AJAX made simple with DWR

Start using AJAX in your Web application with DWR

Summary
This article explains the advantages of using the open source project DWR (Direct Web Remoting) with AJAX (Asynchronous JavaScript and XML) concepts to improve Web application usability. The author takes a step-by-step approach to show how DWR makes adopting AJAX simple and fast. (*1,600 words;* **June 20, 2005**)

By Cloves Carneiro Jr.

AJAX, or Asynchronous JavaScript and XML, describes a Web development technique for creating interactive Web applications using a combination of HTML (or XHTML) and Cascading Style Sheets for presenting information; Document Object Model (DOM); JavaScript, to dynamically display and interact with the information presented; and the `XMLHttpRequest` object to interchange and manipulate data asynchronously with the Web server.

Many examples on the Internet show all the necessary steps for using `XMLHttpRequest` to communicate with the server from within an HTML file. When manually writing and maintaining the `XMLHttpRequest` code, a developer must deal with many potential problems, especially with cross-browser compatibilities like different DOM implementations. This can lead to countless hours spent coding and debugging JavaScript code, which is not known to be developer friendly.

The DWR (Direct Web Remoting) project is an open source solution under the Apache license for the developer who wants to use AJAX and `XMLHttpRequest` in an easy way. It has a set of JavaScript functions that remove the complexity from calling methods in a Java object running on the application server from the HTML page. It handles parameters of different types and helps keep the HTML code readable.

DWR is not intrusive to one's design, as it does not force any sort of inheritance architecture for objects to be exposed. It fits well in any application that runs in a servlet framework. For the less DHTML-experienced developers, DWR also provides a JavaScript library to help with frequently used DHTML tasks, like populating tables, filling select boxes with items, and changing the content of HTML elements such as `<div>` and `<span>`.

The DWR Website is comprehensive and has a fair amount of documentation, which has served as a foundation for this article. Some examples are provided to demonstrate how DWR can be used and what can be accomplished with the library.

This article allows the user to see a step-by-step creation of a Web application that uses DWR. I show you all the details necessary to create the sample application, which you'll be able to download and deploy in your environment to evaluate how DWR works.

Note: Finding information about AJAX is not difficult; several articles and blog entries on the Web cover the subject, each of which tries to identify and comment about a different aspect of the concept. In Resources, you will find some interesting links to examples and articles to learn more about AJAX.

Sample application

The example application used in this article simulates an apartment rental search engine for the city of Toronto. The user can select a set of search criteria before performing the search. To improve interaction, AJAX is used in two occasions:

- The application notifies the user of the number of search results that match his selection. This number is updated—using AJAX—as the user selects the amount of desired bedrooms and bathrooms, and the price range. By showing the user the number of results, the user won't need to hit the search button when no results or too many results match the user's criteria.
- The database query that retrieves the units from the database is performed using AJAX. The database search executes when the user presses the Show Results button. Thus, the application seems more responsive, as the whole page doesn't need to be reloaded to show the results.

Database

The database we use is HSQL, a Java SQL database engine with a small footprint, which can be bundled with the Web application with no additional installation and configuration. A SQL file is used to create the in-memory table and add some records when the Web application context is started.

Java classes

The application contains two main classes called `Apartment` and `ApartmentDAO`. The `Apartment.java` class is a simple Java class with attributes and getter/setter methods. `ApartmentDAO.java` is the data-access class used to query the database and get information based on the user's search criteria. The implementation of the `ApartmentDAO` class is straightforward; it uses straight Java Database Connectivity calls to get the total number of units and the list of available units matching the user's request.

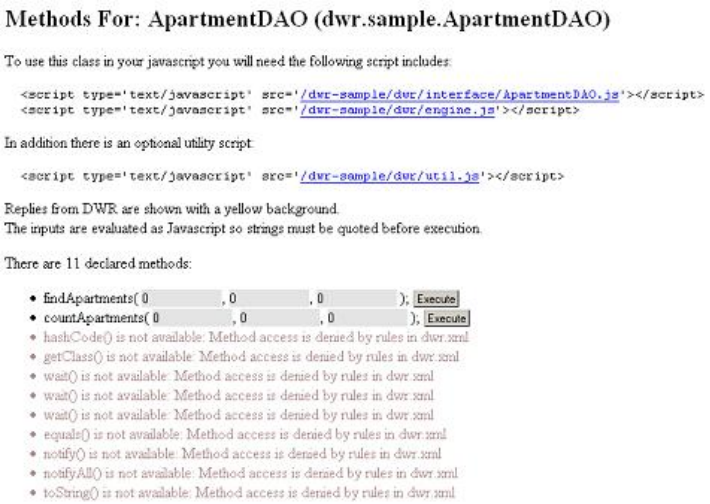
DWR configuration and use

Setting up DWR for use is easy: Copy the DWR jar file to the Web application's `WEB-INF/lib` directory, add a servlet declaration to `web.xml`, and create the DWR configuration file. The DWR distribution only requires the use of a single jar file. You must add the DWR servlet into the application's deployment descriptor in `WEB-INF/web.xml`:

```
    <servlet>
        <servlet-name>dwr-invoker</servlet-name>
        <display-name>DWR Servlet</display-name>
        <description>Direct Web Remoter Servlet</description>
        <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
        <init-param>
            <param-name>debug</param-name>
            <param-value>true</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>dwr-invoker</servlet-name>
        <url-pattern>/dwr/*</url-pattern>
    </servlet-mapping>
```

One optional step is to set up DWR in debug mode—illustrated in the example above—by setting the `debug` parameter to `true` in the `servlet` declaration. With the DWR servlet in debug mode, you can see all the Java objects accessible from HTML pages. A page with a list of available objects appears at the WEBAPP/dwr URL, which shows the public methods for the selected class; the listed methods can be called from that page, allowing you to, for the first time, run methods on objects located on the server. The figure below shows what the debug page looks like:



Debug page

Now you must let DWR know what objects will receive requests through the `XMLHttpRequest` object. Accomplish that task by using DWR's own configuration file called `dwr.xml`. In the configuration file, you define the objects that DWR will allow you to call from your HTML pages. By design, DWR allows access to all the exposed class's public methods, but in our example, we only allow access to a few methods. Here's the configuration file for our example:

```
<dwr>
    <allow>
        <convert converter="bean" match="dwr.sample.Apartment"/>
        <create creator="new" javascript="ApartmentDAO" class="dwr.sample.ApartmentDAO">
            <include method="findApartments"/>
            <include method="countApartments"/>
        </create>
    </allow>
</dwr>
```

The above file accomplishes two goals in our example. First, the `<convert>` tag tells DWR to convert objects of the `dwr.sample.Apartment` type into JavaScript associative arrays, because, for security reasons, DWR doesn't convert regular beans by default. Second, the `<create>` tag makes DWR expose the `dwr.sample.ApartmentDAO` class to be called from JavaScript; the JavaScript file we use in our pages is defined by the `javascript` attribute. We must pay attention to the `<include>` tags, as those identify which methods from the `dwr.sample.ApartmentDAO` class will be made available.

HTML/JSP code

Once the configuration is done, you can start your Web application, and DWR will be ready to call the methods you need from your HTML/JavaServer Pages (JSP) page, without you creating JavaScript files. In the `search.jsp` file, we must add references to the JavaScript interface provided by DWR, as well as the DWR engine, by adding three lines to our code:

```
<script src='dwr/interface/ApartmentDAO.js'></script>
<script src='dwr/engine.js'></script>
<script src='dwr/util.js'></script>
```

The first use of AJAX in the example application can be noticed when the user changes the search criteria; as he can see, as the criteria changes, the number of available apartments is updated. I created two JavaScript functions: The `updateTotal()` function is

called when a value in one of the select boxes changes. The `ApartmentDAO.countApartments()` function is the most important piece. Most interesting is the first parameter, the `loadTotal()` function, which identifies the callback method that DWR will use when it receives a response from the server. `loadTotal()` is then called to display the result on the HTML page's `<div>`. Here's how the JavaScript functions are used in the described interaction scenario:

```
function updateTotal() {
    $("resultTable").style.display = 'none';
    var bedrooms = document.getElementById("bedrooms").value;
    var bathrooms = document.getElementById("bathrooms").value;
    var price = document.getElementById("price").value;
    ApartmentDAO.countApartments(loadTotal, bedrooms, bathrooms, price);
}

function loadTotal(data) {
    document.getElementById("totalRecords").innerHTML = data;
}
```

Obviously, the user will want to see the list of apartments that satisfy his search. So, when the user is satisfied with the criteria and total apartments available, he presses the Show Results button, which calls the `updateResults()` JavaScript method:

```
function updateResults() {

    DWRUtil.removeAllRows("apartmentsbody");
    var bedrooms = document.getElementById("bedrooms").value;
    var bathrooms = document.getElementById("bathrooms").value;
    var price = document.getElementById("price").value;
    ApartmentDAO.findApartments(fillTable, bedrooms, bathrooms, price);
    $("resultTable").style.display = '';
}

function fillTable(apartment) {
    DWRUtil.addRows("apartmentsbody", apartment, [ getId, getAddress, getBedrooms, getBathrooms, getPrice ]);
}
```

The `updateResults()` method clears the table area assigned to hold the search results, gets all the necessary parameters from the UI, and passes those parameters to the `ApartmentDAO` JavaScript object created by DWR. The database query is then performed, and the callback function `fillTable()` is called, which then parses the object returned by DWR, and prints them to the page.

Security concern

To keep the example brief, the `ApartmentDAO` class was kept as simple as possible, but such a class will usually have a set of methods to manipulate data, such as `insert()`, `update()`, and `delete()`. DWR exposes all public methods to any calling HTML page. For security reasons, it's not wise to expose your data-access layer methods in such a way. A developer could create a facade to centralize the communication between the calling JavaScript function and lower-level business components, thus limiting the exposed functionality.

Conclusion

This article's example only gets you started using AJAX with DWR in your own projects. DWR helps you keep focused on how to improve your application's interaction model, removing the burden that comes with coding and debugging JavaScript code. The most interesting challenge of using AJAX is defining where and how to improve usability. DWR helps you focus totally on how to make your application more user-friendly by handling the communication between the Webpage and your Java objects.

I would like to thank Mircea Oancea and Marcos Pereira for reviewing this article and giving valuable feedback.

About the author

Cloves Carneiro Jr. is software engineer working for Extend Media, Toronto, Canada. He is a Sun Certified Programmer and Sun Certified Web Component Developer. He has worked with Java since 1999 and specializes in server-side Java applications.

Resources

- Download the example application with full source:
  http://www.javaworld.com/javaworld/jw-06-2005/dwr/jw-0620-dwr.war
- DWR:
  http://www.getahead.ltd.uk/dwr/index.html
- HSQL:
  http://hsqldb.sourceforge.net
- Definition of AJAX:
  http://en.wikipedia.org/wiki/AJAX
- "AJAX: A New Approach to Web Applications," Jesse James Garrett (*Adaptive Path,* February 2005):
  http://www.adaptivepath.com/publications/essays/archives/000385.php
- "Very Dynamic Web Interfaces," Drew McLellan (*xml.com,* February 2005):
  http://www.xml.com/pub/a/2005/02/09/xml-http-request.html
- XMLHttpRequest & AJAX Working Examples:
  http://www.fiftyfoureleven.com/resources/programming/xmlhttprequest/examples
- "Usable XMLHttpRequest in Practice," Thomas Baekdal (Baekdal.com, March 2005):
  http://www.baekdal.com/articles/Usability/usable-XMLHttpRequest/
- "XMLHttpRequest Usability Guidelines," Thomas Baekdal (Baekdal.com, February 2005):
  http://www.baekdal.com/articles/Usability/XMLHttpRequest-guidelines/
- AJAX Matters:
  http://www.AJAXmatters.com/
- For more articles on XML, browse the **Java and XML** section of *JavaWorld*'s Topical Index:
  http://www.javaworld.com/channel_content/jw-xml-index.shtml
- For more articles on development tools, browse the **Development Tools** section of *JavaWorld*'s Topical Index:
  http://www.javaworld.com/channel_content/jw-tools-index.shtml