AJAX and SQL

CS174

Chris Pollett

Oct 18, 2006.

Outline

- AJAX
- SQL

Beginning AJAX

- AJAX stands for Asynchronous Javascript and XML.
- Asynchronous means that one does not need to wait for the response to an HTTP request to do something
  - one can use Javascript to have several outstanding HTTP requests and still make things happen in the browser.
  - Further one doesn't have to reload the entire page when the results of a request are returns.
- The XML in AJAX is because the results of a request are usual XML, JSON, or YAML data which is then formatted by some Javascript code.
- The basic key to AJAX is the ability for Javascript to make an HTTP request. This is done with the XMLHttpRequest object. As an example, look at the Javascript in the file:

http://www.cs.sjsu.edu/faculty/pollett/test/dept3.html

Step by Step

- To create an XMLHttpRequest one could simply write in Javascript:

  ```
  request = new XMLHttpRequest()
  ```

- This works on modern browsers. For older browsers like IE6 you need to do something like:

  ```
  request = new ActiveXObject('MSXML2.XMLHTTP')
  ```

- For older still use:

  ```
  request = new ActiveXObject('Microsoft.XMLHTTP')
  ```

- See the example page for how checking for browsers can be done using a try-catch block.
- At this point if we need to set up HTTP request headers one can use:

  ```
  request.setRequestHeader("name", "value")
  ```

Step by Step Continued I

- To open a connection back to the server the Javascript came from one can then do:

  request.open(*theHTTPmethod, theURL, theAsync flag*)

  or

  request.open(*theHTTPmethod, theURL, theAsync flag, username, password*)

- For example,

  request.open("GET", "progress.php", true)

- The asynchronous flag says whether or not the Javascript can continue executing (true) or if it must wait for a response (false).
- At this point no data has been sent yet.

Step by Step Continued II

- We next need to set up a callback function which will be called as we get data back from the server:

```
var self = this; /* remember scope of enclosing
                    object */
request.onreadystatechange = function()
{
   switch(request.readyState)
   {
     case 0:// handle uninitialized case
     case 1: // handle open but no send case
     case 2: // handle send but no response case
     case 3: // handle response is being downloaded case
     case 4: // handle response has completed being downloaded case
   }
}
```

Step by Step Continued III

- In case 3 or 4 above, request.responseText or request.responseXML will contain the data that has been returned by the server
- In case 3 or 4 above, request.getAllResponseHeaders() can be used to get the HTTP response headers.
- Sometimes it is useful to start a timer in case 3, using setTimout. If the request is taking to long so that the timer gets called, one can then abort the request using: request.abort.
- Typically, the code in 3 or 4 above would put the response text into some tag in our document. This could be done with:

```
document.getElementById(“myDivTag”).innerHTML = \ request.responseText; /*not
    really a DOM standard but all browsers happy with */
//or we could do…
myDiv = document.getElementById(“myDivTag”)
if(myDiv.firstChild)
{
    myDiv.removeChild(myDiv.firstChild);
}
myDiv.appendChild(document.createTextNode(request.responseText));
```

Step by Step Continued IV

- Once we have set up our callback function we are ready to send data to the server.
- To do this we can do:

```
request.send(null);
/* note: send’s argument can be used if using
   POST method to send the posted data */
```

- That’s it! We just sit back then and wait for our callback to be called.

Web Services

- A *web service* is a programming interface which can be invoked over HTTP.
- The first attempts to standardize such services made use of things like WSDL, SOAP, UDDI, etc. XML languages which tended to violate the KISS (keep it simple stupid) principle.
- Some simpler web service interfaces have been written by major companies using XML-RPC, JSON-RPC and REST.
- A XML-RPC document is a XML document which specifies a *remote procedure call*, i.e., which function of which object to invoke on some server; or it specifies the response of such a call. It later evolved into the more complicated and still evolving SOAP (Simple Object Access protocal)
- JSON-RPC is like XML-RPC but uses JavaScript Object Notation -- basically, a snippet of javascript code for an object.
- REST stands for *Representational State Transfer*. Here the idea is that an application state/method is viewed a resource. Each resource has a URL. There is a well defined way to tack on to this URL a query to invoke the function and return results. For example, the Yahoo News Rest Service might be invoked with a line like:

http://search.yahooapis.com/NewsSearchService/V1/newsSearch?appid=YahooDemo&query=madonna&results=2&language=en

Proxies

- Javascript function is only allowed to make requests back to the server from which it came.
- So if you have a page http://somewhere.com/index.html
  - and you would like the Javascript on it to make use of the Yahoo API, how do you do it?
- You need to use a proxy on your server which passes the request onto Yahoo!
- One example of a PHP script to do such proxy-ing can be found at:

http://developer.yahoo.com/javascript/samples/proxy/php_proxy_simple.txt

- To use such a proxy, you need to have PHP running on your machine.
- You could rename the above file proxy.php set its permissions so that is executable and put it somewhere you know under your document root.
- Then to access the Yahoo! service via the proxy you could do:

http://yourServer/proxy.php?yws_path=urlencodepath

- For example,

http://www.cs.sjsu.edu/faculty/pollett/test/proxy.php?yws_path=NewsSearchService%2FV1%2FnewsSearch%3Fappid%3DYahooDemo%26query%3Dmadonna%26results%3D2%26language%3Den

Databases

- We are now almost ready to switch our attention to different kinds of server side languages.
- Our server side programs will frequently interact with a database, so it is useful to learn a little about databases first.
- A database consists of a collection of tables of data.
- A database management system (DBMS) is typically used to manage several databases.
- A DBMS is used to manage both the permanent storage of data as well as access to this data.
- Each table has a schema consisting of a name for the table, together with the name and types of each of its columns.
- The data in a table consists of sequences of rows of data, Each column in a row containing data of the type specified for that column in the schema.
- SQL (Standard Query Language) is a language used to both define databases as well as to interact with databases.
- For this class we will use the mySQL database management system.

Using mySQL

- At a command prompt we can connect to mySQL database using:

```
>mysql -h hostname -u username -p
******
```

- To exit mysql one can type QUIT.
- To execute a sequence of commands in a file one can do:

```
SOURCE file.sql
```

- To see what databases are available one can do:

```
SHOW DATABASES;
```

- To create a database:

```
CREATE DATABASE someDB;
```

- To use a database one can then do:

```
USE someDB
```

- To see what tables a database has one can do:

```
SHOW TABLES;
```

More Using mySQL

- To create a table one can do:

```
CREATE TABLE mytable
(
   FirstName varchar(10),
   LastName varchar(20),
   Birth date,
   Salary INTEGER /* notice case insensitive */
);
```

- To see the schema for a table one can do:

```
DESCRIBE mytable;
```

- To insert a row into a table one can do:

```
insert into mytable values(‘fname’, ‘lname’, ‘1970-08-19’, ‘2000’);
```

- To see the contents of a table one can do:

```
SELECT * from mytable;
```

- To delete rows from a table:

```
delete from mytable where FirstName=‘fname’;
```

- To delete a table or a database:

```
DROP mytable;
```