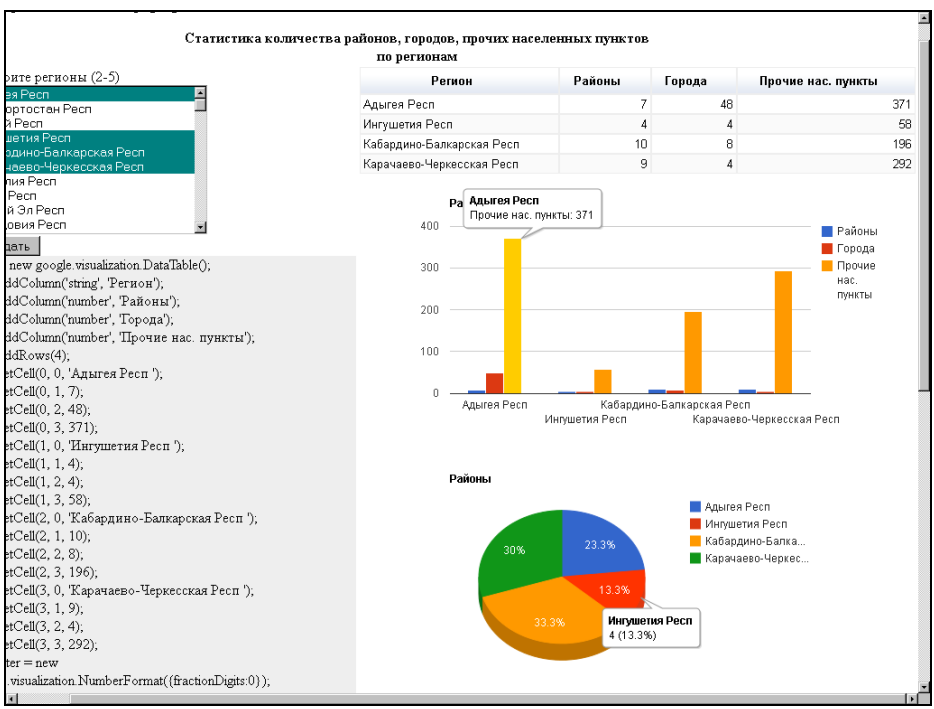


Рис. 7.16. Построение диаграмм с использованием пакетов corechart и table

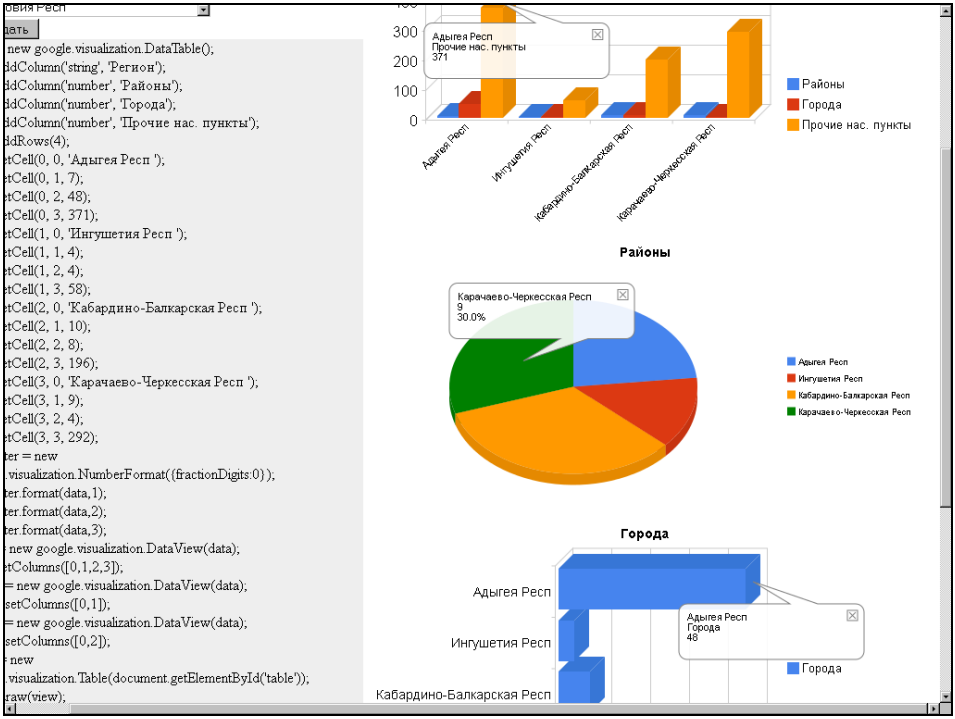


Рис. 7.17. Построение диаграмм с использованием пакетов columnchart и table

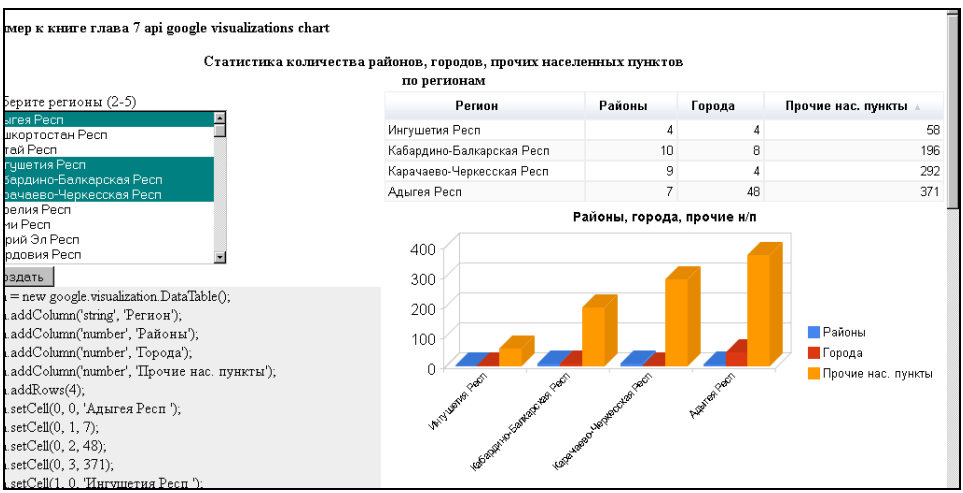


Рис. 7.18. Перерисовка диаграммы по событию `sort`

В Google Code Playground (**http://code.google.com/apis/ajax/playground/?type=visualization#column_chart**) вы можете посмотреть и потренироваться в использовании графиков и диаграмм. Примеры создания графиков можно посмотреть по ссылке

http://code.google.com/intl/en-EN/apis/visualization/documentation/examples.html.

7.7. API Wikipedia

Wikipedia (русскоязычная версия — Википедия, **http://ru.wikipedia.org**) — свободная энциклопедия, которую может редактировать каждый. Wikipedia — это самая большая в мире энциклопедия, подходящая для поиска информации по любой теме. Также вы знаете, что в ней можно найти толкование терминов. Но, возможно, вы не знаете, что Wikipedia содержит в себе массу другой информации, которая позволяет делать намного больше, чем просто помочь найти нужную страницу. С Wikipedia так же можно делать следующее:

- гид по телевизионным шоу и сериалам — наберите в поиске "список серий" или "список эпизодов" с названием шоу или сериала, и вы увидите все серии в каталоге, зачастую с указанием очень интересных мелочей. **http://ru.wikipedia.org/wiki/Список_эпизодов_телесериала_Друзья**;
- краткое содержание произведений — не хотите читать толстый том по литературе? Не беспокойтесь, Wikipedia на вашей стороне. Просто наберите название книги, и вам предоставят краткое содержание, сюжет и прочие результаты анализа, например **http://ru.wikipedia.org/wiki/Дьявол_и_сеньорита_Прим**;[1]
- изучение предметных областей — эта возможность появилась от сестры Wikipedia — Wikiversity (**http://ru.wikiversity.org/wiki/Main_Page**). Если вы студент, желающий получить дополнительные навыки по сложному предмету, или учитель, который ищет домашнее задание либо тезисы для своих студентов, Wikiversity предоставит учебные пособия и рабочие книги по различным предметам от Ajax-программирования до философии. К сожалению, в настоящее время сервис Wikiversity еще недостаточно заполнен, но по многим предметам уже есть полные, хорошо написанные учебные пособия;
- коллекция картинок — нуждаетесь в бесплатной подборке картинок для вашего нового веб-дизайна или для подготовки публикации?[2] Перейдите на Wikimedia Commons (**http://commons.wikimedia.org**/), где вы сможете просмотреть картинки, отсортированные по темам, лицензиям или авторам;
- база музыки — хотите найти новую музыку? В Wikipedia исчерпывающий каталог музыки по жанрам, который позволяет легко найти группы, похожие на те, что вам уже нравятся.

В Википедии есть очень мощный API, предоставляющий прямой высокоуровневый доступ к данным, содержащимся в базах данных MediaWiki. Описание параметров можно посмотреть на странице **http://en.wikipedia.org/w/api.php**.

Создадим скрипт, позволяющий получать информацию о выделенном термине, а также скрипт, выводящий на страницу новости культуры, получаемые с RSS-канала "Яндекс.Новости: Культура". При выделении интересующего нас термина во

[1] Это сервис для ленивых. Если вы уважаете себя — читайте полные издания произведений.

[2] Прежде чем использовать чужие материалы, ознакомьтесь с лицензионным соглашением. Помните, что на все произведения (и даже фото в Интернете) распространяется авторское право.

всплывающем окне будем получать результаты запроса к API Wikipedia со ссылками на статьи. В Интернете данный пример можно посмотреть по адресу **http://examples-api.bazakatalogov.ru/wikipedia/index.php** (рис. 7.19). На компакт-диске пример находится в папке glava_07\wikipedia.

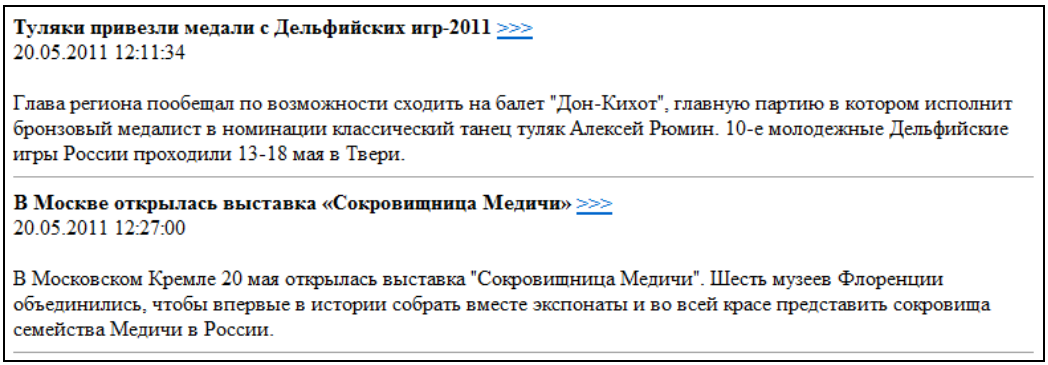


Рис. 7.19. Новости, полученные с "Яндекс.Новости: Культура"

Получаем XML-файл RSS-канала "Яндекс.Новости: Культура" и выводим его на страницу в виде списка новостей. Для документа определим обработчик `onclick`, который будет возвращать фрагмент выделенного текста, вызывать xAjax-функцию `Get_Info_Wiki()` и делать видимым окно (`div id=wiki`) в нужных координатах со значком загрузки либо, при отсутствии выделения, скрывать окно. Содержимое файла index.php приведено в листинге 7.15.

Листинг 7.15

```
<?
// подключение библиотеки xAjax
... ... ...
?>
<html xmlns="http://www.w3.org/1999/xhtml" lang="ru" xml:lang="ru">
<head>
  <script type="text/JavaScript">
    var scrW=screen.width;
    function GetSelectedText ()
    { var selText = "";
      if (window.getSelection)
      { // Firefox, Opera, Google Chrome, Safari
        var selRange = window.getSelection();
        selText = selRange.toString();
      }
      else
      { if (document.selection.createRange)
        { // Internet Explorer
          var range = document.selection.createRange();
```

```
          selText = range.text;}
      }
      return selText;
    }
    // обработчик по onclick
    document.onclick = function (e)
    { var wiki = document.getElementById("wiki");
      var dy=(!e)?document.documentElement.scrollTop:window.pageYOffset;
      var yyy=(!e)?(event.clientY+dy+10):(e.pageY+20);
      var xxx=(!e)?(event.clientX-1):(e.pageX-1);
      xxx=min(xxx,scrW-300);
      wiki.style.left=""+xxx+"px";  wiki.style.top=yyy+"px";
      selText = GetSelectedText (); wiki.style.display="none";
      if(selText.length>0)
      { wiki.style.display="block";
        wiki.innerHTML="<img src='load.gif'>";
        xajax_Get_Info_Wiki(selText);
      }
    }
  </script>
  <?php $xajax->printJavaScript(''); ?>
</head>
<body >
  <div id="wiki" ></div>
  <p><b>Пример к книге, глава 7, API Wikipedia</b></p>
  <?php
    $records=array();
    $url="http://news.yandex.ru/culture.rss";
    $xml = simplexml_load_file($url);
    $records=$xml->xpath('channel/item');
    if (empty($records))
    { echo "Нет новостей "; }
    else
    { foreach ($records as $record)
      { echo "<b>".iconv("utf-8", "windows-1251",$record->title)."
             <a href='".$record->link."'
             target=_blank> >>> </a></b><br>".
             date('d.m.Y H:i:s',strtotime($record->pubDate))."<br><br>".
             iconv("utf-8", "windows-1251",$record->description)."<br>".
             "<hr>";
      }
    }
  ?>
</body>
</html>
```

xAjax-функция `Get_Info_Wiki()` формирует запрос к API Wikipedia. Внимательно читаем описание параметров запроса (**http://en.wikipedia.org/w/api.php**) и выбираем нужные:

- `action` — действие, которое хотим совершить, нам нужен поиск `action=opensearch`;
- `search` — что будем искать, например "Девушка с веслом": `search=Девушка с веслом`
- `prop` — какие характеристики странички хотим получить, нам нужна общая информация о страничке: заглавие, описание: `prop=info`;
- `inprop` — какую дополнительную информацию хотим получить. Нам еще нужна ссылка на страничку Википедии, поэтому `inprop=url`;
- `format` — формат, в котором возвращается результат, для поиска используем XML: `format=xml`.

Формируем запрос к API Wikipedia и получаем ответ. Запрос отправляем через сокет, т. к. API Wikipedia требует передачи заголовка User-Agent, в противном случае ответ не выдается. Далее следует разбор XML-ответа функцией `simplexml_load_string()` и выдача результата (ссылки на статьи Wikipedia) во всплывающее окно (рис. 7.20). При переходе по ссылке попадаем на статью в Wikipedia (рис. 7.21). Функция `Get_Info_Wiki()` расположена в файле `get_info_wiki.php()`, содержимое которого представлено в листинге 7.16.

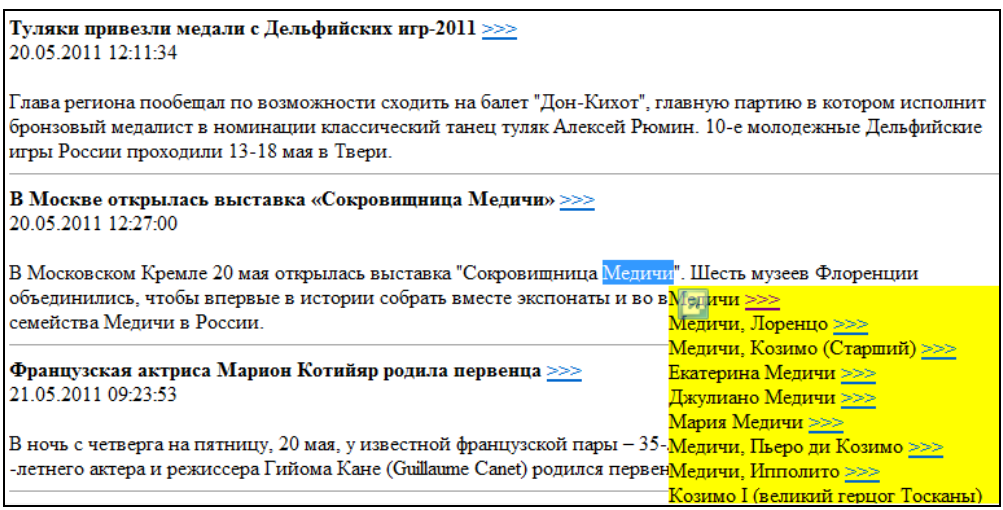


Рис. 7.20. Ответ с API Wikipedia

Листинг 7.16

```
<?php
//
function Get_Info_Wiki($txt)
{ $objResponse = new xajaxResponse();
  //устанавливаем соединение через сокет
  $fp = fsockopen("ru.wikipedia.org", 80, $errno, $errstr, 30);
  if (!$fp)
  { $objResponse->alert("error");
```

```
    return $objResponse;
  }
  else
  { $out = "GET /w/api.php?action=opensearch&search=".urlencode($txt).
           "&prop=info&format=xml&inprop=url HTTP/1.1\r\n";
    $out .= "Host: ru.wikipedia.org\r\n";
    // указывает User-Agent. Без него  будет ошибка
    $out .= "User-Agent: MyCuteBot/0.1\r\n";
    $out .= "Connection: Close\r\n\r\n";
    fwrite($fp, $out);
    $str = '';
    // получает только XML без полученных заголовков сервера
    while (!feof($fp))
    { $tmp_str = fgets($fp, 128);
      if ($str != '' || substr($tmp_str,0,2)=='<?')
        $str .= $tmp_str;
    }
    fclose($fp);
    // парсим строку
    $xml = simplexml_load_string($str);
    $wiki_data=$xml->Section->Item;
    foreach ($wiki_data as $data)
    { $content.=iconv("utf-8", "windows-1251",$data->Text)."
         <a href='".(string)$data->Url."' target='_blank'
          title='".htmlspecialchars((string)$data->Description)."'>
          <b>>>>></b></a><br>";
    }
  }
  $objResponse->assign("wiki","innerHTML",$content);
  return $objResponse;
}
?>
```

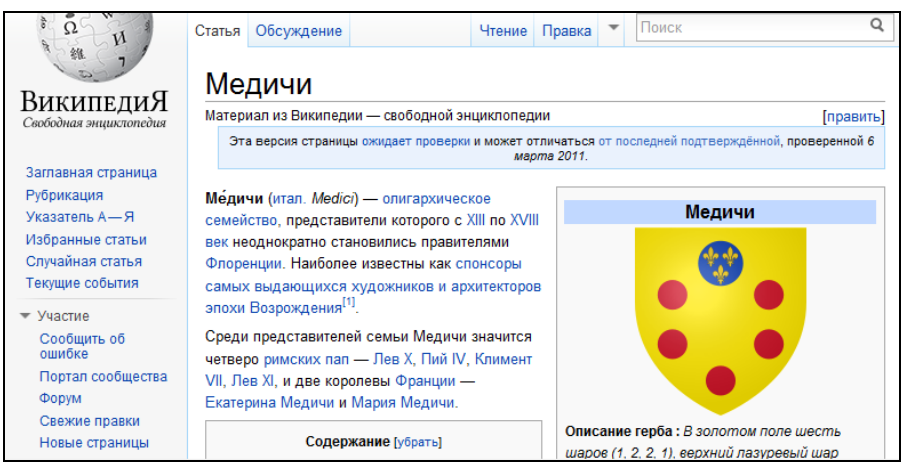


Рис. 7.21. Переход по ссылке на статью в Wikipedia

7.8. API Twitter

Twitter — это онлайн-сервис для ведения (микро)блогов. Почему микро? Сообщение ограничено длиной в 140 символов. Важная особенность Twitter — это возможность общения с читателями — другими блогерами, ведущими свои блоги на твитере. Вы так же можете подписаться на чужие блоги и читать их в единой ленте (как френд-лента в "ЖЖ" ("Живой журнал")). Существует множество программ — twitter-клиентов, превращающих этот сервис в аналог ICQ. Помимо прочего в Twitter блог можно писать и через SMS-сообщения. Краткость, удобство, оперативность, широкий круг читателей — все это делает Twitter популярнейшим сервисом ведения блогов. Чем больше человек пользуются сервисом, тем больше становится спрос на интеграцию программных продуктов с этим сервисом. Поэтому стоит сразу ознакомиться с интерфейсом, чтобы потом сломя голову не искать нужную информацию на просторах Сети. Рассмотрим основы API для работы с твиттером. Описание доступно всем на страничке API-документации (**http://apiwiki.twitter.com/w/page/22554679/Twitter-API-Documentation**). Весь текст на английском языке.

31 августа 2010 года HTTP-аутентификация в твиттере "приказала долго жить". Разработчики решили, что аутентификация с помощью oAuth будет намного эффективнее, а главное — безопаснее. oAuth представляет собой протокол авторизованного доступа к стороннему сервису. Если немного разобраться, то получится, что приложение (веб-приложение, виджет для постинга в твиттер, приложение для iPhone для работы с твиттером и т. д.) авторизуется в сервисе (в твиттере) с помощью (от имени) пользователя. Чтобы приложение могло работать с твиттером, его нужно зарегистрировать на **http://dev.twitter.com/apps/new** (рис. 7.22).

Необходимо заполнить следующие поля:

- **Application Name** — наименование приложения;
- **Description** — описание приложения;
- **Application Website** — сайт приложения;
- **Organization** — ваша организация;
- **Application Type** — тип приложения. **Browser** — это браузерное приложение, т. е. веб-сайт. **Client** — это обычное приложение для любой операционной системы;
- **Callback URL** — URL, на который будет переадресован пользователь после авторизации. Этот пункт заполняем, только если в предыдущем пункте был выбран тип **Browser**;
- **Default Access type** — тип доступа к твиттеру из приложения. **Read & Write** — для чтения и записи. **Read-only** — только чтение. Мы в примерах будем писать твитты, поэтому выберем тип **Read & Write**;
- **Application Icon** — можно загрузить значок приложения в формате JPEG, GIF или PNG, размером не более 700 Кбайт.

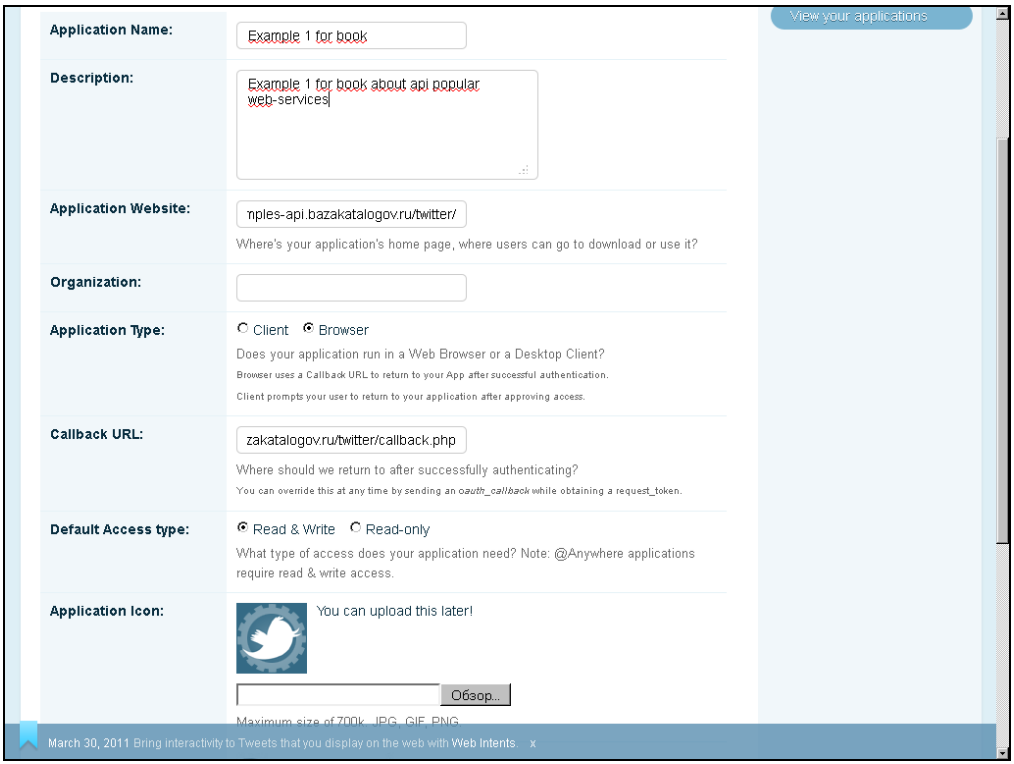


Рис. 7.22. Форма регистрации приложения

Ниже заполняем капчу.

Нажав кнопку **Register application**, регистрируем приложение. После регистрации в разделе **Your apps** появится только что зарегистрированное приложение. В свойствах приложения можно посмотреть данные, которые нам будут нужны для работы: **Consumer key**, **Consumer secret** и **Registered Callback URL** (рис. 7.23).

На данный момент уже существует множество достаточно приемлемых решений для работы с твиттером посредством oAuth. Мы будем использовать библиотеку twitteroauth. Скачать ее можно по адресу **https://github.com/abraham/twitteroauth**, нажав кнопку **Downloads** и выбрав нужный релиз. Рекомендую выбрать последний. Дата последнего релиза немного смущает (март 2010 г.), но, как оказалось, библиотека работает вполне нормально, а главное — имеет простую и понятную структуру. После распаковки архива получится много файлов, из которых нас интересует только config.php. Вся настройка библиотеки состоит в замене значений констант `CONSUMER_KEY`, `CONSUMER_SECRET` и `OAUTH_CALLBACK` своими, которые можно посмотреть в свойствах приложения (см. рис. 7.23).

Создадим небольшой проект, который позволит после oAuth-авторизации пользователя выполнять следующие действия:

- ❒ просмотр timeline-сообщений;
- ❒ просмотр, добавление, удаление tweet пользователя;

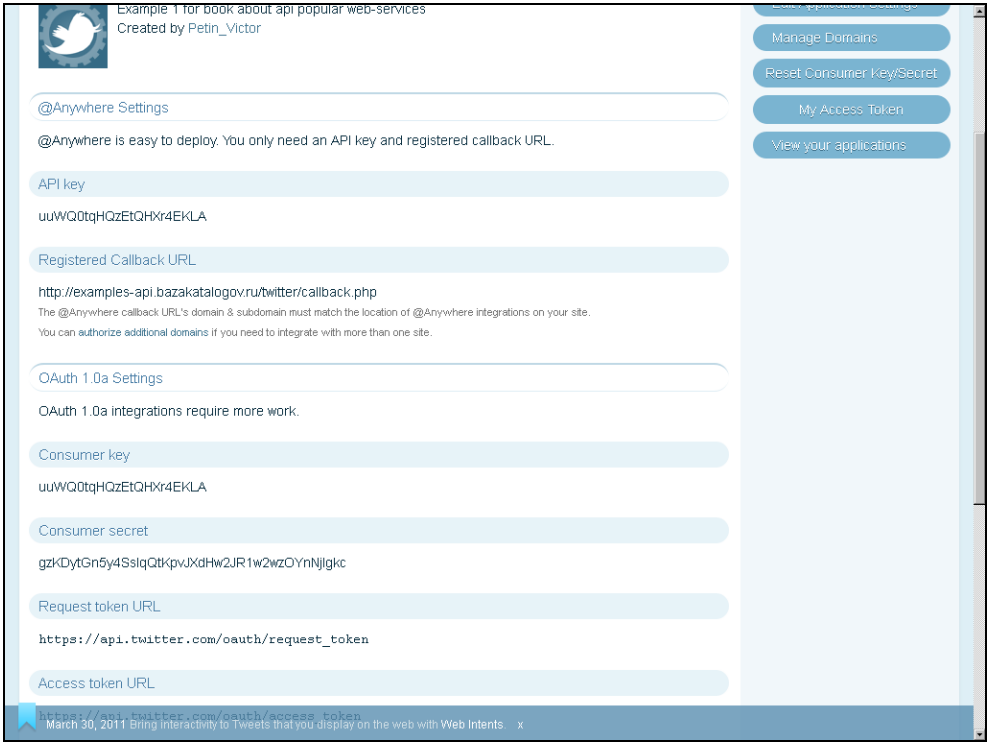


Рис. 7.23. Регистрация приложения

- ❒ просмотр списка друзей (friends);
- ❒ просмотр timeline пользователей.

В проекте будут использованы библиотеки xAjax и twitteroauth. В Интернете данный пример можно посмотреть по адресу **http://examples-api.bazakatalogov.ru/twitter/index.php**. На компакт-диске пример находится в папке glava_07\twitter. При первом заходе на страницу нас перебрасывает на форму oAUTH-авторизации (рис. 7.24).

После успешной авторизации (если вы не были авторизованы) твиттер вернет пользователя на ту самую страницу, которая прописана в параметре **Callback URL**, одновременно передавая уникальные токены — oauth_token и oauth_token_secret.

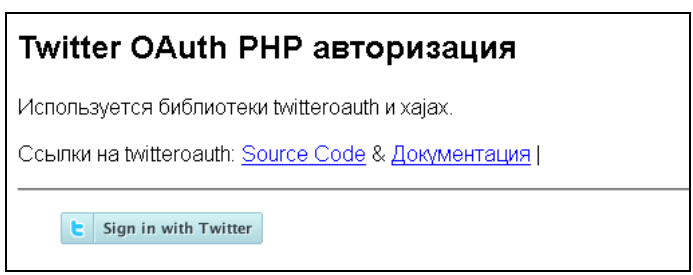


Рис. 7.24. Форма oAUTH-авторизации

Файл index.php начинается с инициализации библиотеки:

```
session_start();
require_once('twitteroauth/twitteroauth.php');
require_once('config.php');
```

Любые скрипты, которые будут обращаться с API, должны начинаться с этих строк. Далее идет проверка на присутствие в текущей сессии каких-либо токенов пользователя. Если их нет, то мы будем перенаправлены на ту самую страницу с кнопкой **Sign in with Twitter** — connect.php. Здесь проверяется, заполнены ли нами уникальные ключи нашего приложения, и если да, то очередной редирект ведет нас к скрипту redirect.php. На этой странице генерируется ссылка для связи аккаунта пользователя с приложением (называется URL авторизации). Для того чтобы получать такую ссылку, также требуются токены — они называются *временными*. Содержание скрипта redirect.php приведено в листинге 7.17.

Листинг 7.17

```
<?php
/* Инициализация библиотеки. */
session_start();
require_once('twitteroauth/twitteroauth.php');
require_once('config.php');
/* Создание объекта TwitterOAuth на базе ключей нашего приложения */
$connection = new TwitterOAuth(CONSUMER_KEY, CONSUMER_SECRET);
/* Получаем временные токены для авторизации. */
$request_token = $connection->getRequestToken(OAUTH_CALLBACK);
/* Записываем временные токены в массив */
$_SESSION['oauth_token'] = $token = $request_token['oauth_token'];
$_SESSION['oauth_token_secret'] = $request_token['oauth_token_secret'];
/* Если связь с сайтом твиттера не дала ошибку (код 200),
   то генерируем ссылку для авторизации*/
switch ($connection->http_code) {
  case 200:
    /* Генерация ссылки и перенаправление на нее. */
    $url = $connection->getAuthorizeURL($token);
    header('Location: ' . $url);
    break;
  default:
    /* Если соединение было неудачным — выдаем ошибку. */
    echo 'Could not connect to Twitter. Refresh the page or try again later.';
}
?>
```

Мы будем перенаправлены на страницу с кнопками **Allow**/**Deny**, разрешаем приложению пользоваться нашим аккаунтом, и Twitter направляет нас на страницу callback.php. Здесь мы получаем переменные текущей сессии `$_SESSION['oauth_`

token'] и $_SESSION['oauth_token_secret'], которые используем для генерации окончательных токенов для работы приложения от имени пользователя. Содержимое файла callback.php представлено в листинге 7.18.

Листинг 7.18

```
<?php
/* Start session and load lib */
session_start();
require_once('twitteroauth/twitteroauth.php');
require_once('config.php');
/* Проверка, не устарела ли текущая сессия (в случае неактивности
   пользователя)*/
if (isset($_REQUEST['oauth_token']) &&
          $_SESSION['oauth_token'] !== $_REQUEST['oauth_token']) {
  $_SESSION['oauth_status'] = 'oldtoken';
  header('Location: ./clearsessions.php');
}
/* Создаем объект TwitteroAuth с текущей сессией */
$connection = new TwitterOAuth(CONSUMER_KEY, CONSUMER_SECRET,
    $_SESSION['oauth_token'], $_SESSION['oauth_token_secret']);
/* Запрашиваем access-токены пользователя для дальнейшего постоянного
   использования*/
$access_token = $connection->getAccessToken($_REQUEST['oauth_verifier']);
/* Сохраняем их в массив. */
$_SESSION['access_token'] = $access_token;
unset($_SESSION['oauth_token']);
unset($_SESSION['oauth_token_secret']);
/* Если ошибки HTTP не было, перенаправляем на первоначальный файл index.php*/
if (200 == $connection->http_code) {
  $_SESSION['status'] = 'verified';
  header('Location: ./index.php');
} else {
  header('Location: ./clearsessions.php');
}
```

После возвращения на index.php у нас есть ключи приложения и токены пользователя. Можем обращаться к любым ресурсам, описанным в API Twitter. В нашем примере мы подгружаем библиотеку xAjax, создаем разметку страницы (блоки div id=zag1, zag2, content1, content2) и по событию onload загрузки страницы вызываем xAjax-функцию xajax_Ini(). Функция Ini() будет выводить в левый блок timeline пользователя, в правый — твитты пользователя и форму для добавления нового твитта. Сначала получаем access_token из сессии, создаем объект класса TwitterOauth и передаем ему все данные. Вызываем запрос account/verify_ credentials, возвращающий информацию о пользователе, который был авторизован. Получать данные от сервера будем в формате XML ($connection->format='xml'). На

основе идентификатора пользователя получим сообщения из home_timeline ленты (запрос statuses/home_timeline) и из его ленты (запрос statuses/user_timeline). При этом будем получать 10 записей с первой страницы (устанавливаем значения в массиве параметров запроса). Возвращаемые данные выводим в блоки `content1` и `content2` (рис. 7.25). xAjax-функция `Ini()` находится в файле x_ini.php, содержимое которого представлено в листинге 7.19.

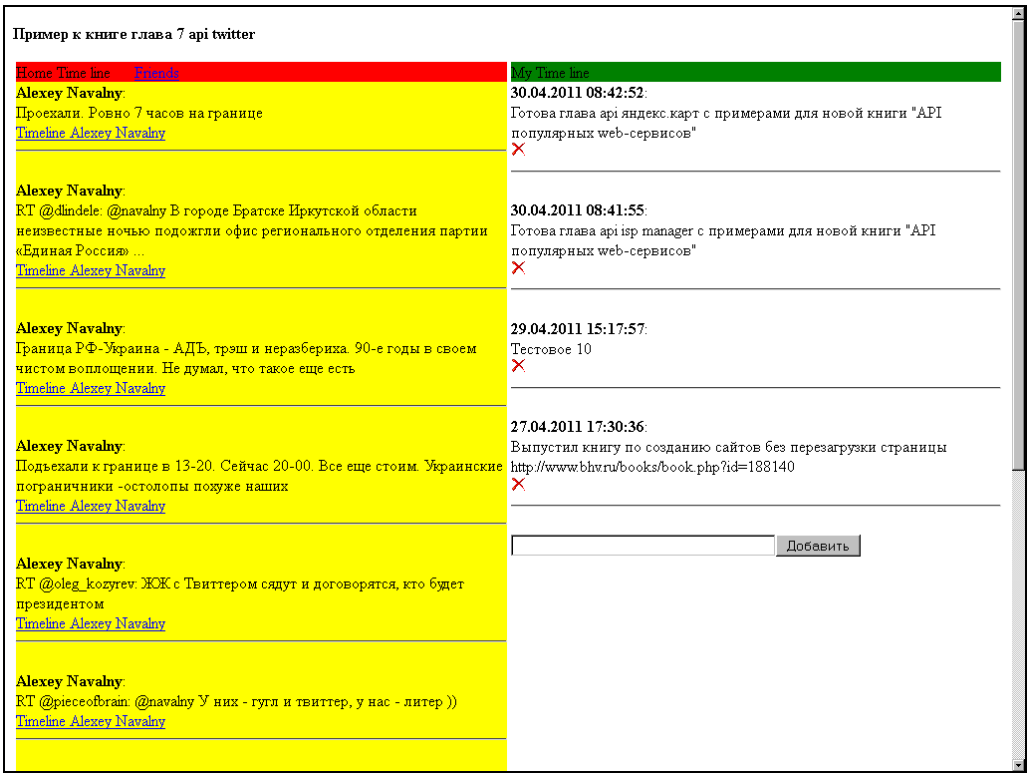


Рис. 7.25. Вывод лент home_timeline и user_timeline

Листинг 7.19

```
<?php
// начальная загрузка
// home_timeline, user_timeline
function Ini()
{ $objResponse = new xajaxResponse();
  // Получаем access tokens из сессии
  $access_token = $_SESSION['access_token'];
  // Создаем объект класса TwitterOauth и передаем ему все данные
  $connection = new TwitterOAuth(CONSUMER_KEY,
  CONSUMER_SECRET, $access_token['oauth_token'],
  $access_token['oauth_token_secret']);
```

```
  //account/verify_credentials
  $connection->format='xml';
  $account=$connection->get('account/verify_credentials');
  $xml = simplexml_load_string($account);
  $my_id = $xml->id;
  // Timeline
  $connection->format='xml';
  $home_timeline=$connection->get('statuses/home_timeline',
                 array('count'=>10, 'page'=>1));
  $xml = simplexml_load_string($home_timeline);
  foreach ($xml->status as $val)
  { $tweet = iconv("utf-8", "windows-1251",$val->text);
    $author = iconv("utf-8", "windows-1251",$val->user->name);
    $id = $val->user->id;
    $content1.="<b>".$author."</b>:<br> ".$tweet."<br>";
    if((int)$my_id!=(int)$id)
      $content1.="<a href='JavaScript:void();'
         onclick='xajax_Timeline_Friend(".$id.")'>
         Timeline ".$author."</a>";
    $content1.="<hr><br>";
  }
// My timeline
  $connection->format='xml';
  $user_timeline=$connection->get('statuses/user_timeline',
                 array('id'=>$my_id,'count'=>10, 'page'=>1));
  $xml = simplexml_load_string($user_timeline);
  foreach ($xml->status as $val)
  { $tweet = iconv("utf-8", "windows-1251",$val->text);
    $data = date('d.m.Y H:i:s',strtotime($val->created_at));
    $id = "id".$val->id;
    $content2.="<b>".$data."</b>:<br> ".$tweet."<br>";
    $content2.="<a href='JavaScript:void();'
               onclick='xajax_Delete_Tweet(\"".$id."\")'>
               <img src='delete.png'></a><hr><br>";
  }
  // Форма добавления твитта
  $content2.="<form id='FormAddTweet' action='JavaScript:void(null);'
          onsubmit='xajax.$(\"ButtonFormAddTweet\").disabled=true;
     xajax.$(\"ButtonFormAddTweet\").value=\"Подождите...\";
          xajax_Add_Tweet(xajax.getFormValues(\"FormAddTweet\"));'>";
  $content2.="<input type='text' name='text' size=40 maxlength=140 value=''>";
  $content2.="<input type='submit' id='ButtonFormAddTweet'
                     value='Добавить'>";
  // вывод контента в блоки
  $objResponse->assign("content1","innerHTML",$content1);
  $objResponse->assign("content2","innerHTML",$content2);
  return $objResponse;
}
?>
```

В правом блоке под лентой сообщений пользователя (user_timeline) расположена форма добавления нового твитта в свою ленту. Пишем сообщение в форму и нажимаем кнопку **Добавить**. При этом вызывается xAjax-функция `Add_Tweet()`. Функция отправляет POST-запрос statuses/update, передавая в массиве параметров идентификатор авторизованного пользователя и добавленное сообщение. Затем делается задержка (`sleep(4)`) и запрос обновленной ленты пользователя. Смотрим, что получается (рис. 7.26 и 7.27). xAjax-функция `Add_Tweet()` расположена в файле x_add_tweet.php, содержимое которого представлено в листинге 7.20.

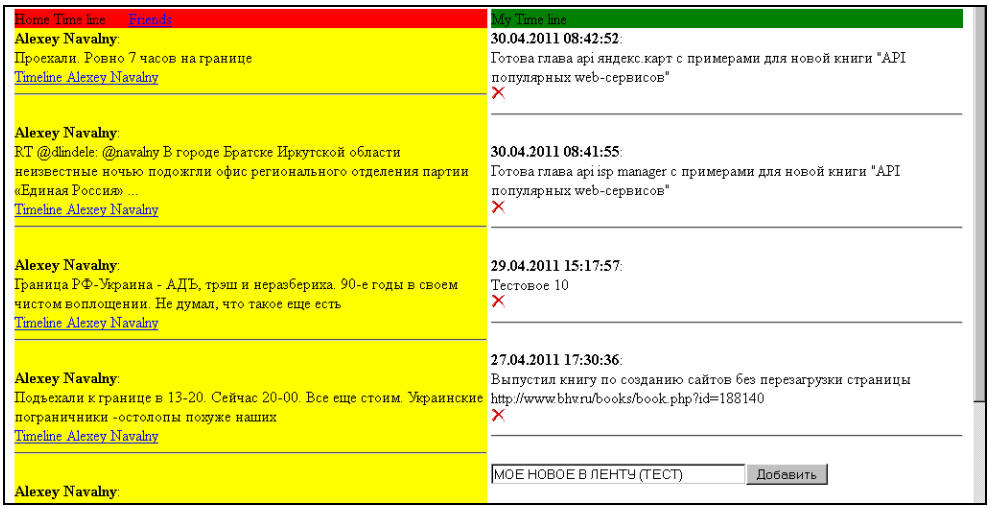


Рис. 7.26. Создание нового сообщения

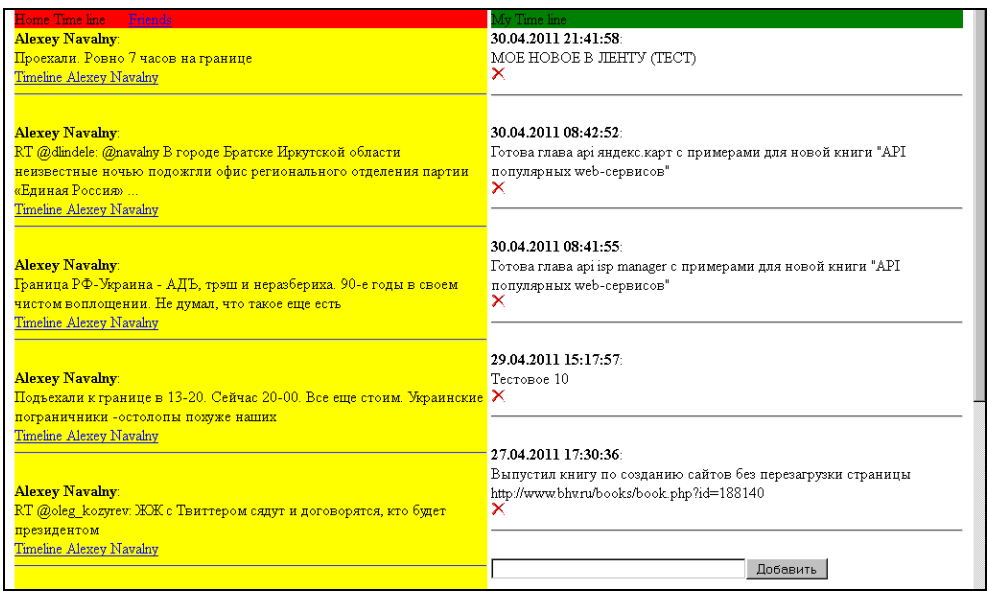


Рис. 7.27. Новое сообщение в ленте пользователя

Листинг 7.20

```
<?php
// Add_Tweet — добавить tweet
function Add_Tweet($Id)
{ $objResponse = new xajaxResponse();
  // Получаем access tokens из сессии
  $access_token = $_SESSION['access_token'];
  // Создаем объект класса TwitterOauth и передаем ему все данные
  $connection = new TwitterOAuth(CONSUMER_KEY,
  CONSUMER_SECRET, $access_token['oauth_token'],
  $access_token['oauth_token_secret']);
  //account/verify_credentials
  $connection->format='xml';
  $account=$connection->get('account/verify_credentials');
  $xml = simplexml_load_string($account);
  $my_id = $xml->id;
  // добавить твитт
  $connection->post('statuses/update', array('user'=>$my_id,
                    'status'=>iconv("windows-1251","utf-8",$Id[text])));
  sleep(4);
  // My timeline
  $connection->format='xml';
  $content2.="";
  $user_timeline=$connection->get('statuses/user_timeline',
      array('id'=>$my_id,'count'=>10, 'page'=>1));
  $xml = simplexml_load_string($user_timeline);
  foreach ($xml->status as $val)
  { $tweet = iconv("utf-8", "windows-1251",$val->text);
    $data = date('d.m.Y H:i:s',strtotime($val->created_at));
    $id = "id".$val->id;
    $content2.="<b>".$data."</b>:<br> ".$tweet."<br>";
    $content2.="<a href='JavaScript:void();'
         onclick='xajax_Delete_Tweet(\"".$id."\")'>
              <img src='delete.png'></a><hr><br>";
  }
  // Форма добавления твитта
  $content2.="<form id='FormAddTweet' action='JavaScript:void(null);'
        onsubmit='xajax.$(\"ButtonFormAddTweet\").disabled=true;
  xajax.$(\"ButtonFormAddTweet\").value=\"Подождите...\";
        xajax_Add_Tweet(xajax.getFormValues(\"FormAddTweet\"));'>";
  $content2.="<input type='text' name='text' size=40 maxlength=140 value=''>";
  $content2.="<input type='submit' id='ButtonFormAddTweet'
              value='Добавить'>";
  // вывод контента
  $objResponse->assign("content2","innerHTML",$content2);
```

```
  return $objResponse;
}
?>
```

Рядом с каждым сообщением ленты пользователя (user_timeline) расположена ссылка для удаления сообщения. При щелчке по ссылке вызывается xAjax-функция `Delete_Tweet()`. Функция отправляет POST-запрос 'statuses/destroy/:id. Затем делается задержка (`sleep()`) и запрос обновленной ленты пользователя. Вид ленты после удаления 2-х сообщений представлен на рис. 7.28 (сравните с рис. 7.27). xAjax-функция `Delete_Tweet()` расположена в файле x_delete_tweet.php, содержимое которого представлено в листинге 7.21.

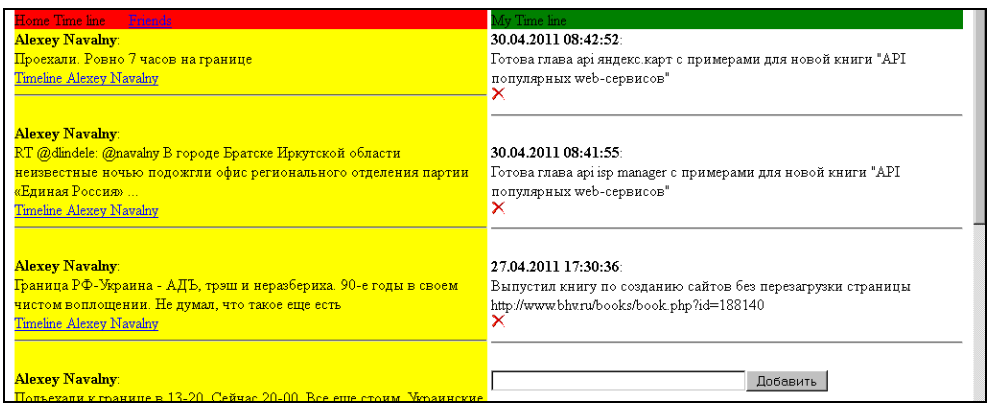


Рис. 7.28. Лента users_timeline после удаления двух сообщений

Листинг 7.21

```
<?php
// Delete_Tweet — удалить tweet
function Delete_Tweet($arg)
{ $objResponse = new xajaxResponse();
  // Получаем access tokens из сессии
  $access_token = $_SESSION['access_token'];
  // Создаем объект класса TwitterOauth и передаем ему все данные
  $connection = new TwitterOAuth(CONSUMER_KEY,
  CONSUMER_SECRET, $access_token['oauth_token'],
  $access_token['oauth_token_secret']);
  //account/verify_credentials
  $connection->format='xml';
  $account=$connection->get('account/verify_credentials');
  $xml = simplexml_load_string($account);
  $my_id = $xml->id;
  // удалить твитт
  $connection->post('statuses/destroy/'.str_replace("id","",$arg));
  sleep(5);
```

```
  // My timeline
  $connection->format='xml';
  $content2.="";
  $user_timeline=$connection->get('statuses/user_timeline',
      array('id'=>$my_id,'count'=>10, 'page'=>1));
  $xml = simplexml_load_string($user_timeline);
  foreach ($xml->status as $val)
  { $tweet = iconv("utf-8", "windows-1251",$val->text);
    $data = date('d.m.Y H:i:s',strtotime($val->created_at));
    $id = "id".$val->id;
    $content2.="<b>".$data."</b>:<br> ".$tweet."<br>";
    $content2.="<a href='JavaScript:void();'
         onclick='xajax_Delete_Tweet(\"".$id."\")'>
         <img src='delete.png'></a><hr><br>";
  }
  // Форма добавления твитта
  $content2.="<form id='FormAddTweet' action='JavaScript:void(null);'
        onsubmit='xajax.$(\"ButtonFormAddTweet\").disabled=true;
  xajax.$(\"ButtonFormAddTweet\").value=\"Подождите...\";
        xajax_Add_Tweet(xajax.getFormValues(\"FormAddTweet\"));'>";
  $content2.="<input type='text' name='text' size=40 maxlength=140 value=''>";
  $content2.="<input type='submit' id='ButtonFormAddTweet' value='Добавить'>";
  // вывод контента
  $objResponse->assign("content2","innerHTML",$content2);
  return $objResponse;
}
?>
```

Теперь вернемся к левому блоку. В ленте home_timeline под сообщением каждого пользователя имеется ссылка для просмотра ленты данного пользователя (последние 10 сообщений). По ссылке вызывается xAjax-функция `Timeline_Friend()`. Функция отправляет GET-запрос statuses/user_timeline, передавая в массиве параметров идентификатор пользователя, количество сообщений и номер страницы. Полученный результат при переходе по ссылке на сообщении MedvedevRussia представлен на рис. 7.29. xAjax-функция `Timeline_Friend()` расположена в файле x_timeline_friend.php, содержимое которого приведено в листинге 7.22.

Листинг 7.22

```
<?php
// Timeline friends
function Timeline_Friend($id)
{ $objResponse = new xajaxResponse();
  // Получаем access tokens из сессии
  $access_token = $_SESSION['access_token'];
  // Создаем объект класса TwitterOauth и передаем ему все данные
  $connection = new TwitterOAuth(CONSUMER_KEY,
  CONSUMER_SECRET, $access_token['oauth_token'],
  $access_token['oauth_token_secret']);
```

```
  // User Timeline
  $connection->format='xml';
  $content1.="";
  $user_timeline=$connection->get('statuses/user_timeline',
     array('id'=>$id,'count'=>10, 'page'=>1));
  $xml = simplexml_load_string($user_timeline);
  foreach ($xml->status as $val)
  { $tweet = iconv("utf-8", "windows-1251",$val->text);
    $author = iconv("utf-8", "windows-1251",$val->user->name);
    $id = $val->user->id;
    $content1.="<b>".$author."</b>:<br> ".$tweet."<br><hr><br>";
  }
  $zag1="<a href='JavaScript:void();' onclick='xajax_Home_Timeline()'>
     Home  Timeline</a>     
     <a href='JavaScript:void();' onclick='xajax_Friends()'>Friends</a>";
  // вывод контента
  $objResponse->assign("zag1","innerHTML",$zag1);
  $objResponse->assign("content1","innerHTML",$content1);
  return $objResponse;
}
?>
```

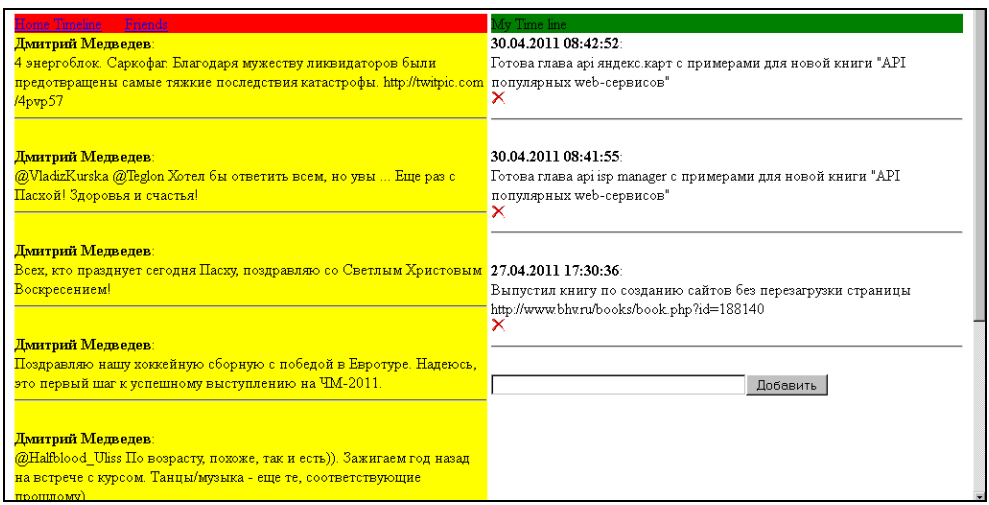


Рис. 7.29. Просмотр ленты MedvedevRussia

В примере есть еще друзья. В API Twitter не предусмотрена такая функция, поэтому получим список за два прохода. Сначала получаем список идентификаторов всех друзей (GET-запрос statuses/friends), а затем для каждого идентификатора получим имя, количество фолловеров[1] и количество друзей (рис. 7.30). xAjax-функция

[1] Фолловер — человек, который подписывается на ленту рассылки новостей, коротких цитат и прочей информации определенного автора в Твиттере.

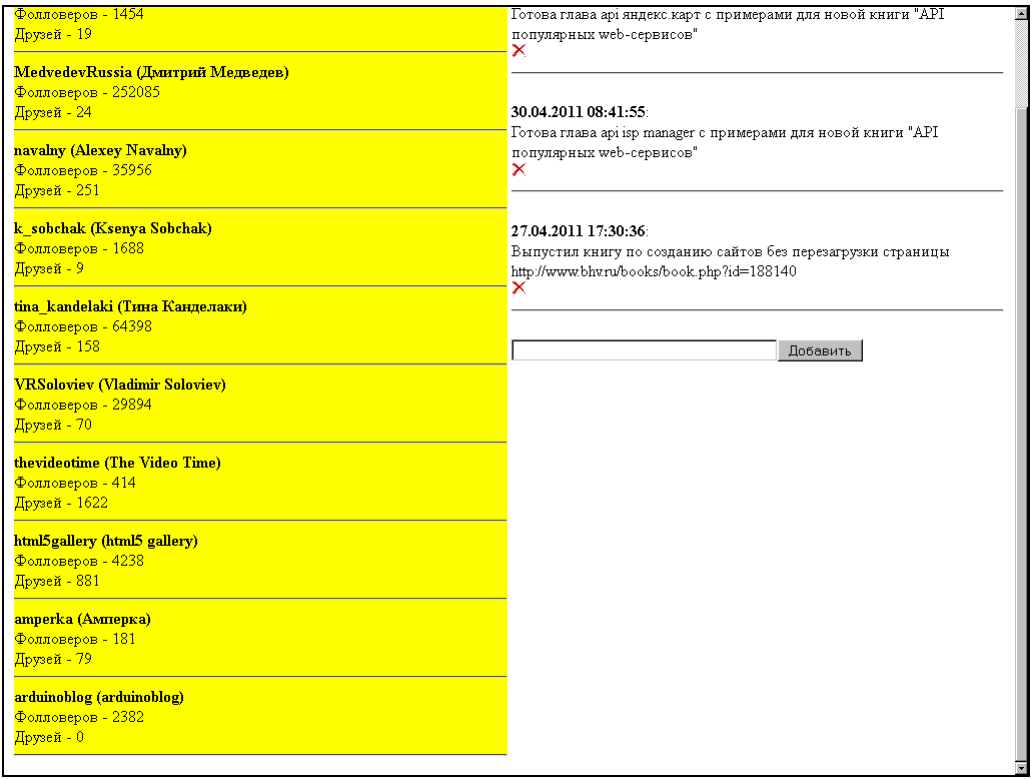


Рис. 7.30. Просмотр данных друзей

`Timeline_Friend()`, выполняющая эти действия, расположена в файле x_timeline_friend.php, содержимое которого представлено в листинге 7.23.

Листинг 7.23

```
<?php
// Friends — список
function Friends()
{ $objResponse = new xajaxResponse();
  // Получаем access tokens из сессии
  $access_token = $_SESSION['access_token'];
  // Создаем объект класса TwitterOauth и передаем ему все данные
  $connection = new TwitterOAuth(CONSUMER_KEY,
  CONSUMER_SECRET, $access_token['oauth_token'],
  $access_token['oauth_token_secret']);
  //account/verify_credentials
  $connection->format='xml';
  $account=$connection->get('account/verify_credentials');
  $xml = simplexml_load_string($account);
  $my_id = $xml->id;
```

```
  // список друзей
  $connection->format='xml';
  $content1.="";
  $friends=$connection->get('statuses/friends',array('count'=>10, 'page'=>1));
  $xml = simplexml_load_string($friends);
  foreach ($xml as $val)
  { $id = $val->id;
    $connection->format='xml';
    $user=$connection->get('users/show',array('id'=>(int)$id));
    $xml1 = simplexml_load_string($user);
    $content1.="<b>".$xml1->screen_name." (".iconv("utf-8",
       "windows-1251",$xml1->name).")</b><br>";
    $content1.="Фолловеров — ".$xml1->followers_count."<br>";
    $content1.="Друзей — ".$xml1->friends_count."<br>";
    $content1.="<hr>";
  }
  $zag1="<a href='JavaScript:void();' onclick='xajax_Home_Timeline()'>
      Home Timeline</a>      Friends";
  // вывод контента
  $objResponse->assign("zag1","innerHTML",$zag1);
  $objResponse->assign("content1","innerHTML",$content1);
  return $objResponse;
}
?>
```

Мы рассмотрели самую малую часть возможностей API Twitter. На самом деле, их гораздо больше, и рассмотрение всех возможностей API Twitter заслуживает отдельной книги. Интересна и возможность создания портала, где у пользователей будет доступ по API к своим профилям в нескольких социальных сетях (например, "ВКонтакте", "Мой Мир", Mail.ru, "Одноклассники", Facebook, Twitter). У автора реализация подобной идеи в ближайших планах. Возможно, это станет темой новой книги.

7.9. API Loginza

Регистрация пользователей давно стала рутинной операцией для многих и многих интернет-сервисов, будь то форумы, блоги или новостные сайты. Зарегистрированные пользователи получают доступ к дополнительным функциям сервиса. Активный интернет-путешественник регулярно использует десятки сайтов. И для каждого нужен свой пароль, свое имя пользователя — это не очень удобно. Добавить пользовательского удобства регистрации призвана технология OpenID. Она позволяет интернет-пользователю единым образом авторизовываться на множестве интернет-сервисов (веб-сайтов). Конечно, эти сервисы должны поддерживать OpenID. Если при использовании "классической" системы авторизации пользователь вводит в форму на сайте пару из идентификатора (логина) и соответствующего ему пароля, то в случае OpenID используется всего один объект — контролируемый пользова-

телем URL. Например, пользователю достаточно ввести адрес своей персональной страницы — **http://test.ru/mypage/**. Такая система авторизации работает благодаря автоматизированным механизмам, позволяющим авторизующему серверу выяснить, что пользователь действительно контролирует заявленный URL. Механизм работы OpenID требует наличия удостоверяющей стороны, которая могла бы подтвердить по запросу сервиса, на котором пытается авторизоваться пользователь, что реквизиты предоставлены верно. Поэтому для успешного использования OpenID требуется наличие у пользователя аккаунта на том или ином сервере OpenID. При этом на подобном сервере пользователю нужно зарегистрироваться только один раз, в дальнейшем сервер самостоятельно будет подтверждать авторизацию пользователя по запросам других сайтов. Крупные социальные сети выступают в качестве серверов OpenID-авторизации. Поэтому, имея аккаунт в какой-нибудь социальной сети, возможно, вы можете авторизоваться на сайтах, поддерживающих OpenID-авторизацию. Сайт зависимой стороны перенаправляет пользователя на сайт провайдера. Сайт провайдера запрашивает у пользователя подтверждение, действительно ли пользователь желает предоставить информацию о своей учетной записи. Если пользователь соглашается, то сайт провайдера перенаправляет пользователя обратно на сайт зависимой стороны. При обратном перенаправлении провайдер передаст информацию о пользователе зависимой стороне. Технология OpenID позволяет передавать в качестве информации о пользователе не только идентификатор этого пользователя (таким идентификатором служит URL), но и ряд других данных. Например, полное имя и фамилию пользователя, сетевой псевдоним пользователя, адрес e-mail и др.

Организация на своем сайте OpenID-авторизации требует знаний. Конечно, существует множество библиотек, облегчающих эту задачу. Однако наиболее удачное, на мой взгляд, решение предоставляет сервис Loginza (**http://loginza.ru**). Loginza — это интерактивный JavaScript-виджет, предоставляющий посетителям сайтов информацию о том, где он установлен, широкий список вариантов аутентификации через учетные записи распространенных веб-порталов и сервисов (Яндекс, Google, Rambler, "ВКонтакте", LiveJournal, Facebook, Twitter, Mail.ru, WebMoney и др.). У каждого провайдера свой механизм авторизации и передачи данных профиля пользователя. Loginza преобразует эти данные в формат, который просто анализировать и использовать. Посмотрим, как организована OpenID-авторизация при помощи виджета Loginza на одном из действующих сайтов. Переходим по ссылке **http://weatheralarm.ru** (рис. 7.31).

Для использования виджета на странице размещен код, представленный в листинге 7.24.

Листинг 7.24

```
<script src="http://loginza.ru/js/widget.js" type="text/JavaScript">
</script>
<a href="https://loginza.ru/api/widget?
token_url=http://weatheralarm.ru/index1.php" class="loginza">
<img src="http://loginza.ru/img/providers/yandex.png">
```

```
<img src="http://loginza.ru/img/providers/google.png>
<img src="http://loginza.ru/img/providers/vkontakte.png">
<img src="http://loginza.ru/img/providers/mailru.png">
<img src="http://loginza.ru/img/providers/twitter.png">
<img src="http://loginza.ru/img/providers/webmoney.png">
</a>
```

После перехода по ссылке мы попадаем на форму выбора сервера OpenID (рис. 7.32).

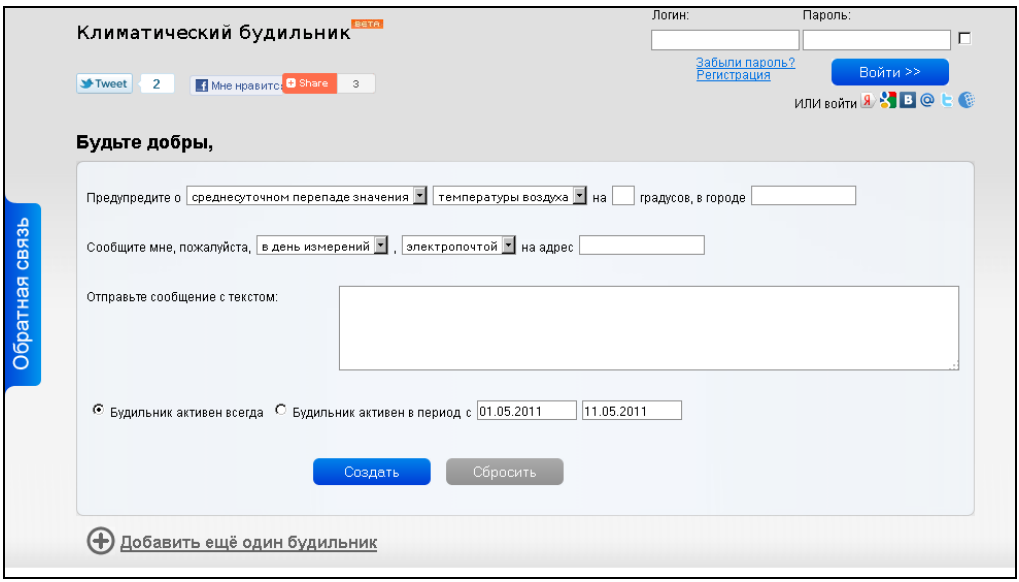


Рис. 7.31. Виджет Loginza на сайте

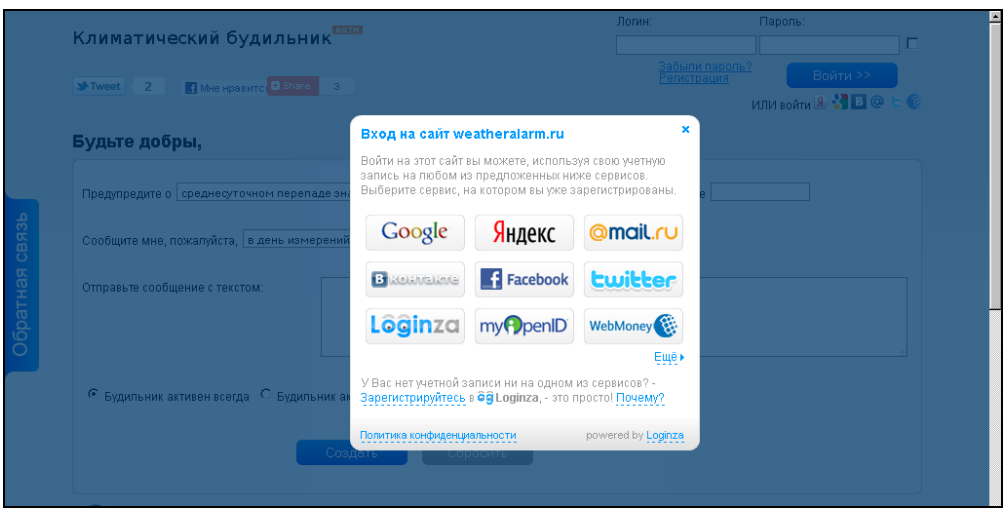


Рис. 7.32. Форма выбора сервера OpenID-авторизации

После выбора сервера OpenID-авторизации (Яндекс) нас перебрасывает на страницу запроса авторизации, где после ввода данных (рис. 7.33) и подтверждения авторизации (рис. 7.34) мы попадаем на страницу **http://weatheralarm.ru/index1.php**, указанную в параметре `token_url` (см. листинг 7.24). На эту страницу в POST передается уникальный ключ `token`, необходимый для получения профиля авторизован-

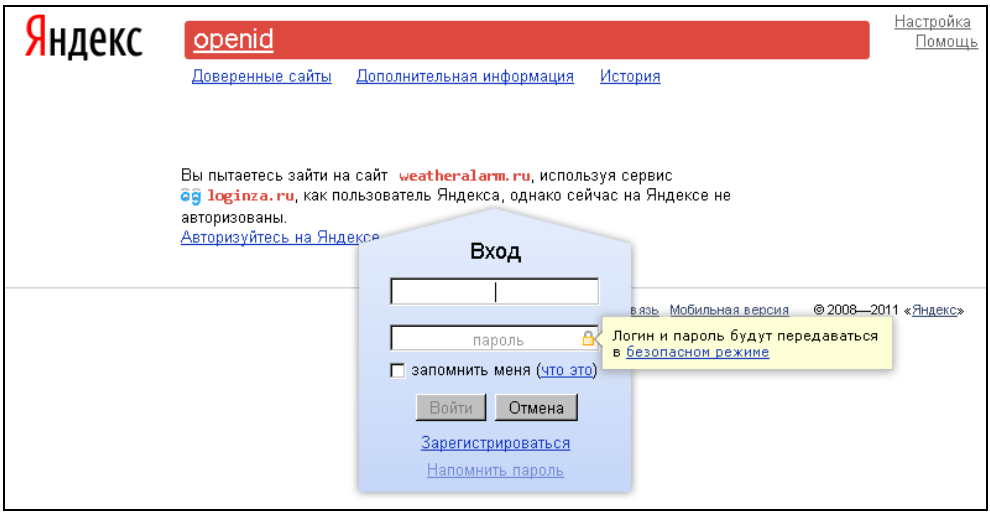


Рис. 7.33. Форма авторизации на OpenID-сервере Яндекс

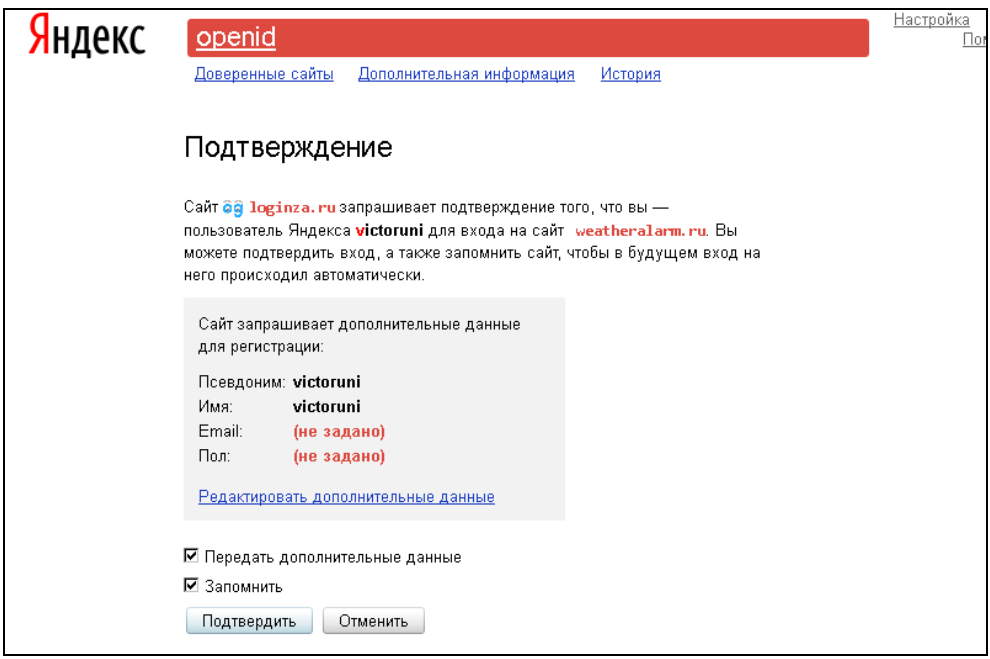


Рис. 7.34. Подтверждение авторизации

ного пользователя. Для того чтобы получить профиль, необходимо сделать межсерверный запрос на URL и получить данные в формате JSON. В файле index1.php должен присутствовать код, представленный в листинге 7.25.

Листинг 7.25

```
$url_auth="http://loginza.ru/api/authinfo?token=".$_POST[token];
$data_json=file_get_contents($url_auth);
$arr_auth = json_decode($data_json);
$nickname=$arr_auth->{'nickname'};
$provider=$arr_auth->{'provider'};
```

Заключение

Создание веб-сайтов требует от разработчиков немало опыта и навыков. При этом они должны постоянно следить за новшествами и быть в курсе современных технологий. Повышение привлекательности интернет-проектов идет на данном этапе за счет внедрения все большего функционала, многое из которого уже не может быть реализовано с нуля, собственными силами. К счастью, многие интернет-сервисы позволяют использовать свои возможности на сторонних сайтах, предоставляя доступ с помощью API. Количество сервисов, предоставляющих публичные API, стремительно растет, что помогает владельцам сайтов применять возможности этих сервисов на своих площадках для увеличения функционала и привлекательности собственных проектов. Надеюсь, изложенный в книге материал помог вам разобраться с особенностями использования API рассмотренных популярных веб-сервисов. В книге рассмотрено создание нескольких больших проектов, готовых к размещению в сети, а также небольших примеров, которые вы можете внедрить на свои сайты. Все проекты выполнены по технологии Ajax с использованием популярных библиотек xAjax и jQuery, позволяющих создавать проекты без перезагрузки страницы.

В случае возникновения вопросов пишите на электронную почту **victoruni@km.ru** и **kmvnews@bk.ru** или заходите в блог **http://goodtovars.ru/blog**.

Буду рад, если изученный материал окажется полезен вам при написании собственных проектов.

Успехов вам и всего доброго!

ПРИЛОЖЕНИЕ

Описание компакт-диска

Компакт-диск содержит все рассматриваемые в книге примеры приложений, чьи коды и фрагменты кодов приведены в тексте. Материал компакт-диска можно скачать по ссылке: **ftp://85.249.45.166/9785977507431.zip**.

Все примеры распределены по папкам в соответствии с главами. Так, примеры для *главы 1* размещены в папке glava_01, примеры для *главы 2* — в папке glava_02 и т. д.

В свою очередь папки, соответствующие главам, могут содержать вложенные каталоги, в которых хранятся файлы конкретных примеров. Ссылки на эти файлы и каталоги представлены в тексте книги.

Более подробное описание представлено в следующей таблице.

Таблица П1. *Описание компакт-диска*

Каталог	Вложенный каталог	Описание примера
glava_01	1	Сайт — тренировочный стенд для изучения xAjax
	2-1	Форма регистрации с проверкой правильности заполнения полей "на лету"
	2-2	Пример динамической загрузки `select`-элементов
	2-3	Многоуровневый неоднородный каталог
	2-4	Динамическое управление количеством полей формы
	2-5	Пример динамической подгрузки на страницу плагина jQuery
	2-6	Совместное использование виджетов Tabs и Accordion
glava_02	yandex_blog	Сервис-тренажер составления запросов при поиске по блогам с выдачей результатов запроса
	yandex_fotki	Галерея фотографий пользователя с использованием API Яндекс.Фоток
glava_03	3-2-2	Пример динамического управления встроенными элементами на созданной карте
	3-2-4	Пример динамического управления внешними элементами на созданной карте

Таблица П1 *(окончание)*

Каталог	Вложенный каталог	Описание примера
	3-2-5	Пример динамического управления кнопкой на панели элементов карты
	3-4-1	Пример динамического управления свойствами метки
	3-4-2	Применение пользовательских значков для меток
	3-4-3	Пример динамического управления свойствами ломаной линии
	3-4-4	Пример динамического управления свойствами многоугольника
glava_04	4-1	Сайт каталога предприятий России, структурированный по регионам и видам деятельности
	4-2	Сайт учета заказов такси
	4-3	Карта местности с несколькими слоями пользовательских карт
	4-4	Создание, редактирование меток для карты местности с несколькими слоями пользовательских карт
	4-5	Модифицированный вариант примера 4-4 с возможностью поиска
glava_05	api_isp_manager	Сайт-тренажер для изучения запросов к API ISPmanager
glava_06	hosting	Личный кабинет для сайта хостинговой компании
	Файл hosting.sql	Дамп базы данных с заполнением таблиц
glava_07	google_js_language	Пример чтения RSS-ленты газеты "Times", вывод новостей на языке оригинала (английский)
	google_search	Пример использования поиска Google на собственном сайте
	google_static_chart	Построение диаграмм для сравнения количества районов, городов, прочих населенных пунктов в различных регионах Российской Федерации
	google_visualizations_chart	Построение диаграмм для сравнения количества районов, городов, прочих населенных пунктов в различных регионах Российской Федерации с использованием Google Visualization
	twitter	Работа с Twitter
	wikipedia	Вывод на веб-страницу новостей культуры, получаемых с RSS-канала "Яндекс.Новости: Культура", возможность выделения термина и получение информации о нем через Википедию

Предметный указатель

Я