АНДРЕЙ ОВЧАРЕНКО

Ajax

НА ПРИМЕРАХ

ОСНОВЫ JavaScript, HTML, CSS И DOM

ОБМЕН ДАННЫМИ С СЕРВЕРОМ В ФОРМАТЕ XML И JSON

РАБОТА С APACHE, PHP И MySQL

ЗАЩИТА ПРИЛОЖЕНИЙ, АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ

СООТВЕТСТВИЕ СТАНДАРТАМ И КРОССБРАУЗЕРНОСТЬ

+CD

bhv®

УДК 681.3.068
ББК 32.973.26-018.1
О-35

Овчаренко А. В.

О-35 Ajax на примерах. — СПб.: БХВ-Петербург, 2009. — 432 с.: ил. + CD-ROM

ISBN 978-5-9775-0299-3

На практических примерах рассмотрены эффективные приемы разработки динамических Web-приложений, построенных по технологии Ajax. Каждая глава посвящена разработке законченного компонента пользовательского интерфейса Web-приложения. Даны необходимые для быстрого старта сведения по HTML и CSS, XML и DOM Level 1, PHP и MySQL, а также примеры совместного их применения. Большое внимание уделено программированию на языке JavaScript и асинхронному обмену данными между клиентом и сервером при помощи объекта XMLHttpRequest в формате XML и JSON. Подробно описан процесс разработки компонентов пользовательского интерфейса: "Аккордеон", панель с закладками, слайд-шоу, выпадающее меню, плавающие окна и др. Рассмотрены вопросы доступа к базам данных MySQL, управления учетными записями пользователей, защиты данных, аутентификации и авторизации, кроссбраузерности разрабатываемых приложений и др. На прилагаемом к книге компакт-диске содержится полный программный код компонентов и приложений, рассмотренных в книге.

Для Web-программистов

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

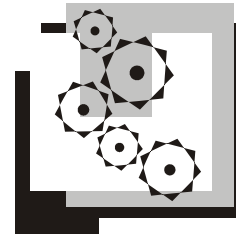
Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.09.08.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 34,83.
Тираж 2500 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

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

ISBN 978-5-9775-0299-3

Оглавление



Глава 2

Использование объекта *XMLHttpRequest* в Ajax-приложениях

В этой главе будут подробно рассмотрены все вопросы, связанные с использованием объекта `XMLHttpRequest`, который является ключевым элементом в инфраструктуре Ajax-приложения. При помощи объекта `XMLHttpRequest` можно:

- ❒ отправить http-запрос к Web-серверу;
- ❒ получить ответ с Web-сервера в виде текста или XML-документа.

Отправка запроса на Web-сервер и получение ответа с Web-сервера посредством объекта `XMLHttpRequest` выполняется в фоновом режиме, незаметно для пользователя или, как говорят, *асинхронно*. Под асинхронностью в данном случае понимается то, что пользователь продолжает работать с Web-браузером, пока запрос путешествует по сети и возвращается обратно в виде ответа Web-сервера. Это позволяет создавать Web-приложения, которые по комфортности работы для пользователя ничем не уступают настольным (desktop) приложениям с графическим пользовательским интерфейсом. Такой уровень комфортности для пользователей обеспечивает совместное использование объекта `XMLHttpRequest` и Динамического HTML (DHTML).

Объект `XMLHttpRequest` в данной главе будет разобран достаточно полно, так что у вас не останется больше вопросов к особенностям его работы. Кроме того, мы разработаем очень полезную функцию `sendRequest()`, которая существенно упростит работу с объектом `XMLHttpRequest`. Эта функция послужит основой для создания Ajax-компонентов во всех последующих главах книги.

Работа с объектом `XMLHttpRequest` напрямую, без вспомогательных функций или библиотек, не совсем удобна для разработчиков Web-приложений. Это связано с тем, что создатели объекта `XMLHttpRequest` и не предполагали, что их детище станет основным звеном инфраструктуры Ajax. В некоторых новых версиях Web-браузеров объект `XMLHttpRequest` частично усовершенствован. Но использовать его новые возможности, например доступ к объекту `XMLHttpRequest` в теле функции-обработчика через неявный параметр `this`,

еще не пришло время. При разработке Web-приложений мы должны стараться охватить как можно больший круг пользователей, у которых могут быть установлены самые различные версии Web-браузеров. Кроме того, даже усовершенствованный объект `XMLHttpRequest` все равно не обеспечивает удобный интерфейс для работы Web-программистов.

В чем же заключаются неудобства для разработчика прикладного программного обеспечения при работе с объектом `XMLHttpRequest`? Трудности начинаются с того, что в некоторых Web-браузерах используется ActiveX-объект `Msxml2.XMLHTTP` или `Microsoft.XMLHTTP`, в то время как другая часть Web-браузеров использует встроенный объект `window.XMLHttpRequest`. Эти два объекта различаются в деталях работы. Например, ActiveX-объект `Msxml2.XMLHTTP` или `Microsoft.XMLHTTP` вызывает функцию-обработчик ответа Web-сервера и для асинхронных, и для синхронных запросов, в то время как объект `window.XMLHttpRequest` вызывает функцию-обработчик ответа Web-сервера только для асинхронных запросов. Проблемой при работе с объектом `XMLHttpRequest` является невозможность без использования *замыкания* (closure) передать параметры функции-обработчику ответа Web-сервера. Подробно о замыканиях и функциях-обработчиках ответа Web-сервера мы поговорим далее в этой главе. Абсолютно все перечисленные проблемы найдут исчерпывающее решение, и мы общими усилиями создадим функцию `sendRequest()`, которая скроет от Web-разработчика все неудобства работы с объектом `XMLHttpRequest`.

Разрабатываемая в этой главе функция `sendRequest()` использует достаточно сложные для начинающих конструкции языка JavaScript. Если это ваше первое знакомство с JavaScript — можно не добиваться полного понимания всех тонкостей работы программного кода. Вам важно получить общее представление о принципах работы с объектом `XMLHttpRequest`, а код функции `sendRequest()` можно скопировать с прилагаемого к книге компакт-диска. Когда же вы освоите JavaScript в полном объеме, то сможете вернуться к более подробному изучению программного кода функции `sendRequest()`.

Текст примеров данной главы вы можете найти в папке \bhvajax\xmlhttprequest, а функцию `sendRequest()` — в файле \bhvajax\bhv\util.js на прилагаемом к книге компакт-диске.

2.1. Варианты использования объекта *XMLHttpRequest* при взаимодействии Web-браузера с Web-сервером

Процесс отправки запроса и получения ответа при помощи объекта `XMLHttpRequest` происходит без перезагрузки текущего HTML-документа,

загруженного в Web-браузер. Как же можно использовать возможности объекта `XMLHttpRequest`? Все разнообразие взаимодействия Web-браузера с Web-сервером в Ajax-приложениях при помощи объекта `XMLHttpRequest` сводится к трем основным вариантам:

- ответ Web-сервера содержит фрагмент HTML-документа;
- ответ Web-сервера содержит XML-документ;
- ответ Web-сервера содержит фрагмент программы JavaScript.

Гораздо реже используют запрос с Web-сервера файлов, содержащих определение каскадных таблиц стилей CSS.

Перечисленные варианты использования больше касаются технической стороны взаимодействия Web-браузера с Web-сервером. С точки зрения логики использования перечисленных технических возможностей, вариантов использования будет гораздо больше. Например, XML-документ может быть использован как источник данных, может быть подвергнут XSLT-преобразованию в HTML-документ или использоваться для работы с Web-сервисами. Фрагмент программы JavaScript может представлять собой код статического модуля JavaScript (js-файла), загружаемого по требованию в Web-браузер, или динамически генерируемый Web-сервером фрагмент программы JavaScript, подробности которого будут зависеть от параметров запроса. Но чаще всего в качестве фрагмента программы JavaScript загружается объект, записанный в JavaScript Object Notation (JSON). Давайте познакомимся со всеми перечисленными возможностями подробнее.

2.1.1. Использование объекта *XMLHttpRequest* для загрузки фрагмента HTML-документа

Объект `XMLHttpRequest` может запросить с Web-сервера фрагмент HTML-документа. Ответ Web-сервера будет содержаться в свойстве `responseText` объекта `XMLHttpRequest`, которое имеет тип `String`. Для обновления HTML-документа данными, полученными с Web-сервера, значение свойства `responseText` необходимо присвоить свойству `innerHTML` существующего или вновь создаваемого HTML-элемента. Web-браузер в соответствии с новым значением свойства `innerHTML` HTML-элемента обновляет объектную модель документа (DOM) и внешний вид документа. Это обычный путь использования объекта `XMLHttpRequest`, если ответ Web-сервера содержит фрагмент HTML-документа.

Рассмотрим простой пример. Пусть в условном приложении интернет-магазина Web-сервер в качестве ответа на запрос `XMLHttpRequest` формирует

фрагмент HTML-документа с разметкой переключателей для выбора цвета заказываемого товара с учетом текущего остатка на складе:

```
<input type="radio" name="color" value="13" checked>оливковый<br>
<input type="radio" name="color" value="19">хаки<br>
<input type="radio" name="color" value="21">индиго<br>
```

Полученный с Web-сервера фрагмент HTML-документа присваивается свойству `innerHTML` HTML-элемента `FIELDSET` с атрибутом `id="selectColor"`:

```
var xhr = new (window.XMLHttpRequest || ActiveXObject)("Msxml2.XMLHTTP");
xhr.open("GET", "ajax_response.php", false);
xhr.send(null);
var selectColorVar = document.getElementById("selectColor");
selectColorVar.innerHTML = xhr.responseText;
```

Свойство `innerHTML` представляет собой часть HTML-документа между открывающим и закрывающим тегом элемента. После выполнения приведенного кода, элемент `FIELDSET` будет содержать три элемента `<input type="radio">`, как если бы в тексте HTML-документа содержалась следующая разметка:

```
<fieldset id="selectColor">
  <input type="radio" name="color" value="13">оливковый<br>
  <input type="radio" name="color" value="19" checked>хаки<br>
  <input type="radio" name="color" value="21">индиго<br>
</fieldset>
```

Такой вариант взаимодействия Web-браузера с Web-сервером является переходным от классического Web-приложения к Ajax-приложению. Его использование позволяет с минимальными изменениями в существующем коде адаптировать классическое Web-приложение к технологии Ajax. Шагом вперед является то, что текущая страница не перезагружается и работа пользователя становится более комфортной, без вынужденных перерывов, связанных с ожиданием ответа Web-сервера и полной перерисовкой страницы в окне Web-браузера. Но роль Web-браузера по-прежнему остается статичной. Web-браузер только отображает полученный с Web-сервера готовый фрагмент HTML-документа.

Использовать загрузку фрагмента HTML-документа в качестве ответа Web-сервера можно рекомендовать в тех случаях, когда традиционно использовались HTML-элементы `IFRAME`. Однако для более серьезных приложений, интенсивно работающих с данными, такой подход приведет к усложнению разработки за счет появления жесткой зависимости серверного и клиентского кода. Загружаемый с Web-сервера фрагмент HTML-документа должен быть сформирован с таким расчетом, чтобы точно вписаться в текущий HTML-

документ. Следовательно, программный код серверного и клиентского приложений должен быть тщательно согласован, и разработчик серверного кода должен хорошо знать HTML. Этот недостаток первого из трех основных вариантов использования асинхронных запросов к Web-серверу при помощи объекта `XMLHttpRequest` в объектно-ориентированном анализе называют *жесткой связанностью*. Недостатки жесткой связанности особенно ярко проявляются при разработке проектов, в которых участвует несколько разработчиков разного направления, как это часто бывает при разработке Web-приложений.

ЗАМЕЧАНИЕ

Вас может заинтриговать фрагмент кода, создающего объект `XMLHttpRequest`:

```
var xhr = new (window.XMLHttpRequest || ActiveXObject)
("Msxml2.XMLHTTP");
```

Этот оператор следует читать так. Выражение в скобках вернет первое значение, отличное от `null`:

```
(window.XMLHttpRequest || ActiveXObject)
```

Это значение будет использовано для создания объекта оператором `new`. Фактически будет выполнен один из вариантов программного кода, в зависимости от модели Web-браузера:

```
var xhr = new window.XMLHttpRequest("Msxml2.XMLHTTP");
```

или

```
var xhr = new ActiveXObject("Msxml2.XMLHTTP");
```

В первом варианте параметр функции-конструктора будет проигнорирован. Второй вариант будет выполнен только при отсутствии у Web-браузера объекта `window.XMLHttpRequest`.

2.1.2. Использование объекта *XMLHttpRequest* для загрузки XML-документа

Объект `XMLHttpRequest` может запросить с Web-сервера XML-документ. Для того чтобы упростить работу с XML-документом на стороне Web-браузера, объект `XMLHttpRequest` имеет, кроме свойства `responseText`, еще и свойство `responseXML`. В отличие от свойства `responseText`, которое имеет тип `String`, свойство `responseXML` имеет тип `Document` (иногда его называют `DOMDocument`). Это свойство реализует спецификацию *DOM Level 1* (и выше). Подробно об этом мы поговорим в *главе 4*. А сейчас приведем фрагмент кода, для того чтобы вы постепенно готовились к восприятию материала *главы 4*:

```
var xhr = new (window.XMLHttpRequest || ActiveXObject)("Msxml2.XMLHTTP");
xhr.open("GET", "ajax_response.xml", false);
```

```
xhr.send(null);
var doc = xhr.responseXML;
var products = doc.getElementsByTagName("product")
var id = products[0].getElementsByTagName("id")[0].firstChild.data;
```

Как вы заметили, работа с объектом типа `Document` (`DOMDocument`) немного напоминает работу с предопределенным объектом `document` Web-браузера. И это не удивительно, потому что объектная модель HTML-документа (DOM), которую реализует предопределенный объект Web-браузера `document`, также определена в спецификации DOM Level 1. Именно в этой спецификации определены часто используемые свойства, коллекции и методы предопределенного объекта `document` Web-браузера: `documentElement`, `forms`, `images`, `cookie`, `getElementById()`, `getElementsByTagName()`, `write()`, `writeln()` и др.

На основании разбора полученного с Web-сервера XML-документа обновляются модель данных на стороне Web-браузера, объектная модель документа (DOM) и внешний вид документа. Этот вариант использования объекта `XMLHttpRequest` наиболее перспективный, т. к. XML является своеобразным общим знаменателем, язык которого понимают разработчики разных направлений. Формат XML-документов может быть согласован на самых ранних этапах разработки приложения, после чего каждый разработчик (группа разработчиков) продолжает работу над своей частью приложения, опираясь на формат согласованного XML-документа. Это существенно ускоряет разработку приложения и устраняет *жесткую связанность* между серверным и клиентским кодом.

На работе с XML построена технология преобразования XML-документов в HTML-документы, которая называется XSLT. В большинстве случаев такое преобразование выполняется на Web-сервере. Но вполне возможно, что у вас будет необходимость реализовать XSLT-преобразование средствами Web-браузера. Для использования технологии XSLT на стороне Web-браузера, вам пришлось бы проделать слишком много работы, если бы такая работа не была уже проделана разработчиками специализированных библиотек JavaScript. Такие библиотеки обеспечивают поддержку кроссбраузерной работы с XML-документами и собственно XSLT-преобразование XML-документа в HTML-документ.

Другой технологией, построенной на использовании XML-документов, является использование Web-сервисов. Возможно, вы уже сейчас планируете использовать Ajax-запросы для обращения к Web-сервисам, но тут вас может ожидать одна проблема. В Web-браузере действует модель защиты, называемая "песочницей". Такое образное название достаточно точно характеризует

суть происходящего. Если пользователь загрузил основной HTML-документ с вашего домена, то все его попытки обратиться при помощи объекта `XMLHttpRequest` к интернет-ресурсам за пределами вашего домена (за пределами "песочницы") вызовут ошибку запрета доступа. Это делает практически невозможным непосредственный доступ при помощи объекта `XMLHttpRequest` к Web-сервисам, расположенным за пределами вашего домена. Чтобы преодолеть такое ограничение, доступ к Web-сервисам на чужих доменах необходимо реализовывать, используя свой Web-сервер как ретранслятор запросов. Разумеется, для этого следует использовать готовые специализированные серверные и клиентские библиотеки, так что для прикладного Web-программиста вся работа по ретрансляции запросов и ответов к Web-сервисам останется за кадром.

> ***ЗАМЕЧАНИЕ***
>
> Следует учитывать, что модель защиты "песочница" не распространяется на загружаемые изображения (HTML-элементы `IMG`) и скрипты (HTML-элементы `SCRIPT`), если только не задать такое ограничение явно в опциях Web-браузера. Загрузка скриптов с чужих доменов является потенциально опасной операцией, о чем должен всегда помнить Web-разработчик.

2.1.3. Использование объекта *XMLHttpRequest* для загрузки фрагментов программы JavaScript

Объект `XMLHttpRequest` может запросить с Web-сервера фрагмент программы JavaScript. Полученный с сервера фрагмент программы JavaScript выполняется встроенной функцией JavaScript `eval()`. Эта функция выполняет фрагмент программы JavaScript, заданный в качестве параметра типа `String`, и возвращает значение последнего выполненного оператора, которое можно присвоить некоторой переменной:

```
var xhr = new (window.XMLHttpRequest || ActiveXObject)("Msxml2.XMLHTTP");
xhr.open("GET", "ajax_script.js", false);
xhr.send(null);
var returnValue = eval(xhr.responseText);
```

Использовать такую возможность можно по-разному. Например, фрагмент программы JavaScript может динамически генерироваться Web-сервером, исходя их параметров запроса Web-браузера. Основным недостатком такого способа является необходимость подробнейшего согласования серверного и клиентского кода. При этом связанность серверного и клиентского кодов получается даже более жесткой, чем при загрузке фрагмента HTML-документа (т. е. первого варианта использования объекта `XMLHttpRequest`).

Поэтому чаще применяется подход, при котором функцию `eval()` используют для создания полученного с Web-сервера *JSON-объекта*, т. е. объекта, записанного в *JavaScript Object Notation*. JSON-объект является одновременно и облегченной альтернативой XML, и фрагментом исполняемого кода на языке JavaScript. Например, XML-документ:

```
<colors>
  <color value='13'>оливковый</color>
  <color value='19'>хаки</color>
  <color value='21'>индиго</color>
</colors>
```

можно записать как JSON-объект:

```
[
  {value: "13", color: "оливковый"},
  {value: "19", color: "хаки"},
  {value: '21', color: "индиго"}
]
```

или даже так:

```
{
  "13": "оливковый",
  "19": "хаки",
  "21": "индиго"
}
```

Такая запись может оказаться существенно проще, чем XML-документ. Кроме того, JSON-объект не требует XML-разбора. Вызов функции `eval()` помещает JSON-объект в память интерпретатора JavaScript и возвращает ссылку на этот объект. Для корректной работы функции `eval()`, строка, содержащая JSON-объект, обязательно должна быть взята в круглые скобки:

```
var jsonObject1 = {
  "13": "оливковый",
  "19": "хаки",
  "21": "индиго"
};
var jsonString1 = '{'+
'  "13": "оливковый",'+
'  "19": "хаки",'+
'  "21": "индиго"'+
```

```
'}';
jsonObject2 = eval('(' + jsonString1 + ')');
```

После выполнения функции `eval()` вы можете работать с JSON-объектом при помощи всего спектра возможностей языка программирования JavaScript. Использование JSON-объектов при взаимодействии Web-браузера и Web-сервера считается хорошим приемом. Этот способ по своей логике ближе к использованию XML-документа в качестве ответа Web-сервера, а по формальному признаку является загрузкой фрагмента программы JavaScript. В PHP 5, начиная с версии 5.2.0, реализовано JSON-расширение. Функция `json_encode()` транслирует объект (ассоциативный массив) PHP 5 в строку JSON, а функция `json_decode()` транслирует строку JSON в объект (ассоциативный массив) PHP 5. Очевидно, что JSON-нотация вышла за рамки применения в контексте исключительно JavaScript и превратилась в удобное средство обмена данных в текстовом формате между разнородными системами. Обратите внимание на технологию JSON. Существует мнение, что JSON может реально заменить технологию XML.

Подведем итоги. Взаимодействие Web-браузера с Web-сервером в Ajax-технологии реализуется при помощи объекта `XMLHttpRequest`. Мы рассмотрели три основных варианта взаимодействия Web-браузера и Web-сервера в Ajax-приложениях, когда в качестве ответа Web-браузер получает с Web-сервера:

- фрагмент HTML-документа;
- XML-документ;
- фрагмент программы на языке JavaScript.

Наиболее универсальным и перспективным способом взаимодействия клиента и Web-сервера является использование XML-документов в качестве ответа Web-сервера. Хорошей альтернативой использования XML-документа служит использование JSON-объектов в качестве ответа Web-сервера.

2.2. Основы работы с объектом *XMLHttpRequest*

Объект `XMLHttpRequest` создается при помощи оператора JavaScript `new`. Для каждого запроса к Web-серверу можно создавать новый объект `XMLHttpRequest`. Или можно создать пул объектов `XMLHttpRequest`, чтобы использовать для очередного запроса освободившийся объект из пула. Можно, наконец, обойтись всего одним объектом `XMLHttpRequest`, направляя запросы в очередь и обрабатывая их в порядке поступления одним-единственным экземпляром

объекта XMLHttpRequest. Существуют доводы за и против использования каждой из этих возможностей.

Создание нового объекта для каждого запроса к Web-серверу может привести в некоторых типах Web-браузеров к утечке памяти, потому что объект запроса обычно содержит циклическую ссылку на функцию-обработчик запроса, что затрудняет работу "сборщика мусора". Вариант с использованием пула объектов позволяет ограничить количество создаваемых объектов XMLHttpRequest и облегчает работу "сборщика мусора". Но реализация пула замедляет работу приложения, а самое главное, независимо от количества объектов в пуле, реально, с Web-сервером открывается не более двух соединений (если не менять настройки системы по умолчанию). Поэтому реализация пула объектов XMLHttpRequest на JavaScript применяется сравнительно редко. Реализация пула объектов XMLHttpRequest разобрана в статье автора "Как организовать пул объектов XMLHttpRequest для использования в Ajax-приложениях", которая доступна по URL-адресу **http://www.gotdotnet.ru/LearnDotNet/ASPNET/453076.aspx**.

В связи с ограничением на два активных соединения с Web-сервером идея использовать всего один объект XMLHttpRequest для всех запросов может показаться достаточно обоснованной. Если бы все запросы к Web-серверу были успешными, одного объекта XMLHttpRequest вполне хватило бы для работы Ajax-приложения. Но если хотя бы один запрос зависнет на время, пока не истечет тайм-аут, такой запрос блокирует единственный объект XMLHttpRequest и работу всего приложения.

В наших примерах мы всегда будем использовать новый объект XMLHttpRequest для каждого запроса к Web-серверу, принимая при этом меры, предупреждающие утечку памяти.

После создания объекта XMLHttpRequest оператором new запрос открывается методом open(). Открытие запроса методом open() — это еще не отправка запроса к Web-серверу, которая выполняется методом send(). После открытия запроса можно устанавливать заголовки запроса методом setRequestHeader(). У объекта запроса нет отдельного метода, устанавливающего заголовок Content-Type, который тоже устанавливается методом setRequestHeader(). Самым естественным было бы использовать Content-Type: application/xml (тип text/xml в настоящее время объявлен устаревшим) и разбирать на сервере не параметры запроса, а тело запроса, которое в этом случае должно содержать XML-документ.

Несмотря на то, что объект XMLHttpRequest был разработан как раз для обмена XML-документами, его часто используют для запросов к серверу типа application/x-www-form-urlencoded. Объясняется это тем, что такие запросы более привычны Web-разработчикам, и серверные языки программирования

без дополнительных усилий со стороны Web-программиста разбирают параметры этих запросов, в то время как работа с телом запроса, которое содержит XML-документ, требует, пусть даже самых небольших, дополнительных усилий.

Для корректной работы запросов типа `application/x-www-form-urlencoded` все параметры запроса должны быть правильно закодированы в соответствии со спецификацией MIME-типа `application/x-www-form-urlencoded`. Для этого в JavaScript предусмотрена функция `encodeURIComponent()`. Функция `encodeURIComponent()` отличается от похожей функции `encodeURI()` тем, что дополнительно перекодирует знак `+` (плюс). Функцию `encodeURIComponent()` следует всегда использовать для перекодирования строковых параметров запроса перед их отправкой на сервер с помощью объекта `XMLHttpRequest`, если запрос имеет тип `application/x-www-form-urlencoded`:

```
var xmlHttpRequest = new XMLHttpRequest();
xmlHttpRequest.open("post","test.php",true);
xmlHttpRequest.setRequestHeader("Content-Type",
                          "application/x-www-form-urlencoded");
xmlHttpRequest.send("count=0&what=" + encodeURIComponent(
  "JavaScript & DOM & CSS + XMLHttpRequest == Ajax-технология?"));
```

Если вы отправите на сервер подобный запрос без использования функции `encodeURIComponent()`, смысл запроса будет сильно искажен, потому что строка, передаваемая в качестве параметра `what`, содержит пробелы, специальные знаки и кириллические символы. Использовать функцию `encodeURIComponent()` для отправки на сервер запросов другого типа, в том числе `application/xml`, не следует.

После отправки запроса на сервер методом `send()` объект `XMLHttpRequest` соединяется с Web-сервером и получает от него ответ, доступ к которому можно получить при помощи двух свойств этого объекта — `responseText` и `responseXML`.

Свойство `responseText` объекта `XMLHttpRequest` содержит ответ Web-сервера в виде строки типа `String`. Свойство `responseXML` объекта `XMLHttpRequest` содержит ответ Web-сервера в виде объекта типа `Document` (иногда его называют `DOMDocument`). Некоторые Web-браузеры не загружают ответ Web-сервера в свойство `responseXML`, если в http-заголовках ответа Web-сервера явно не указано, что тело ответа содержит XML-документ. Этот http-заголовок Web-сервер может отправить автоматически, в том случае, если запрашивается файл с расширением xml. Чтобы понять, как это происходит, рассмотрим фрагмент конфигурационного файла mime.types Web-сервера Apache:

```
application/xhtml+xml                    xhtml xht
application/xml                          xml xsl
application/xml-dtd                      dtd
```

Отправляя файл с расширением xml, Web-сервер Apache, в полном соответствии с конфигурационными параметрами, отправляет и соответствующий http-заголовок `application/xml`. Для этого вам не придется конфигурировать Web-сервер самостоятельно, поскольку такая конфигурация предусмотрена по умолчанию. Получив http-заголовок ответа Web-сервера `Content-Type: application/xml`, Web-браузер распознает, что получен XML-документ, и пытается сделать разбор этого XML-документа. Если полученный XML-документ не содержит ошибок, формируется объект типа `Document` (`DOMDocument`), ссылка на который присваивается свойству `responseXML` объекта `XMLHttpRequest`.

Если XML-документ формируется серверным скриптом, например, на языке программирования PHP 5, заголовок ответа необходимо отправить явно, поскольку запрашиваемый ресурс будет иметь расширение php, либо другое (из соображений безопасности), но не xml:

```
<?php
  #head("Content-Type: text/xml");
  head("Content-Type: application/xml");
?>
```

Если такой заголовок не отправить явно, для некоторых типов Web-браузеров свойство `responseXML` объекта `XMLHttpRequest` будет пустым. Еще одной возможностью сообщить Web-браузеру, что текст ответа содержит XML-документ, является явное объявление XML-документа в первой строке текста этого документа:

```
<?xml version="1.0" encoding="windows-1251"?>
```

Но полагаться только на объявление XML-документа не стоит. Явное задание http-заголовка `Content-Type: application/xml` является единственно надежным способом заставить Web-браузер сформировать в свойстве `responseXML` непустой объект `Document`, содержащий XML-документ.

В приведенном фрагменте кода, объявление XML-документа содержит атрибут `encoding`, который задает кодировку XML-документа. Кроме этого, кодировку можно задать в конфигурации Web-сервера *(см. главу 1)*, конфигурационном файле PHP 5 php.ini и в серверном скрипте PHP 5:

```
<?php
  #head("Content-Type: text/xml");
  head("Content-Type: application/xml; charset='windows1251'");
?>
```

В отличие от явного задания типа `application/xml`, явно задавать кодировку `charset` в скрипте PHP 5 не рекомендуется. Все, что связано с кодировками,

лучше выносить в конфигурационные параметры Web-сервера и конфигурационный файл PHP 5 php.ini. Это позволит легко адаптировать приложения для использования с различными кодировками. Во всяком случае, явное задание кодировки `charset` в PHP-скрипте нужно использовать только в том случае, если все остальные способы испробованы и не принесли положительного результата.

Вопрос с кириллическими кодировками может стать настоящей проблемой для начинающего Web-программиста. Когда Web-браузер отображает текст в неверной кодировке, заочно никто не поможет вам преодолеть эту проблему, особенно если речь идет о чужом хостинге, на котором вы размещаете свои приложения, не имея доступа к конфигурации Web-сервера. Если вы следуете моим рекомендациям по конфигурированию Web-сервера и имеете полный контроль над конфигурацией, то должны избежать проблем с кодировкой. Обеспечение правильной работы с кириллическими кодировками в Ajax-технологии относится, в большей степени, не к сфере программирования, а к сфере конфигурирования серверной среды. Если у вас что-то не ладится с кодировками и вы, как говорится, "застряли", я настоятельно рекомендую отложить на некоторое время вопрос с кодировками и идти вперед. Пока вы еще не освоились с конфигурированием (а нужно ли это Web-программисту вообще?), можно изучать Ajax-технологию, используя только символы латиницы из диапазона кодов символов от 0 до 127, которые без проблем отображаются при любой конфигурации серверной среды.

Я боюсь больше испытывать ваше терпение и перехожу к реальному примеру. При помощи объекта `XMLHttpRequest` можно отправлять запросы к Web-серверу в *синхронном* и *асинхронном* режимах. В большинстве случаев используют асинхронные запросы. Тем не менее, бывают случаи (их не так много), когда без синхронных запросов не обойтись, или асинхронные решения получаются слишком сложными. Это касается таких задач, как загрузка программного кода JavaScript, и некоторые другие случаи, когда для продолжения работы текущего скрипта необходимо, чтобы запрос наверняка отработал.

В большинстве случаев используются асинхронные запросы. Положительным моментом при использовании асинхронных запросов является то, что выполнение текущего скрипта не приостанавливается и интерпретатор JavaScript Web-браузера не блокируется. Платой за использование асинхронных запросов является некоторое усложнение программирования. Мы начнем знакомство с объектом `XMLHttpRequest` на самом простом примере. В данном конкретном примере следовало бы использовать асинхронный запрос, но для этого у нас еще нет достаточных знаний. Первый наш запрос будет синхронным только потому, что такой запрос легче реализовать. Кроме

того, познакомившись с синхронными запросами, вы лучше оцените преимущества асинхронных запросов.

Давайте загрузим при помощи синхронного запроса `XMLHttpRequest` фрагмент HTML-документа и на основании полученного фрагмента обновим текущий документ в Web-браузере. Это соответствует первому варианту взаимодействия Web-браузера с Web-сервером в технологии Ajax *(см. разд. 2.1)*. Вернемся к нашему примеру из *разд. 2.1* и реализуем его в полном объеме. Для начала, создадим файл response.txt (см. компакт-диск) и поместим в него фрагмент HTML-документа, который будет загружаться в Web-браузер при помощи объекта `XMLHttpRequest`.

Мы поместили фрагмент HTML-документа в файл с расширением txt, а не html, подчеркивая тем самым, что этот фрагмент не является самостоятельным HTML-документом. В реальных приложениях формирование фрагментов HTML-документов будет выполняться с использованием серверного программирования, например при помощи PHP 5.

Основной документ testxmlhttp.html (см. компакт-диск) содержит HTML-элемент `FIELDSET`, в который будет помещаться подготовленный на сервере фрагмент. Напоминаю, что все файлы мы договорились сохранять в кодировке Windows-1251, и для этого специально сконфигурировали Web-сервер Apache *(см. главу 1)*. Для сохранения файла в кодировке Windows-1251, необходимо задать специальные опции текстового редактора или использовать пункт меню редактора **Сохранить как** (Save as), где выбрать кодировку Windows-1251. В некоторых редакторах, например Блокноте, для сохранения в кодировке Windows-1251 необходимо в появившемся диалоговом окне **Сохранить как** (Save as) выбрать кодировку ANSI.

В разметке HTML-документа testxmlhttp.html присутствуют HTML-элементы `FIELDSET` и `INPUT type="button"`:

```
<fieldset id="selectColor" style="width:250"></fieldset>
<input type="button"
       onclick="sendColorRequest();" value="Отправить запрос">
```

Для HTML-элемента `FIELDSET` задан атрибут `id="selectColor"`. Атрибут `id`, который называют идентификатором, используется для поиска элемента в объектной модели документа (DOM):

```
var selectColor = document.getElementById("selectColor");
```

Элемент `INPUT type="button"` (кнопка) служит для вызова функции, которая загружает файл response.txt с Web-сервера. Для этого в элементе задан атрибут `onclick="sendColorRequest();"`.

Теперь рассмотрим, как отправляется запрос на Web-сервер. Сначала необходимо создать с учетом кроссбраузерности новый объект запроса:

```
var xmlHttpRequest;
if (window.XMLHttpRequest)
  xmlHttpRequest = new window.XMLHttpRequest();
else
  xmlHttpRequest = new ActiveXObject("Msxml2.XMLHTTP");
```

После того как объект запроса создан, запрос необходимо открыть:

```
xmlHttpRequest.open("get","response.txt",false);
```

Первый параметр запроса задает http-метод запроса, который может принимать различные значения, но чаще всего это будет запрос типа `GET` или `POST`. Второй параметр запроса задает URL-адрес загружаемого ресурса. В данном примере задан относительный путь к ресурсу, поэтому файл response.txt будет загружен из той же папки, из которой загружен основной документ testxmlhttp.html. Третий параметр, который имеет логический тип `false`, определяет, будет ли запрос асинхронным. Значение `false` означает, что запрос будет синхронным (не асинхронным). При выполнении синхронного запроса интерпретатор JavaScript приостанавливает выполнение текущего скрипта до полной загрузки запрашиваемого с Web-сервера документа. Как правило, стандартные HTML-элементы ввода `INPUT`, `TEXTAREA`, `SELECT`, `BUTTON` и `A` (anchor) на это время не блокируется. Но элементы интерфейса, которые разработаны с использованием технологий DHTML и Ajax, будут блокированы вместе с интерпретатором JavaScript. Если соединение с сервером медленное или запросы к серверу интенсивные, синхронные запросы могут очень серьезно повлиять на производительность приложения.

```
xmlHttpRequest.send(null);
var selectColor = document.getElementById("selectColor");
selectColor.innerHTML = xmlHttpRequest.responseText;
```

В приведенном фрагменте при синхронном (не асинхронном) запросе после вызова метода `send()` и до выполнения следующего оператора может пройти достаточно большой промежуток времени. На рис. 2.1 приведена диаграмма синхронного запроса к Web-серверу при помощи объекта `XMLHttpRequest`. На диаграмме видно, что все то время, пока Web-браузер отправляет запрос на сервер, сервер обрабатывает запрос и возвращает ответ Web-браузеру, интерпретатор JavaScript находится в состоянии ожидания. Давайте подумаем, не слишком ли большая роскошь блокировать работу приложения на время работы запроса? Для того чтобы сделать работу приложения более производительной, следует использовать преимущественно асинхронные запросы. Но об этом мы поговорим в следующем разделе.

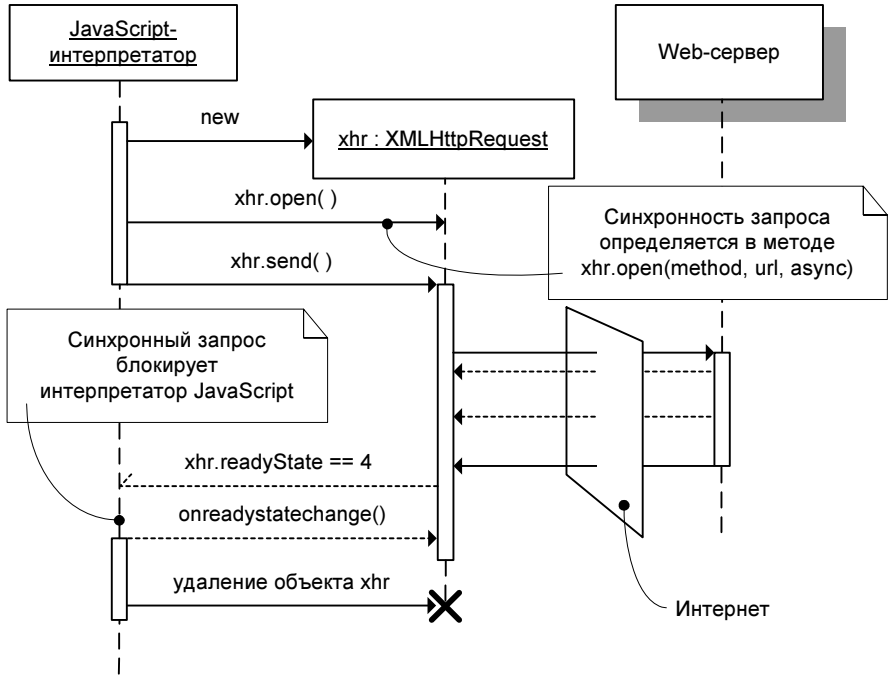


Рис. 2.1. Синхронный запрос при помощи объекта `XMLHttpRequest`

Файлы response.txt и testxmlhttp.html вы можете найти на прилагаемом к книге компакт-диске в папке \bhvajax\xmlhttprequest. Если вы установили Web-сервер так, как это было предложено в *главе 1*, то можете протестировать свою и мою работу, введя в строке адреса Web-браузера:

http://localhost:8080/bhvajax/xmlhttprequest/testxmlhttp.html

После загрузки документа в Web-браузер нажмите кнопку **Отправить запрос** — и можно немного похвалить себя, если все работает правильно. Или заняться отладкой — последнее даже интересней.

Теперь попробуйте раскрыть Web-браузером файл testxmlhttp.html из локальной файловой системы без использования Web-сервера, например, щелкнув по значку файла кнопкой мыши в Проводнике. Похоже, что все работает, но не торопитесь делать выводы. Нажмите кнопку **Отправить запрос**, и вы увидите, что текст, загруженный из файла response.txt в кириллической кодировке Windows-1251, отображается неправильно. А почему? Ведь оба файла мы сохранили в одинаковой кодировке Windows-1251. Тут вам пора познакомиться с одной особенностью работы объекта `XMLHttpRequest`. Этот объект независимо от используемой текущим HTML-документом кодировки

по умолчанию ожидает, что ответ пришел в кодировке UTF-8. Когда файл запрашивался с Web-сервера, который мы сконфигурировали для использования по умолчанию кодировки Windows-1251 *(см. главу 1)*, ответ Web-сервера содержал специальные http-заголовки:

```
Date: Thu, 03 Apr 2008 05:11:26 GMT
Server: Apache/2.2.8 PHP/5.2.5
Last-Modified: Sat, 29 Mar 2008 04:37:27 GMT
Accept-Ranges: bytes
Content-Length: 185
Content-Type: text/plain; charset=windows-1251

200 OK
```

Среди заголовков присутствует и заголовок `Content-Type`, который сообщает объекту `XMLHttpRequest`, что пришел ответ в кодировке Windows-1251, и именно поэтому текст отображается Web-браузером правильно. Когда файл открывается из локальной файловой системы — список заголовков пуст, их просто некому отправить. Поэтому объект `XMLHttpRequest` предполагает, что ответ пришел в кодировке по умолчанию UTF-8. В результате, над текстом из файла response.txt проводится лишняя операция преобразования в кодировку Windows-1251, хотя текст изначально уже имел кодировку Windows-1251.

Без использования Web-сервера можно отобразить загружаемый документ правильно, сохранив файл response.txt в кодировке UTF-8. Для этого необходимо задать специальные опции текстового редактора или использовать пункт меню редактора **Сохранить как** (Save as), где выбрать кодировку UTF-8. Вообще, использовать кодировку UTF-8 в Ajax-приложениях проще, чем любую другую. Если вы решите использовать кодировку UTF-8 для своих Ajax-приложений, первый вопрос, который вам необходимо проработать — как будет обеспечиваться взаимодействие с базой данных. В настоящее время все ведущие системы управления базами данных поддерживают работу с кодировками Unicode и UTF-8. Вы должны убедиться, что работа с базой данных при использовании кодировки UTF-8 в ваших приложениях не вызовет проблем с конкретной системой управления базами данных.

Итак, мы создали простейшую программу, которая направила синхронный запрос `XMLHttpRequest` к Web-серверу и получила ответ в виде фрагмента HTML-документа. Текущий HTML-документ был обновлен в соответствии данными, полученными с Web-сервера. Для этого свойству `innerHTML` HTML-элемента `FIELDSET` было присвоено значение, полученное с Web-сервера.

2.3. Функция-обработчик события *onreadystatechange* объекта *XMLHttpRequest*

Асинхронные запросы к серверу — это те запросы, которые вы будете в основном применять в своих Ajax-приложениях. При работе с асинхронными запросами используется событийно-ориентированная модель программирования. Объект `XMLHttpRequest` отправляет асинхронный запрос на сервер методом `send()` и сразу же, не дожидаясь получения ответа Web-сервера, возвращает управление интерпретатору JavaScript. В процессе получения ответа с Web-сервера, генерируется несколько событий `onreadystatechange`, которые сообщают, что состояние запроса изменилось. В случае успешного прохождения запроса, последнее событие `onreadystatechange` генерируется после полной загрузки ответа Web-сервера в Web-браузер.

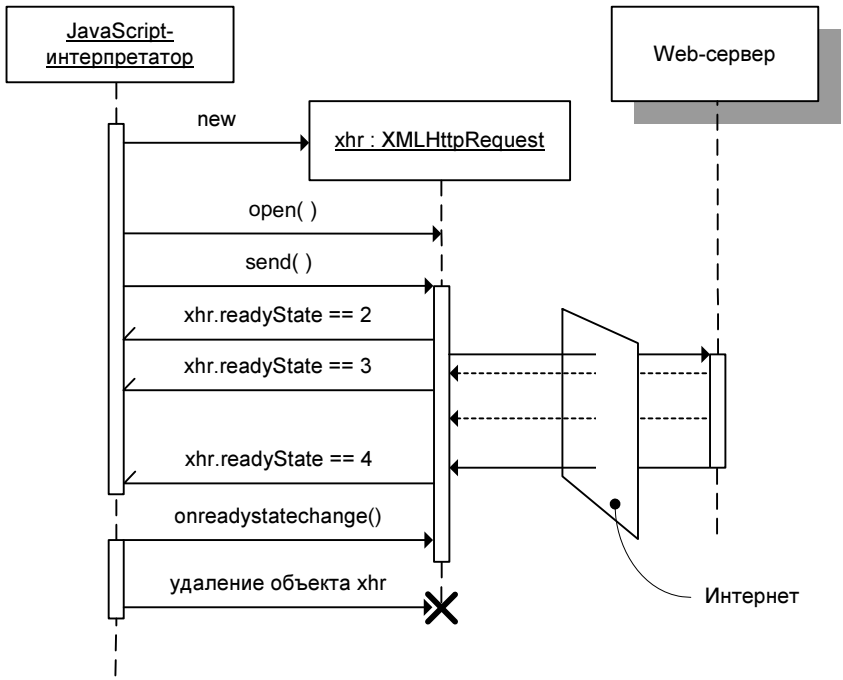


Рис. 2.2. Асинхронный запрос при помощи объекта `XMLHttpRequest`

На рис. 2.2 процесс работы с асинхронным запросом показан в виде диаграммы. Вы видите, что объект запроса создается оператором `new`, открывается методом `open()`, отсылается методом `send()`. После отправки методом

`send()` запрос начинает взаимодействовать с Web-сервером, а интерпретатор JavaScript продолжает работу с текущим скриптом. Так они работают параллельно, асинхронно, и объект `XMLHttpRequest` сообщает о прохождении запроса генерацией событий. События на схеме показаны стрелками с одной засечкой в виде вязального крючка (см. рис. 2.2). Состояние запроса можно определить из свойства `readyState` объекта `XMLHttpRequest`.

Обратите внимание на один очень важный момент. На схеме видно, что в моменты времени, когда поступают события от объекта `XMLHttpRequest`, интерпретатор JavaScript может быть занят выполнением текущего скрипта. Поэтому обработка событий может начинаться не сразу при их поступлении, а только после завершения работы текущего скрипта, когда интерпретатор JavaScript освободиться для обработки поступивших событий. До этого момента события, а это могут быть события сразу от нескольких объектов `XMLHttpRequest`, несколько событий от одного объекта `XMLHttpRequest`, события от пользовательского интерфейса — находятся в очереди и ожидают освобождения интерпретатора JavaScript. Освободиться интерпретатор JavaScript может не только после полного завершения работы текущего скрипта, а еще в случае его приостановки SUSPEND. Например, выполнение функции `window.alert()` вызывает приостановку SUSPEND текущего скрипта и дает возможность отработать обработчикам событий, которые находятся в очереди. Поэтому функцию `window.alert()` не рекомендуется использовать в Ajax-приложениях — это может непредсказуемо повлиять на порядок, в котором будут вызываться функции-обработчики событий. Замечу сразу, что явно вызывать приостановку SUSPEND текущего скрипта нельзя — такого оператора в JavaScript не существует.

После получения ответа Web-сервера вызывается функция-обработчик события `onreadystatechange` (см. рис. 2.2), которое генерируется при смене состояния запроса `XMLHttpRequest`. Вызов функции-обработчика события может происходить не сразу при поступлении соответствующего события, а только после высвобождения интерпретатора JavaScript, который может быть занят в момент поступления ответа Web-сервера выполнением текущего скрипта. Я особо обращаю ваше внимание на этот момент, потому что это не укладывается в стандартную модель асинхронной обработки событий и вызывает массу вопросов у начинающих Ajax-программистов. Дело в том, что Web-браузер является многопоточной средой, поэтому вы можете продолжать взаимодействовать с интерфейсом, в то время как выполняется загрузка изображений в HTML-элементы `IMG` или в то время, когда объект `XMLHttpRequest` взаимодействует с Web-сервером в асинхронном режиме. В отличие от Web-браузера, интерпретатор JavaScript допускает только последовательную работу программного кода JavaScript. Если в текущий