ГЛАВА

5

Введение в программные Ajax-оболочки

В ЭТОЙ ГЛАВЕ...

Желающие избежать самостоятельного написания большей части Ajax-кода могут воспользоваться одной из множества Ajax-оболочек, которая берет на себя основную работу. *Ajax-оболочки (Ajax framework)* содержат код, необходимый для работы Ajax-приложений, и этот код уже написан и проверен в деле. Все что нужно сделать для использования оболочки — загрузить ее и применить в своей работе (большая часть оболочек распространяется бесплатно). Чтобы сэкономить время при разработке Ajax-приложений, используйте Ajax-оболочки.

Большинство Ajax-оболочек представляют собой клиентские библиотеки JavaScript-кода, которые можно добавить к своим веб-страницам (эта тема рассматривается в данной и последующей главе). После этого, чтобы работать с Ajax, в разрабатываемых приложениях необходимо вызывать функции этих библиотек. Есть и другой вид Ajax-оболочек — серверные оболочки, они рассматриваются в главе 7.

Поскольку Ajax-оболочки бесплатны и часто разрабатываются энтузиастами как хобби, иногда для их создания используются далеко не самые правильные подходы к программированию. По этой причине данная глава начинается с описания специально разработанной для этой книги Ajax-оболочки, библиотеки Ajax Utility Library. В ходе этой разработки использовались хорошие приемы программирования. Оболочка Ajax Utility Library проста в установке и использовании.

Использование Ajax Utility Library

На рис. 5.1 показана страница, которая использует Ajax Utility Library. Чтобы загрузить текст, используя эту библиотеку, нужно просто нажать на кнопку.

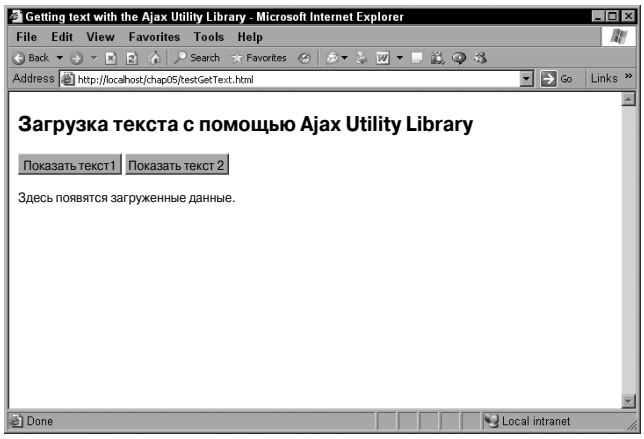


Рис. 5.1. Использование Ajax Utility Library

Когда пользователь нажимает на кнопку, библиотека Ajax Utility Library вступает в работу и загружает текстовое сообщение, как показано на рис. 5.2.

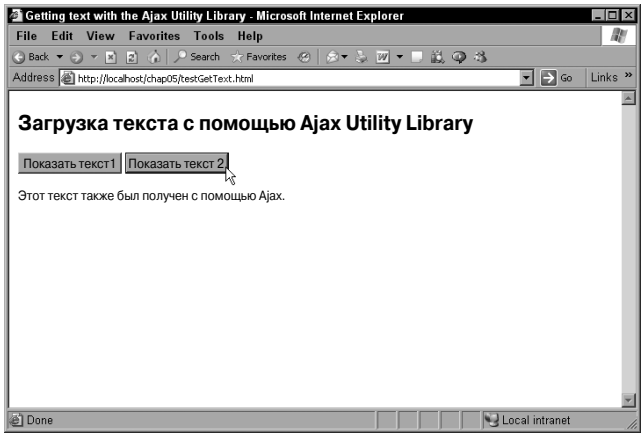
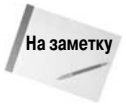


Рис. 5.2. Загрузка текста с помощью Ajax Utility Library

Это первая Ajax-оболочка, которая рассматривается в этой книге, а ее разработка будет описана в данной главе. Для того чтобы использовать эту Ajax-оболочку, достаточно подключить к своей веб-странице JavaScript-файл `ajaxutility.js`:

```
<script type = "text/javascript" src = "ajaxutility.js"></script>
```



При этом предполагается, что файл `ajaxutility.js` находится на сервере в том же каталоге, что и веб-страница, к которой он подключен. В противном случае необходимо указать полный URL к файлу `"ajaxutility.js"`, чтобы браузер мог его найти и загрузить.

Теперь можно свободно использовать функции библиотеки Ajax Utility Library, например, функцию `getText`, которая с помощью Ajax-методики получает текст с сервера. Для вызова функции `getText` необходимо указать URL получаемых данных (URL может быть относительным) и имя функции обратного вызова. Функции библиотеки Ajax Utility Library будут вызывать эту функцию вместе с загруженными данными.

Например, на странице, которая показана на рис. 5.1, кнопки связаны с функцией `getText` для загрузки файла `data.txt` или `data1.txt` и функций обратного вызова `callbackMessage1` или `callbackMessage2`:

```
<H1>Загрузка текста с помощью Ajax Utility Library</H1>

<form>
  <input type = "button" value = "Показать текст 1"
    onclick = "getText('data.txt', callbackMessage1)">
  <input type = "button" value = "Показать текст 2"
    onclick = "getText('data2.txt', callbackMessage2)">
</form>
```

Теперь необходимо только создать функции обратного вызова. Например, в функции `callbackMessage1`, которой передается текст загруженного файла `data.txt`, можно отобразить этот текст в `<div>`-элементе с именем `targetDiv`:

```
<script language = "javascript">

  function callbackMessage1(text)
  {
    document.getElementById("targetDiv").innerHTML = text;
  }
    .
    .
    .
</script>
```

и добавить `targetDiv`-элемент в тело веб-страницы (в элемент `<body>`):

```
<body>

  <H1>Загрузка текста с помощью Ajax Utility Library</H1>

  <form>
    <input type = "button" value = "Показать текст 1"
      onclick = "getText('data.txt', callbackMessage1)">
    <input type = "button" value = "Показать текст 2"
      onclick = "getText('data2.txt', callbackMessage2)">
  </form>

  <div id="targetDiv">
    <p>Здесь появятся загруженные данные.</p>
  </div>

</body>
```

Точно так же создается функция `callbackMessage2`, которой передается текст, загруженный из файла `data2.txt`:

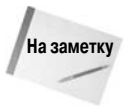
```
    <script language = "javascript">
      function callbackMessage1(text)
      {
        document.getElementById("targetDiv").innerHTML = text;
      }

      function callbackMessage2(text)
      {
        document.getElementById("targetDiv").innerHTML = text;
      }
</script>
```

В данном приложении используется Ajax, хотя не было написано ни одной строчки Ajax-кода. Просто к веб-странице был подключена Ajax-оболочка Ajax Utility Library (файл `ajaxutility.js`), а затем были вызваны функции этой Ajax-оболочки.

Какие функции поддерживает Ajax Utility Library? В файле `ajaxutility.js` определено четыре функции для получения текста и XML-данных путем отправки на сервер `GET`- и `POST`-запросов:

- `getText (urlToCall, functionToCallBack)`: функция использует метод `GET` для получения текста с сервера;
- `getText (urlToCall, functionToCallBack)`: функция использует метод `GET` для получения с сервера XML-данных;
- `postDataGetText (urlToCall, dataToSend, functionToCallBack)`: функция использует метод `POST` для отправки на сервер данных `dataToSend` и получает обратно текст; данные передаются в парах параметр/значение, например, `value=100`;
- `postDataGetXml (urlToCall, dataToSend, functionToCallBack)`: функция использует метод `POST` для отправки на сервер данных `dataToSend` и получает обратно XML; данные передаются в парах параметр/значение, например, `value=100`.

Как эти функции работают в `ajaxutility.js`? Все подробности описаны в следующих разделах.



Более подробная информация о передаче данных на сервер с помощью метода `GET` приведена в главе 3.

Получение текста с сервера с помощью функции `getText`

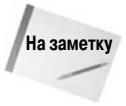
Первой функцией библиотеки Ajax Utility Library является функция `getText`, которая загружает текст с сервера. Чтобы использовать эту функцию, необходимо передать ей URL для доступа и имя функции обратного вызова, которая должна вызываться после завершения загрузки текста. Функция `getText` использует метод `GET` для сообщения с сервером.

Код функции начинается с объявления ее имени и указания передаваемых ей параметров (URL для вызова и имени функции обратного вызова для обработки загруженного текста):

```
function getText(urlToCall, functionToCallBack)
{
    .
    .
    .
}
```

Затем создается XMLHttpRequest-объект, который будет использоваться данной функцией для сообщения с сервером:

```
function getText(urlToCall, functionToCallBack)
{
  var XMLHttpRequestObject = false;
    .
    .
    .
}
```



Процесс создания XMLHttpRequest-объекта заключен в саму функцию getText. Это означает, что getText или любую другую функцию библиотеки Ajax Utility Library можно вызвать сколько угодно раз в любой последовательности и каждая функция будет использовать свой собственный XMLHttpRequest-объект, даже если она вызывается многократно.

Сначала функция getText пытается создать XMLHttpRequest-объект для браузеров Internet Explorer 7/Netscape/Firefox:

```
function getText(urlToCall, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  }
    .
    .
    .
}
```

Если эта попытка оказывается неудачной, функция getText пытается создать XMLHttpRequest-объект в Microsoft Internet Explorer:

```
function getText(urlToCall, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }
    .
    .
    .
}
```

Оставшийся код заключен в оператор if. Это необходимо для того, чтобы убедиться, что XMLHttpRequest-объект был создан до того, как сценарий попытается его использовать:

```
function getText(urlToCall, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }
```

```
  if(XMLHttpRequestObject) {
        .
        .
        .
  }
}
```

Если XMLHttpRequest-объект существует, функция getText открывает и конфигурирует его — в методе open этого объекта указывается метод передачи данных на сервер (GET), а также URL для доступа:

```
function getText(urlToCall, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }

  if(XMLHttpRequestObject) {
    XMLHttpRequestObject.open("GET", urlToCall);
        .
        .
        .
  }
}
```

Затем свойство onreadystatechange XMLHttpRequest-объекта подключается к безымянной внутренней функции

```
function getText(urlToCall, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }

  if(XMLHttpRequestObject) {
    XMLHttpRequestObject.open("GET", urlToCall);

    XMLHttpRequestObject.onreadystatechange = function()
    {
        .
        .
        .
    }
  }
}
```

В этой функции проверяется значение свойства readyState XMLHttpRequest-объекта, а также значение свойства status. Загрузка была завершена корректно, если значения этих свойств равны соответственно 4 и 200:

```
function getText(urlToCall, functionToCallBack)
{
```

```
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }

  if(XMLHttpRequestObject) {
    XMLHttpRequestObject.open("GET", urlToCall);

    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
        XMLHttpRequestObject.status == 200) {
        .
        .
        .
      }
    }
  }
}
```

Главная задача этой функции заключается в том, чтобы после загрузки текста вызвать callback-функцию и передать ей загруженный текст:

```
function getText(urlToCall, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }

  if(XMLHttpRequestObject) {
    XMLHttpRequestObject.open("GET", urlToCall);

    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
        XMLHttpRequestObject.status == 200) {
          functionToCallBack(XMLHttpRequestObject.responseText);
        .
        .
        .
      }
    }
  }
}
```

Чтобы убедиться, что `XMLHttpRequest`-объект не отнимает ресурсы, после того как он выполнил свое предназначение и более не нужен, используется оператор `delete` для явного удаления этого объекта:

```
function getText(urlToCall, functionToCallBack)
{
  var XMLHttpRequestObject = false;
```

```
  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }

  if(XMLHttpRequestObject) {
    XMLHttpRequestObject.open("GET", urlToCall);

    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
        XMLHttpRequestObject.status == 200) {
          functionToCallBack(XMLHttpRequestObject.responseText);
          delete XMLHttpRequestObject;
          XMLHttpRequestObject = null;
      }
    }
  }
}
```

Наконец, как всегда при использовании метода `GET`, на сервер отправляется значение `null`:

```
function getText(urlToCall, functionToCallBack)
{
  var XMLHttpRequestObject = false;
        .
        .
        .

    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
        XMLHttpRequestObject.status == 200) {
          functionToCallBack(XMLHttpRequestObject.responseText);
          delete XMLHttpRequestObject;
          XMLHttpRequestObject = null;
      }
    }

    XMLHttpRequestObject.send(null);
  }
}
```

В примере с веб-страницей `testGetText.html` (рис. 5.1) уже было показано, как применить функцию `getText` — функции передается URL и имя функции обратного вызова:

```
<form>
  <input type = "button" value = "Показать текст 1"
    onclick = "getText('data.txt', callbackMessage1)">
  <input type = "button" value = "Показать текст 2"
    onclick = "getText('data2.txt', callbackMessage2)">
</form>
```

Полученный с сервера текст передается функции обратного вызова, в которой его можно обработать любым способом, например, отобразить на веб-странице:

```
<script language = "javascript">
  function callbackMessage1(text)
  {
    document.getElementById("targetDiv").innerHTML = text;
```

```
      }

      function callbackMessage2(text)
      {
        document.getElementById("targetDiv").innerHTML = text;
      }
</script>
```

Так решается простейшая задача: получение текста с сервера. Далее рассматривается получение XML-данных и использование функции `getXML`.

Получение XML-данных с помощью функции `getXML`

Функция `getXML` библиотеки Ajax Utility Library позволяет, используя метод `GET`, загружать с сервера XML-данные. Как и в случае `getText`, функции `getXML` нужно просто передать URL XML-файла для загрузки и имя функции обратного вызова, которая будет обрабатывать загруженный XML-файл.

Вот как начинается функция `getXML` в файле `ajaxutility.js`:

```
function getXml(urlToCall, functionToCallBack){
    .
    .
    .
}
```

Как нетрудно предположить, эта функция работает аналогично `getText`, но с некоторыми отличиями, поскольку обрабатывается не простой текст, а XML.

В частности, для получения загруженного XML-текста она использует свойство `responseXML` `XMLHttpRequest`-объекта:

```
function getXml(urlToCall, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }

  if(XMLHttpRequestObject) {
    XMLHttpRequestObject.open("GET", urlToCall);

    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
        XMLHttpRequestObject.status == 200) {
          functionToCallBack(XMLHttpRequestObject.responseXML);
          delete XMLHttpRequestObject;
          XMLHttpRequestObject = null;
      }
    }

    XMLHttpRequestObject.send(null);
  }
}
```

Вот так выглядит функция getXML.

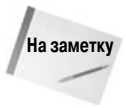
Как применить эту функцию в работе? Например, можно модифицировать страницу lunch.html из главы 3, которая загружает два XML-документа menu1.xml и menu2.xml и отображает в раскрывающемся списке пункты меню, которые содержатся в этих документах.

Сначала к странице textGetXml.html необходимо подключить файл ajaxutility.js:

```
<html>
  <head>
    <title>Загрузка XML с помощью Ajax Utility Library</title>

    <script type = "text/javascript" src =
      "ajaxutility.js"></script>
        .
        .
        .
```

Затем добавляются необходимые элементы управления: элемент <select> (список) для отображения пунктов меню, две кнопки, позволяющие пользователям выбрать одно из меню, а также -элемент, в котором отображается выбранный пользователем пункт меню:

```
    <form>
      <select size="1" id="menuList"
        onchange="setmenu()">
        <option>Выберите пункт меню</option>
      </select>
      <br>
      <br>
      <input type = "button" value = "Выбрать меню 1"
        onclick = "getXml('menu1.xml', getmenu1)">
      <input type = "button" value = "Выбрать меню 2"
        onclick = "getXml('menu2.xml', getmenu2)">
    </form>

    <div id="targetDiv" width =100 height=100>Выбранное вами блюдо
появится здесь.</div>
```



Каждая кнопка связана с функцией getXML — она направляет эту функцию на загрузку файла menu1.xml или menu2.xml и использование функции обратного вызова getmenu1 или getmenu2 соответственно.

Функциям обратного вызова getmenu1 и getmenu2 передаются загруженные с сервера XML-данные в форме JavaScript XML-документа. Функции декодируют XML-данные и передают их функции listmenu, которая отображает пункты меню в элементе <select>:

```
<html>
  <head>

    <title>Загрузка XML с помощью Ajax Utility Library</title>

    <script type = "text/javascript" src = "ajaxutility.js"></script>

    <script language = "javascript">
      var menu;

      function getmenu1(xmlDocument)
      {
        menu = xmlDocument.getElementsByTagName("menuitem");
```

```
    listmenu();
  }

  function getmenu2(xmlDocument)
  {
    menu = xmlDocument.getElementsByTagName("menuitem");
    listmenu();
  }

  function listmenu ()
  {
    var loopIndex;
    var selectControl = document.getElementById('menuList');

    for (loopIndex = 0; loopIndex < menu.length; loopIndex++ )
    {
        selectControl.options[loopIndex] = new
           Option(menu[loopIndex].firstChild.data);
    }
}
    .
    .
    .
```

Остается только создать функцию `setmenu`, которая вызывается благодаря элементу `<select>`, когда пользователь выбирает пункт меню, и отображает пользовательский выбор на веб-странице:

```
<html>
  <head>

    <title>Загрузка XML с помощью Ajax Utility Library</title>

    <script type = "text/javascript" src = "ajaxutility.js"></script>

    <script language = "javascript">
      var menu;

      function getmenu1(xmlDocument)
      {
        menu = xmlDocument.getElementsByTagName("menuitem");
        listmenu();
      }

      function getmenu2(xmlDocument)
      {
        menu = xmlDocument.getElementsByTagName("menuitem");
        listmenu();
      }

      function listmenu ()
      {
        var loopIndex;
        var selectControl = document.getElementById('menuList');

        for (loopIndex = 0; loopIndex < menu.length; loopIndex++ )
        {
            selectControl.options[loopIndex] = new
               Option(menu[loopIndex].firstChild.data);
        }
    }
```

```
function setmenu()
{
  document.getElementById('targetDiv').innerHTML =
    "Вы выбрали " + menu[document.getElementById
      ('menuList').selectedIndex].firstChild.data;
}
```

Как выглядят меню? В документе `menu1.xml` содержатся пункты `ham`, `turkey` и `beef`:

```
<?xml version = "1.0" ?>
<menu>
  <menuitem>Ham</menuitem>
  <menuitem>Turkey</menuitem>
  <menuitem>Beef</menuitem>
</menu>
```

а в документе `menu2.xml` — пункты `tomato`, `cucumber` и `rice`:

```
<?xml version = "1.0" ?>
<menu>
  <menuitem>Tomato</menuitem>
  <menuitem>Cucumber</menuitem>
  <menuitem>Rice</menuitem>
</menu>
```

В браузере на странице `testGetXml.html` выглядит так, как показано на рис. 5.3.

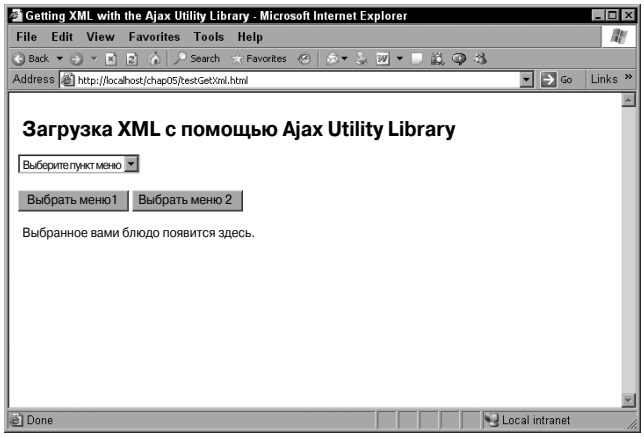


Рис. 5.3. Загрузка XML с помощью Ajax Utility Library

Когда пользователь нажимает кнопку “Выбрать меню 1”, в раскрывающемся списке появляются пункты первого меню (`ham`, `turkey` и `beef`), как показано на рис. 5.4.

А если пользователь нажмет кнопку “Выбрать меню 2”, то в списке появятся пункты второго меню (`tomato`, `cucumber` и `rice`), как показано на рис. 5.5.

Как видите, библиотека Ajax Utility Library способна загружать с сервера XML-данные и передавать их в любой JavaScript-сценарий. Конечно, необходимо знать, как обрабатывать полученные XML-данные, потому что они поступают в форме JavaScript XML-документа.

Один из способов облегчить обработку полученных XML-данных заключается в расширении библиотеки Ajax Utility Library удобными дополнительными функциями. Например, можно добавить функцию `getElements`, которая будет распаковывать объект XML-документа и возвращать массив необходимых элементов. Этой функции можно передавать имя искомого элемента:

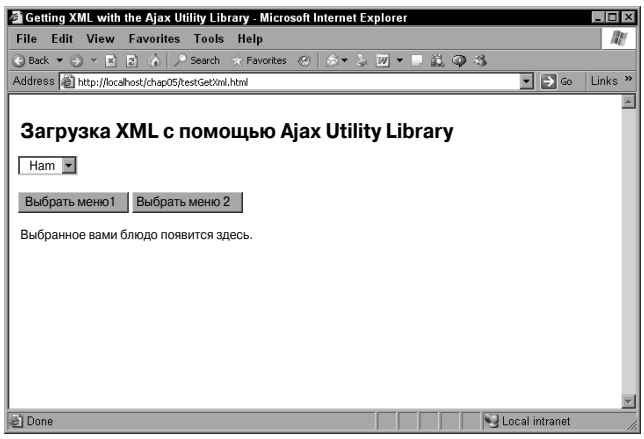


Рис. 5.4. Загрузка первого меню

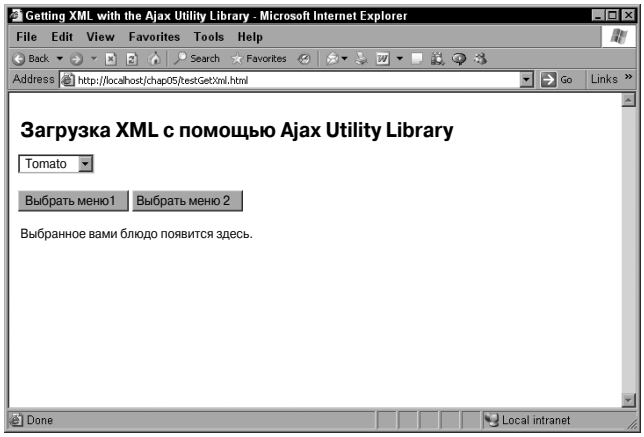


Рис. 5.5. Загрузка второго меню

```
function getElements(elementName)
{
    .
    .
    .
}
```

Функция будет использовать метод `getElementsByTagName` объекта XML-документа для поиска и возвращения массива запрашиваемых элементов или значения `null`, если таких элементов нет:

```
function getElements(elementName)
{
    .return xmlDocument.getElementsByTagName(elementName);
}
```

В данном разделе рассматривалось получение текста и XML-данных с сервера с помощью функций `getText` и `getXML`. Далее обсуждается тема отправки данных на сервер и получения от сервера текста с помощью метода `POST`.

Отправка данных на сервер и получение текста в ответ

Как уже отмечалось, Ajax-приложения обычно отправляют на сервер некоторые данные, а затем получают обратно другие данные. Этот процесс поддерживается в Ajax Utility Library двумя функциями: `postDataGetText` и `postDataGetXml`. Первая функция, `postDataGetText`, позволяет отправлять на сервер данные и получать обратно текст.

Совет

Отправлять данные с помощью метода `POST` не обязательно. Для передачи данных можно также использовать функции `getText` и `getXML` из библиотеки Ajax Utility Library. Просто эти данные необходимо URL-кодировать и присоединять их в конец URL, например, `scripter.php?data=Now+is+the+time`.

Функции `postDataGetText` передается URL для доступа, данные для отправки на этот URL и имя функции обратного вызова:

```
function postDataGetText(urlToCall, dataToSend, functionToCallBack)
{
    .
    .
    .
}
```

Как и другие функции в Ajax Utility Library, `postDataGetText` начинается с создания `XMLHttpRequest`-объекта:

```
function postDataGetText(urlToCall, dataToSend, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }
    .
    .
    .
}
```

Далее выполняется проверка того, было ли создание `XMLHttpRequest`-объекта успешным:

```
function postDataGetText(urlToCall, dataToSend, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }

  if(XMLHttpRequestObject) {
    .
```

```
    .
    .
  }
}
```

Подключение к серверу начинается с метода open XMLHttpRequest-объекта, в котором указывается метод POST, а также URL для подключения, переданный функции postDataGetText:

```
function postDataGetText(urlToCall, dataToSend, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }

  if(XMLHttpRequestObject) {
    XMLHttpRequestObject.open("POST", urlToCall);
      .
      .
      .
}
```

При использовании в Ajax метода POST необходимо также установить заголовок Content-Type в application/x-www-form-urlencoded:

```
function postDataGetText(urlToCall, dataToSend, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }

  if(XMLHttpRequestObject) {
    XMLHttpRequestObject.open("POST", urlToCall);
    XMLHttpRequestObject.setRequestHeader('Content-Type',
      'application/x-www-form-urlencoded');
        .
        .
        .
  }
}
```

Теперь можно подключать безымянную внутреннюю функцию к свойству onreadystatechange объекта XMLHttpRequest и ждать загрузки данных:

```
function postDataGetText(urlToCall, dataToSend, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
```

```
  }

  if(XMLHttpRequestObject) {
    XMLHttpRequestObject.open("POST", urlToCall);
    XMLHttpRequestObject.setRequestHeader('Content-Type',
      'application/x-www-form-urlencoded');

    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
        XMLHttpRequestObject.status == 200) {
          .
          .
          .
      }
    }
}
```

После загрузки данных функция `postDataGetText` передает их указанной функции обратного вызова, а затем удаляет `XMLHttpRequest`-объект:

```
function postDataGetText(urlToCall, dataToSend, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }

  if(XMLHttpRequestObject) {
    XMLHttpRequestObject.open("POST", urlToCall);
    XMLHttpRequestObject.setRequestHeader('Content-Type',
      'application/x-www-form-urlencoded');

    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
        XMLHttpRequestObject.status == 200) {
          functionToCallBack(XMLHttpRequestObject.responseText);
          delete XMLHttpRequestObject;
          XMLHttpRequestObject = null;
      }
    }
        .
        .
        .
  }
}
```

Теперь все готово и нужно только передать данные для отправки на сервер. В коде это выглядит так:

```
function postDataGetText(urlToCall, dataToSend, functionToCallBack)
{
  var XMLHttpRequestObject = false;
        .
        .
        .
    XMLHttpRequestObject.onreadystatechange = function()
```

```
    {
      if (XMLHttpRequestObject.readyState == 4 &&
        XMLHttpRequestObject.status == 200) {
          functionToCallBack(XMLHttpRequestObject.responseText);
          delete XMLHttpRequestObject;
          XMLHttpRequestObject = null;
      }
    }

    XMLHttpRequestObject.send(dataToSend);
  }
}
```

Так завершается функция `postDataGetText`. Рассмотрим ее применение.

Проверить функцию `postDataGetText` в работе можно с помощью веб-страницы `testpostDataGetText.html`, которая включена в список загружаемых файлов для этой главы. Эта страница просто отправляет текст небольшому PHP-сценарию, `echoer.php` в параметре `message`, а сценарий выводит этот текст в окне браузера. Ниже приведен код `echoer.php`:

```
<?
echo ($_POST["message"]);
?>
```

Как выглядит веб-страница `testpostDataGetText.html`? Ее код начинается с подключения библиотеки Ajax Utility Library. Это позволяет использовать в JavaScript-коде страницы функцию `postDataGetText`:

```
<html>
  <head>

    <title>Отправка данных и получение в ответ текста с помощью Ajax Util-
ity Library</title>

    <script type = "text/javascript" src =
      "ajaxutility.js"></script>
        .
        .
        .
```

Затем необходимо отобразить кнопку “Получить текст”, подключенную к функции `postDataGetText`. Странице передается URL, на который нужно отправить данные (`echoer.php`), а также собственно данные для отправки. Поскольку данные отправляются в формате `параметр=значение`, в примере данные передаются сценарию `echoer.php` в параметре `message`, значение которого равно `"Hello from Ajax Utility"`. Кроме того, передается имя функции обратного вызова, которой в свою очередь передаются загруженные данные. В данном случае эта функция называется `display`:

```
  <body>

    <h1>Отправка данных и получение в ответ текста с помощью Ajax Utility
Library</h1>

    <form>
      <input type = "button" value = "Получить текст"
      onclick = "postDataGetText('echoer.php',
        'message=Hello from Ajax Utility.', display)">
    </form>
        .
        .
        .
```

```
  </body>

</html>
```

Когда пользователь нажимает кнопку, данные "message=Hello from Ajax Utility" передаются сценарию echoer.php, который отправляет обратно в браузер текст "Hello from Ajax Utility", а функция postDataGetText передает этот текст функции обратного вызова display. Функция display отображает загруженный текст в -элементе:

```
<html>
  <head>

    <title>Отправка данных и получение в ответ текста с помощью Ajax Uti-
lity Library</title>

    <script type = "text/javascript" src = "ajaxutility.js"></script>

    <script language = "javascript">

      function display(text)
      {
        document.getElementById('targetDiv').innerHTML = text;
      }

    </script>
  </head>
```

Как все это выглядит в действии, показано на рис. 5.6.

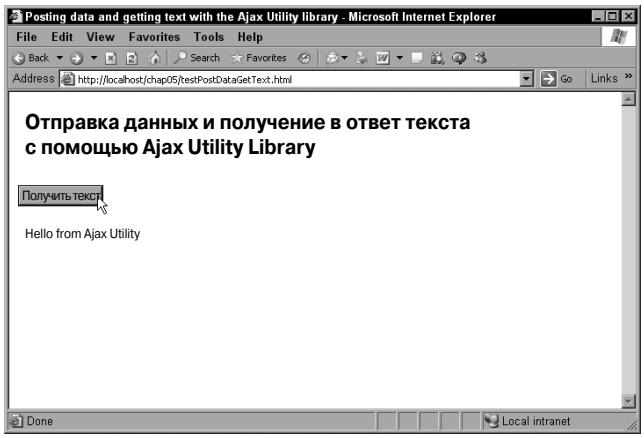


Рис. 5.6. Отправка текста и получение его обратно

Есть еще одна функция, которую стоит добавить в Ajax Utility Library, — это функция postDataGetXml, которая передает данные на сервер и в ответ получает XML-текст.

Отправка данных на сервер и получение XML-данных в ответ

Иногда в ответ на отправку данных на сервер необходимо получить не текст, а XML-данные. Библиотека Ajax Utility Library решает эту задачу с помощью функции `postDataGetXml`. В качестве параметров эта функция принимает URL для доступа, данные для отправки и имя функции обратного вызова:

```
function postDataGetXml(urlToCall, dataToSend, functionToCallBack)
{
    .
    .
    .
}
```

Как и другие функции, `postDataGetXml` начинается с создания `XMLHttpRequest`-объекта:

```
function postDataGetXml(urlToCall, dataToSend, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();

  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }
        .
        .
        .
}
```

Если `XMLHttpRequest`-объект был успешно создан, то его необходимо открыть, указав метод передачи данных (`POST`) и URL для подключения, который был передан функции `postDataGetXml`:

```
function postDataGetXml(urlToCall, dataToSend, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }

  if(XMLHttpRequestObject) {
    XMLHttpRequestObject.open("POST", urlToCall);
    XMLHttpRequestObject.setRequestHeader('Content-Type',
      'application/x-www-form-urlencoded');
        .
        .
        .
  }
}
```

Затем к свойству onreadystatechange этого объекта необходимо подключить безымянную внутреннюю функцию:

```
function postDataGetXml(urlToCall, dataToSend, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }

  if(XMLHttpRequestObject) {
    XMLHttpRequestObject.open("POST", urlToCall);
    XMLHttpRequestObject.setRequestHeader('Content-Type',
      'application/x-www-form-urlencoded');

    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
        XMLHttpRequestObject.status == 200) {
          .
          .
          .
      }
    }
  }
}
```

Когда XML-файл будет загружен, его содержимое будет передано функции обратного вызова, после чего XMLHttpRequest-объект будет удален:

```
function postDataGetXml(urlToCall, dataToSend, functionToCallBack)
{
  var XMLHttpRequestObject = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObject = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObject = new
     ActiveXObject("Microsoft.XMLHTTP");
  }

  if(XMLHttpRequestObject) {
    XMLHttpRequestObject.open("POST", urlToCall);
    XMLHttpRequestObject.setRequestHeader('Content-Type',
      'application/x-www-form-urlencoded');

    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
        XMLHttpRequestObject.status == 200) {
          functionToCallBack(XMLHttpRequestObject.responseXML);
          delete XMLHttpRequestObject;
          XMLHttpRequestObject = null;
      }
    }
  }
}
```

Остается только фактически отправить данные на сервер; в коде это выглядит так:

```
function postDataGetXml(urlToCall, dataToSend, functionToCallBack)
{
  var XMLHttpRequestObject = false;
        .
        .
        .
    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
        XMLHttpRequestObject.status == 200) {
          functionToCallBack(XMLHttpRequestObject.responseXML);
          delete XMLHttpRequestObject;
          XMLHttpRequestObject = null;
      }
    }

    XMLHttpRequestObject.send(dataToSend);
  }
}
```

Итак, функция `postDataGetXml` завершена и ее можно проверить в работе на новой веб-странице `testPostDataGetText.html`, которая включена в список загружаемых файлов для этой главы. Можно создать тестовый сценарий, `menus.php`, который будет принимать данные и отправлять обратно XML-текст для одного из двух меню. Например, если передать этому сценарию параметр `menu` со значением `1`, то сценарий сформирует первое меню:

```
<?
header("Content-type: text/xml");
if ($_POST["menu"] == "1")
  $menuitems = array('Ham', 'Turkey', 'Beef');
    .
    .
    .
?>
```

Если значение параметра `menu` будет равно `2`, то PHP-сценарий сформирует второе меню:

```
<?
header("Content-type: text/xml");
if ($_POST["menu"] == "1")
  $menuitems = array('Ham', 'Turkey', 'Beef');
if ($_POST["menu"] == "2")
  $menuitems = array('Tomato', 'Cucumber', 'Rice');



?>
```

Теперь нужно только сохранить меню в форме XML и передать обратно в браузер:

```
<?
header("Content-type: text/xml");
if ($_POST["menu"] == "1")
  $menuitems = array('Ham', 'Turkey', 'Beef');
if ($_POST["menu"] == "2")
  $menuitems = array('Tomato', 'Cucumber', 'Rice');
echo '<?xml version="1.0" ?>';
echo '<menu>';
foreach ($menuitems as $value)
{
```

```
  echo '<menuitem>';
  echo $value;
  echo '</menuitem>';
}
echo '</menu>';
?>
```

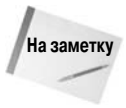
Итак, если передать сценарию `menus.php` данные в виде `menu=1`, то в ответ он вернет первое меню, а если в виде `menu=2`, то будет возвращено второе меню.

Теперь нужно написать веб-страницу `testPostDataGetXml.html`, которая заставляет работать сценарий `menus.php` и функцию `postDataGetXml`. Сначала следует добавить HTML-код для `<select>`-элемента, в котором отображаются пункты меню:

```
<form>
  <select size="1" id="menuList"
    onchange="setmenu()">
    <option>Выберите пункт меню</option>
  </select>
  .
  .
  .
</form>
```

Затем добавляются две кнопки, позволяющие пользователю выбрать одно из двух меню:

```
<form>
  <select size="1" id="menuList"
    onchange="setmenu()">
    <option>Выберите пункт меню</option>
  </select>
  <br>
  <br>
  <input type = "button" value = "Выбрать меню 1"
    onclick = "postDataGetXml('menus.php', 'menu=1', getmenu)">
  <input type = "button" value = "Выбрать меню 2"
    onclick = "postDataGetXml('menus.php', 'menu=2', getmenu)">
</form>
```



Отличие этих кнопок заключается в данных, передаваемых с их помощью функции `postDataGetXml`; одна передает `menu=1`, а вторая — `menu=2`, заставляя сценарий `menus.php` возвращать нужное меню.

Наконец, нужен `<div>`-элемент для отображения пользовательского выбора:

```
<form>
  <select size="1" id="menuList"
    onchange="setmenu()">
    <option>Выберите пункт меню</option>
  </select>
  <br>
  <br>
  <input type = "button" value = "Выбрать меню 1"
    onclick = "postDataGetXml('menus.php', 'menu=1', getmenu)">
  <input type = "button" value = "Выбрать меню 2"
    onclick = "postDataGetXml('menus.php', 'menu=2', getmenu)">
</form>
<div id="targetDiv" width =100 height=100>Здесь появится выбранное
вами блюдо.</div>
```

Это весь код для HTML-элементов. Большая часть работы здесь (отправка данных на сервер и получение XML-ответа) отводится функции `postDataGetXml` библиотеки Ajax Utility Library. Чтобы использовать эту функцию, необходимо подключить файл `ajaxutility.js`:

```
<html>
  <head>

    <title>Отправка данных и получение в ответ XML-текста с помощью Ajax
Utility Library</title>

    <script type = "text/javascript" src =
      "ajaxutility.js"></script>
    .
    .
    .
```

Функции обратного вызова (`getmenu`) передается JavaScript XML-документ, в котором содержится присланный сервером XML-текст:

```
<html>
  <head>

    <title>Отправка данных и получение в ответ XML-текста с помощью Ajax
Utility Library</title>

    <script type = "text/javascript" src = "ajaxutility.js"></script>

    <script language = "javascript">

      function getmenu(xmlDocument)
      {
        .
        .
        .
      }
        .
        .
        .
```

Функция `getmenu` может восстановить пункты меню (`<menuitem>`-элементы), сохранить их в массиве `menus`, а затем вызывать функцию `listmenu`, которая сформирует `<select>`-элемент из пунктов меню:

```
<html>
  <head>

    <title>Отправка данных и получение в ответ XML-текста с помощью Ajax
Utility Library</title>

    <script type = "text/javascript" src = "ajaxutility.js"></script>

    <script language = "javascript">

      var menu;

      function getmenu(xmlDocument)
      {
         menu = xmlDocument.getElementsByTagName("menuitem");
         listmenu();
      }
        .
        .
        .
```

Далее необходимо создать функцию listmenu:

```
<html>
  <head>

    <title>Отправка данных и получение в ответ XML-текста с помощью Ajax
Utility Library</title>

    <script type = "text/javascript" src = "ajaxutility.js"></script>

    <script language = "javascript">

      var menu;

      function getmenu(xmlDocument)
      {
         menu = xmlDocument.getElementsByTagName("menuitem");
         listmenu();
      }

      function listmenu ()
      {
        var loopIndex;
        var selectControl = document.getElementById('menuList');

        for (loopIndex = 0; loopIndex < menu.length; loopIndex++ )
        {
            selectControl.options[loopIndex] = new
               Option(menu[loopIndex].firstChild.data);
        }
    }
        .
        .
        .
```

Когда пользователь выбирает пункт меню, элемент управления <select> вызывает функцию setmenu:

```
<html>
  <head>

    <title>Отправка данных и получение в ответ XML-текста с помощью Ajax
Utility Library</title>

    <script type = "text/javascript" src = "ajaxutility.js"></script>

    <script language = "javascript">
        .
        .
        .
    function setmenu()
    {
      document.getElementById('targetDiv').innerHTML =
        "Вы выбрали" + menu[document.getElementById
          ('menuList').selectedIndex].firstChild.data;
    }

    </script>
  </head>
```

В работе страница testPostDataGetXml.html показана на рис. 5.7.

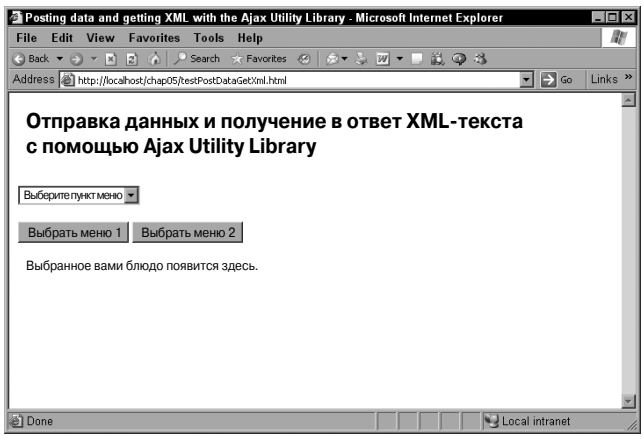


Рис. 5.7. Отправка данных и получение в ответ XML

Если пользователь нажмет на первую кнопку, приложение получит от сценария `menus.php` первое меню в формате XML, как показано на рис. 5.8. Второе меню выводится в результате нажатия на вторую кнопку (рис. 5.9).

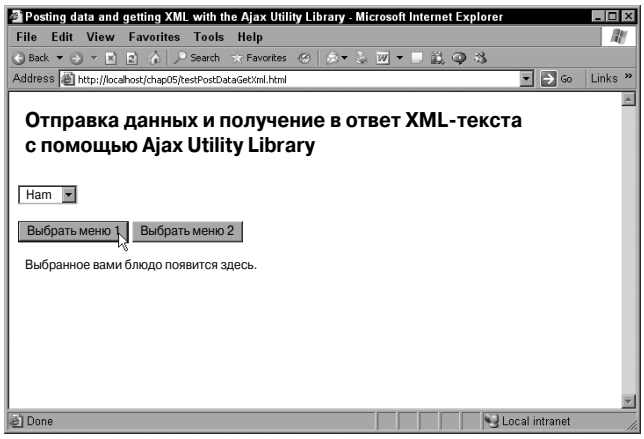


Рис. 5.8. Вывод меню 1

Примите поздравления! Только что вы заставили работать функцию `postDataGetXml` и вместе с ней библиотеку Ajax Utility Library.

Как видите, библиотека Ajax Utility Library оказывается полезной для работы с Ajax. Основа Ajax-программирования уже создана, и нет необходимости писать ее самостоятельно. Нужно лишь выбрать необходимую функцию для подключения к серверу — все остальное сделает Ajax Utility Library. Ниже перечислены доступные функции библиотеки:

- `getText(urlToCall, functionToCallBack);`
- `getXML(urlToCall, functionToCallBack);`
- `postDataGetText(urlToCall, dataToSend, functionToCallBack);`
- `postDataGetXml(urlToCall, dataToSend, functionToCallBack).`

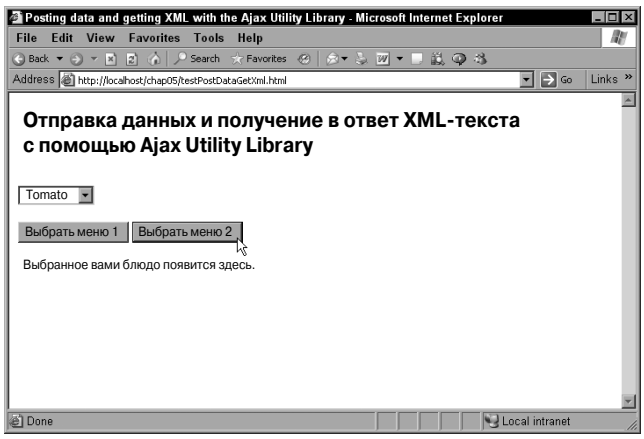


Рис. 5.9. Вывод меню 2

В этом и заключается идея, которая лежит в основе программных Ajax-оболочек; Ajax-код уже написан, и все, что остается сделать, — это вызывать необходимые функции библиотеки. Библиотека Ajax Utility Library — одна из Ajax-оболочек. Далее рассматриваются другие бесплатные библиотеки, начиная с `libXmlRequest`, которые вы также сможете применять в работе.

Использование библиотеки `libXmlRequest` для загрузки XML

Одной из популярных Ajax-оболочек является библиотека `libXmlRequest`, которую можно получить бесплатно на веб-странице `www.whitefrost.com/reference/2003/06/17/libXmlRecruest.html`. Как и Ajax Utility Library, эта библиотека представляет собой просто JavaScript-файл `libXmlRequest.js`.

Данная библиотека построена на основе двух главных Ajax-функций — `getXML` и `postXML`. Основное назначение библиотеки — работа с XML-данными, для этого в библиотеке имеется множество функций, позволяющих обрабатывать XML. Ниже представлен краткий обзор функций `libXmlRequest`:

- `getXml (url)`: *синхронный* `GET`-запрос; функция возвращает `null` или объект XML-документа;
- `getXml (url, handler, 1)`: асинхронный `GET`-запрос; функция возвращает значение `1`, если запрос был успешным, и обращается к функции обратного вызова после загрузки XML-данных;
- `postXml (url, data)`: *синхронный* `POST`-запрос; функция возвращает `null` или объект XML-документа;
- `postXml (url, data, handler, 1)`: асинхронный `POST`-запрос; функция возвращает `1`, если запрос был успешным, и вызывает функцию обратного вызова после загрузки XML-данных.

Функция обратного вызова (параметр *handler*) вызывается с двумя параметрами. Интересен второй из этих параметров, потому что он содержит загруженный XML. Свойство xdom этого параметра является XML-объектом, содержащим данные.

Чтобы использовать эту библиотеку, вызывая ее функции, необходимо предварять их имена префиксом org.cote.js.xml. Например, для вызова функции postXml необходимо ввести имя org.cote.js.xml.postXml.

Рассмотрим применение библиотеки libXmlRequest в веб-странице testlibXmlRequest.html. Предположим, что требуется прочитать текст внутри элемента <text>, который содержится в файле message.xml:

```
<?xml   version   =   "1.0"   ?>
<text>
Этот текст был получен с помощью libXmlRequest.
</text>
```

Рассмотрим загрузку и извлечение данных из файла message.xml с помощью libXmlRequest. Для начала необходимо подключить библиотеку libXmlRequest.js к веб-странице testlibXmlRequest.html:

```
<html>
  <head>
    <title>Использование библиотеки libXmlRequest</title>

    <script src = "libXmlRequest.js"></script>
        .
        .
        .
```

Затем можно добавить кнопку с надписью “Получить сообщение”:

```
    <form>
      <input type = "button" value = "Получить сообщение"
        .
        .
        .
    </form>
```

Когда пользователь нажимает кнопку, для загрузки файла message.xml можно вызвать асинхронную версию org.cote.js.xml.getXml, которая в свою очередь вызывает функцию обратного вызова callback:

```
    <form>
      <input type = "button" value = "Получить сообщение"
        onclick = "org.cote.js.xml.getXml('message.xml', callback, 1)">
    </form>
```

Для отображения загруженных данных можно добавить -элемент:

```
    <form>
      <input type = "button" value = "Получить сообщение"
        onclick = "org.cote.js.xml.getXml('message.xml', callback, 1)">
    </form>

    <div id="targetDiv">
      <p>Здесь появятся полученные данные.</p>
    </div>
```

Функция callback будет вызываться с двумя параметрами:

```
<html>
  <head>
    <title>Использование библиотеки libXmlRequest</title>

    <script src = "libXmlRequest.js"></script>
```

```
    <script language = "javascript">

      function callback(a, b)
      {
        .
        .
        .
      }
    </script>
  </head>
```

Второй параметр представляет особый интерес; его свойство xdom дает доступ к загруженным XML-данным. Чтобы получить массив XML-элементов, можно использовать метод getElementsByTagName. В данном случае единственным XML-элементом является элемент <text>, поэтому массив <text>-элементов можно получить так:

```
<html>
  <head>
    <title>Использование библиотеки libXmlRequest</title>

    <script src = "libXmlRequest.js"></script>

    <script language = "javascript">

      function callback(a, b)
      {
        var xmlData = b.xdom.getElementsByTagName("text");
        .
        .
        .
      }
    </script>
  </head>
```

В массиве должен содержаться только один элемент, поскольку в файле messages.xml есть только один <text>-элемент. Доступ к этому элементу можно получить по имени xmlData[0]. Текст внутри этого элемента хранится в текстовом узле, который является первым дочерним узлом xmlData[0], а доступ к тексту внутри текстового узла можно получить с помощью свойства data. Все это означает, что текст внутри <text>-элемента можно получить и отобразить на веб-странице с помощью следующего кода:

```
<html>
  <head>
    <title>Использование библиотеки libXmlRequest</title>

    <script src = "libXmlRequest.js"></script>

    <script language = "javascript">

      function callback(a, b)
      {
        var xmlData = b.xdom.getElementsByTagName("text");

        var div = document.getElementById('targetDiv');

        div.innerHTML = xmlData[0].firstChild.data;

      }
    </script>
  </head>
```

На рис. 5.10 показана страница `testlibXmlRequest.html`. Когда пользователь нажимает кнопку, `libXmlRequest`-функция `getXML` "за кулисами" извлекает данные из файла `message.xml`, которые впоследствии отображаются на веб-странице.

Библиотека `libXmlRequest` является лишь одной из многих Ajax-оболочек. Далее рассматривается библиотека AJAXLib.

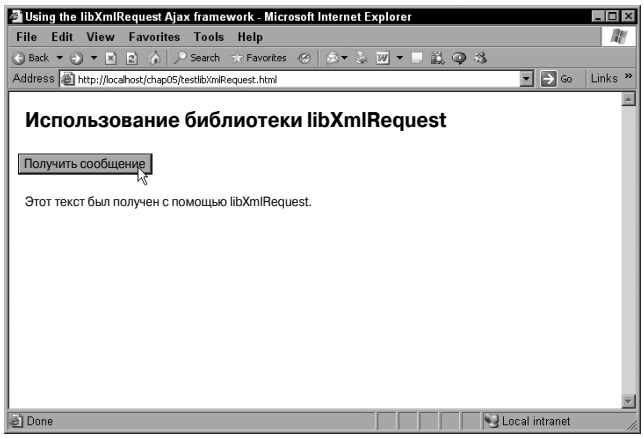


Рис. 5.10. Загрузка XML-данных с помощью `libXmlRequest`

Использование библиотеки AJAXLib для загрузки XML

AJAXLib — очень простая оболочка Ajax, которую можно получить бесплатно на странице `http://karaszewski.com/tools/ajaxlib/`. Библиотека представляет собой JavaScript-файл `ajaxlib.js`.

Рассматриваемая библиотека проста в использовании. Все, что нужно сделать, это вызвать функцию `loadXMLDoc` библиотеки, передав ей URL XML-источника, имя функции обратного вызова, которая вызывается после загрузки XML, а также значение `true/false`, позволяющее удалять пробельные символы из загруженного XML-документа (`true` означает, что AJAXLib удалит все пробельные символы). Загруженные XML-данные доступны через переменную `resultXML`.

Рассмотрим пример. Предположим, что нужно прочитать текст из нового XML-документа, `message2.xml`, со следующим содержимым:

```
<?xml version = "1.0" ?>
<text>
Этот текст был получен с помощью AJAXLib.
</text>
```

Сначала, как обычно, нужно подключить библиотеку `ajaxlib.js` к веб-странице `testAJAXLib.html`:

```
<html>
  <head>
    <title>Использование AJAXLib</title>

    <script src = "ajaxlib.js"></script>
```

Чтобы загрузить XML, можно подключить кнопку к AJAXLib-функции `loadXMLDoc`, указав ей, что нужно загрузить файл `message2.xml`, вызывать после загрузки функцию `callback`, и не удалять пробелы в XML:

```
<body>

  <H1>Использование AJAXLib</H1>

  <form>
    <input type = "button" value = "Получить сообщение"
      onclick = "loadXMLDoc('message2.xml', callback, false)">
  </form>
```

Кроме того, необходимо создать `<div>`-элемент для отображения загруженных XML-данных:

```
  <form>
    <input type = "button" value = "Получить сообщение"
      onclick = "loadXMLDoc('message2.xml', callback, false)">
  </form>

  <div id="targetDiv">
    <p>Здесь появятся полученные данные.</p>
  </div>
```

Если функция `loadXMLDoc` успешно выполняет свои задачи, то она вызывает функцию `callback`. В этой функции доступна переменная `resultXML`, введенная библиотекой AJAXLib. В этой переменной содержатся XML-данные, сохраненные в виде JavaScript-объекта XML-документа. Таким образом, чтобы извлечь все загруженные `<text>`-элементы, можно использовать метод `getElementsByTagName`:

```
<html>
  <head>
    <title>Использование AJAXLib</title>

    <script src = "ajaxlib.js"></script>

    <script language = "javascript">

      function callback()
      {
        var xmlData = resultXML.getElementsByTagName("text");

        .
        .
        .
      }
    </script>
  </head>
```

Поскольку в массиве, созданном методом `getElementsByTagName`, существует только один `<text>`-элемент, он доступен по имени `xmlData[0]`. Оставшаяся часть процесса извлечения текстовых данных из этого элемента такая же, как и в случае с библиотекой `libXmlRequest`:

```
<html>
  <head>
    <title>Использование AJAXLib</title>

    <script src = "ajaxlib.js"></script>

    <script language = "javascript">
```

```
    function callback()
    {
      var xmlData = resultXML.getElementsByTagName("text");

      var div = document.getElementById('targetDiv');

      div.innerHTML = xmlData[0].firstChild.data;

    }
  </script>
</head>
```

На практике это выглядит так, как показано на рис. 5.11 — пользователь нажал на кнопку, и функция `loadXMLDoc` выполнила свои задачи, загрузив данные, которые затем были отображены на веб-странице.

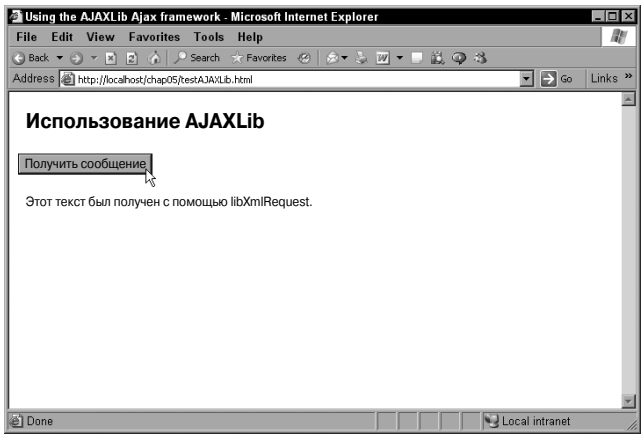


Рис. 5.11. Загрузка XML-данных с помощью AJAXLib

Резюме

В этой главе рассматривались три Ajax-оболочки: Ajax Utility, libXmlRequest и AJAXLib. Это только три библиотеки из огромного множества подобных Ajax-оболочек. В двух следующих главах описаны еще несколько библиотек.