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

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

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 возвращенный 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>
```

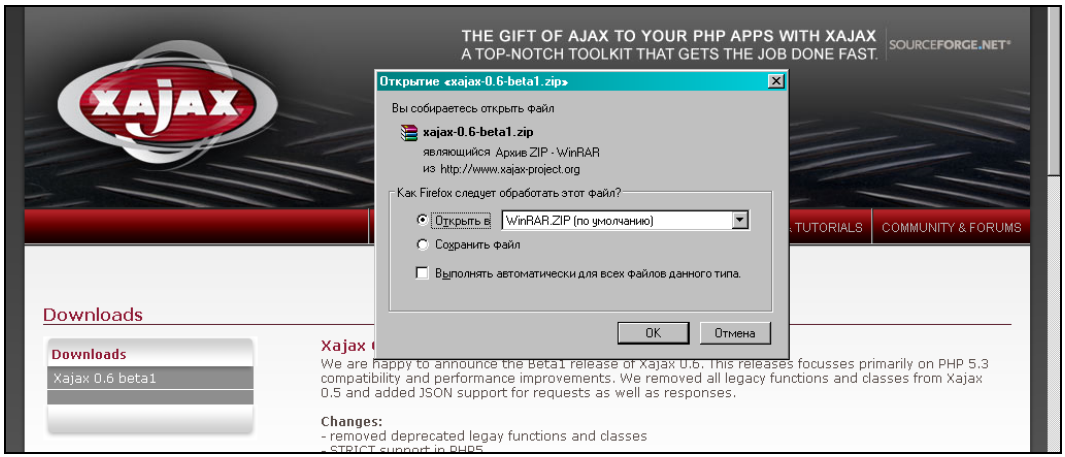

Рис. 1.1. Страница скачивания библиотеки xAjax

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;

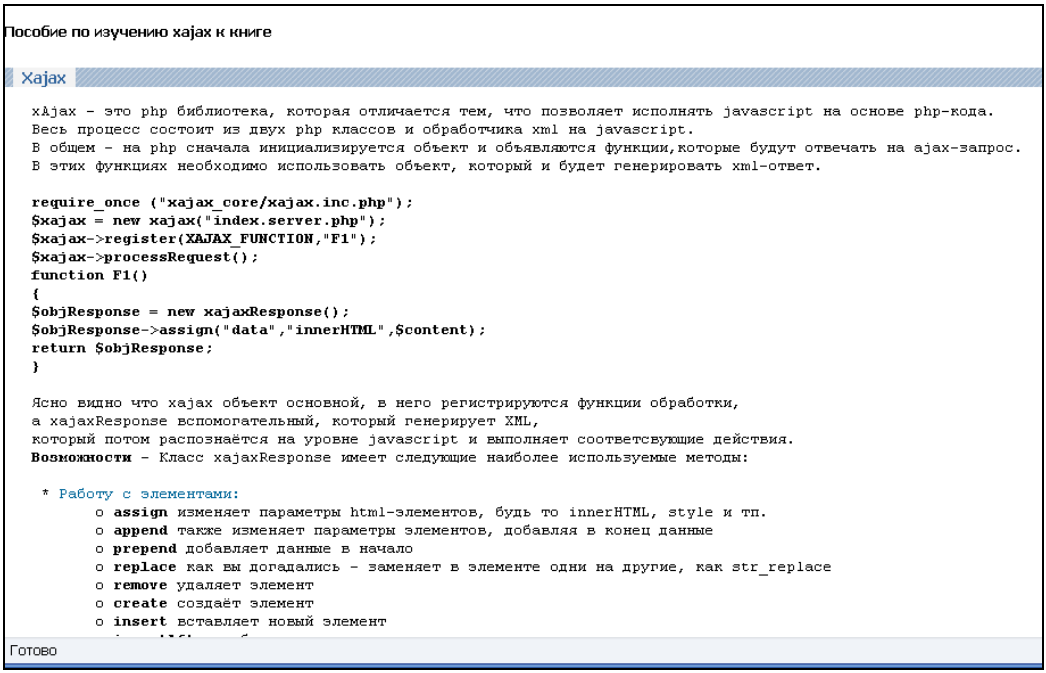

Рис. 1.2. Страница сайта — тренировочного стенда

- 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.

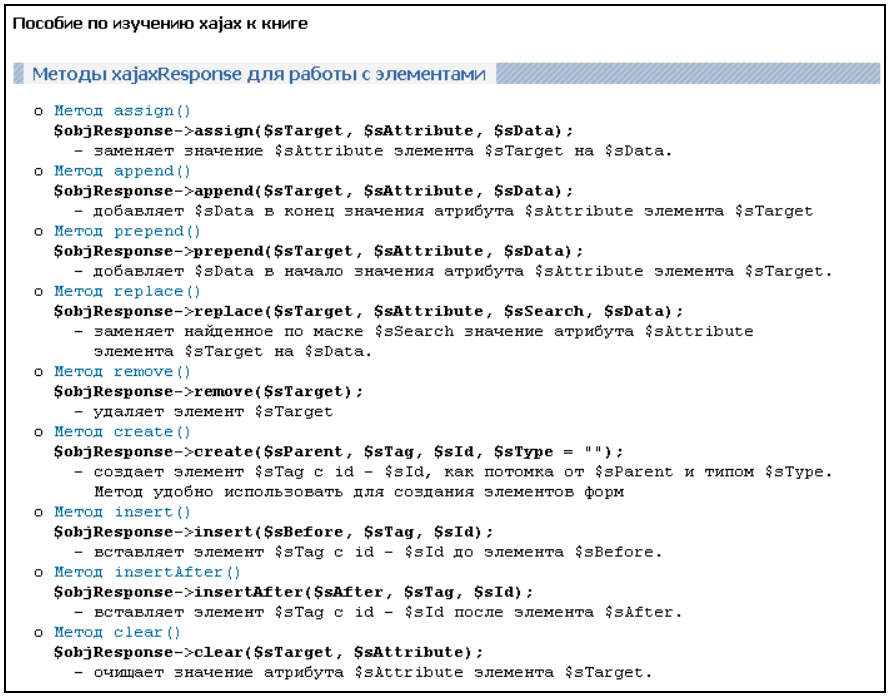

Рис. 1.3. Выбор методов для работы с элементами

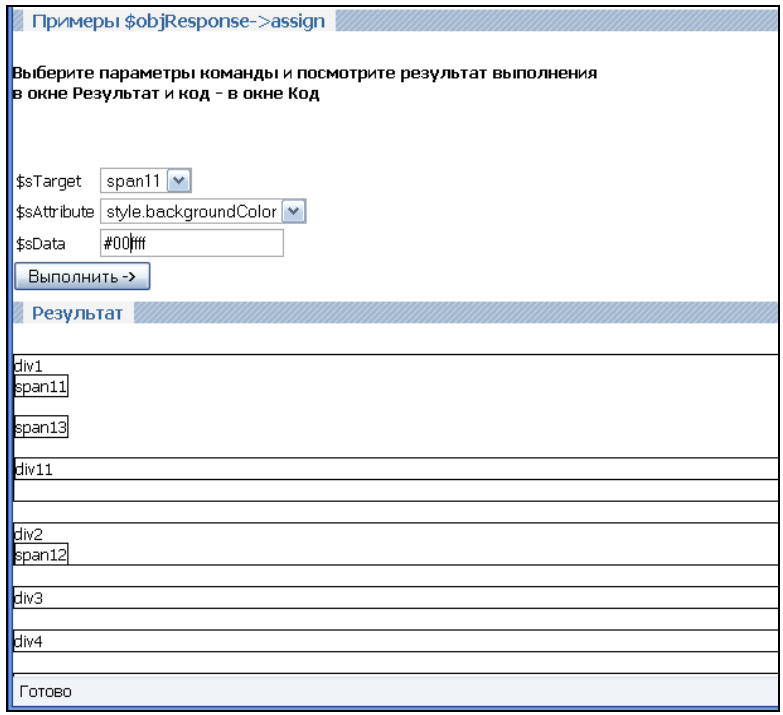

Рис. 1.4. Выбор параметров функции `assign()`

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`.

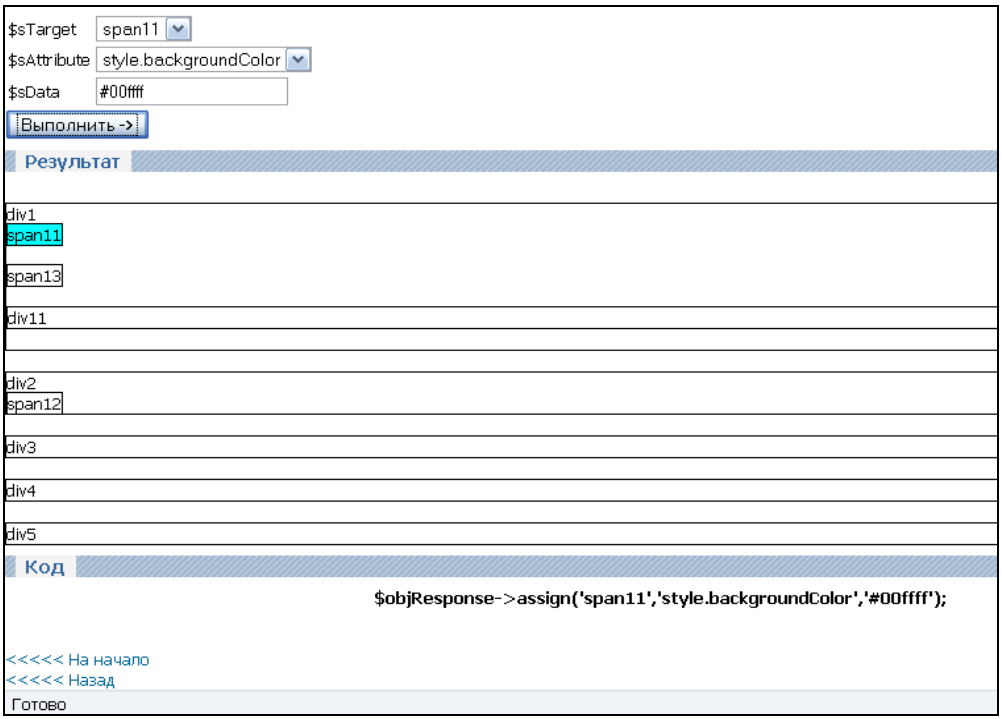

Рис. 1.5. Вид страницы после выполнения функции `assign()`

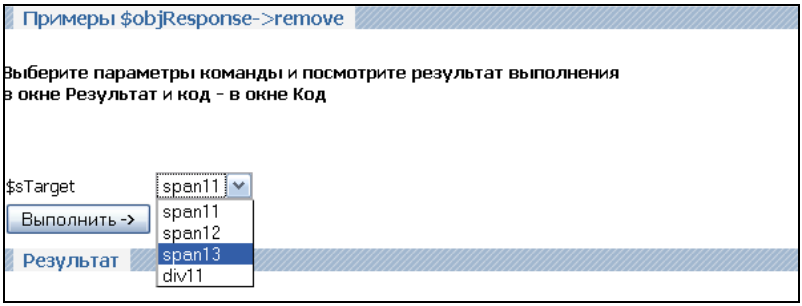

Рис. 1.6. Выбор параметров функции `remove()`

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