Consuming web services with the Dojo Toolkit

Use the Dojo Toolkit for Ajax and RESTful services

Nathan A. Good
Senior Consultant and Freelance Developer
Enterprise Frameworks

Skill Level: Intermediate

Date: 07 Sep 2010

Learn how to consume services using the Dojo Toolkit to enable Asynchronous JavaScript and XML (Ajax) on a web page. Using this article, understand traditional Ajax-style services and get an introduction to RESTful web services abilities in the Dojo Toolkit.

Get started with Dojo development http://www.ibm.com/developerworks/training/kp/wa-kp-dojo/index.html
This article is about consuming web services—both simple services and RESTful web services— using the Dojo Toolkit. To get the most out of this article, you need to have the following installed and configured on your system:

- A text editor or integrated development environment (IDE) (This article uses the Eclipse JavaScript IDE.)
- A web server

Dojo Toolkit overview

As the focus on building better Rich Internet Applications (RIAs) increases, JavaScript frameworks emerge to enable web developers to make their applications more engaging.

Prepackaged JavaScript libraries offer many advantages. First, using code that is already verified in different browsers and different platforms can significantly reduce the amount of testing required to verify functions on many different browsers, thereby reducing the amount of defects in your code. Second, using code that is already written and tested saves a substantial amount of time and enables you complete your application faster.

test, analyze, and deploy high-quality Java™, Java Platform, Enterprise Edition (Java EE), Web 2.0, service-oriented architecture (SOA), and portal applications. Rational Application Developer includes features to quickly build skills on emerging Java EE and web technologies, automate code verification, build and test, and enables agile software development for today's fast-paced software delivery needs.

The Dojo Toolkit (see Resources) is a collection of JavaScript code that offers significant functions. The Dojo provides JavaScript methods that you can use to animate elements, fade them in and out, and make Ajax calls.

This article is about using the Dojo Toolkit along with Ajax and RESTful web services to offer your users better functions and a better user experience. The article provides an auto-completion example that fills in a text box to match results as a user types. This user interaction pattern for searching has emerged as a commonplace—albeit smooth—way to make searches better for your users.

Ajax overview

Ajax is a term used to describe a combination of technologies to make calls (asynchronously) to a server while the user's web page stays loaded. This technique has become fairly ubiquitous in today's web pages and plays an important role in making the web experience richer.

In the example in this article, every time the user changes the text in a text box the JavaScript code makes a call to a server to get suggested values. On one hand, this additional function can cause a lot of traffic. On the other hand, if the user is searching repeatedly by posting the entire form, this technique can actually save some amount of traffic.

Creating the project

This example uses the latest version of Eclipse with the JavaScript tooling installed. The tooling offers improved ability to edit JavaScript and HTML pages.

Follow these steps to create a project you can use to create a few HTML files so you can follow the example:

1. Create a new static web project by using **File > New > Project**.
2. Select **Web\Static Web Project** and click **Next**.
3. Type the name of your project (for example, *MyDojoExample*).
4. Click **New Runtime**.
5. Select **New Server** from the list and select the **Create a new local server** check box.
6. Type the name of the server as it will display in the server list, and type the name of the directory in which you want to publish your files. Typically, this should be a directory location in which you can publish user web files.
7. Click **Next** on the Static Web Project wizard.

8. Leave the context root the same as the name of the project, and leave the name of the web content folder name WebContent.
9. Click **Finish**. Eclipse creates the new project for you.

When you add files to your project, Eclipse automatically publishes the files to the directory that you configured for your server. You can see this in action by creating your first HTML file in this project, which is an index.html file. You will modify this file to include the search box used for the example.

Follow these steps to create the new HTML file:

1. Select the WebContent folder in your new project and use your alternate mouse button to open the context menu.
2. From the menu, select **New > HTML file**.
3. Type the name of the file in the **File name** field, and click **Next** to see the HTML template chooser.
4. For this example, select the **New XHTML File (1.0 strict)** option and click the **Finish** button.

The new file will look like Listing 1.

Listing 1. The new index.html file

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Insert title here</title>
</head>
<body>

</body>
</html>
```

Now that you have a new static web project and an HTML page, it's time to add an input control to the HTML page. The HTML file contains a few `div` tags that make positioning the input control and the suggestions a little easier. The result is the HTML file shown in Listing 2.

Listing 2. The index.html file with controls

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Test web page</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="stylesheet" type="text/css" href="styles/main.css" />
</head>
<body>
<div id="wrapper">
<form>
<div id="search"><input id="searchBox" type="text"
    maxlength="2048" name="criteria" title="Search" onkeyup="update()" /></div>
<div id="suggestions"></div>
</form>
</div>
</body>
</html>
```

At this point, it isn't much different from the blank version. The bulk of the work is done by using JavaScript code in the web page.

Including the Dojo Toolkit

You can include the Dojo Toolkit in one of two ways:

- Use one of the publicly hosted Dojo files locations.
- Download the Dojo Toolkit JavaScript files yourself and include them alongside your own code.

Which method is better depends on your needs.

Using a publicly hosted file

There are a couple of good reasons to use a publicly hosted file:

- Using a content distribution network (CDN) to host the file simply eliminates the need for you to keep track of the file locally.
- It's one less file that you have to deploy.

The main disadvantage of using a publicly hosted file is that you don't have any control if the CDN is unavailable (a situation that is unlikely, but possible). If, for some reason, the CDN is down, your site might not function well. If you have internal web applications that require high uptime, consider maintaining the Dojo script files locally to decrease your application's points of failure.

To link in the Dojo file from one of the CDNs, modify your index.html file to look like the one in Listing 3.

Listing 3. Including the link to the Dojo CDN location

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Test web page</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="stylesheet" type="text/css" href="styles/main.css" />
<script
    src="http://ajax.googleapis.com/ajax/libs/dojo/1.5/dojo/dojo.xd.js"
    type="text/javascript"></script>
</head>
<!-- snipped... -->
</html>
```

Downloading and including the Dojo Toolkit base file

If you want to download and include the Dojo Toolkit base file instead of linking to it, you can download the Dojo JavaScript code from the site linked in Resources. After you download this file, you can import it into your project in Eclipse using **File > Import**. By convention, it's a good idea to create a folder called js or scripts in which you can put not only your JavaScript code files but also the files that you download.

To include the files that you downloaded into your web page, modify index.html to look like Listing 4.

Listing 4. Including a local reference to downloaded dojo.js

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

Now that you have a basic web page, you are ready to add more functions by adding the JavaScript code that calls the service.

Calling a service

Now that you have the index.html file created and the Dojo JavaScript file is linked into your index.html file, it's time to call a simple service to make suggestions to the user.

This article provides two different examples of calling a service and obtaining a result. This first example is a simple service that doesn't necessarily conform to the typical RESTful web service URL conventions. REST is not a standard, but there are ways of crafting the URLs that allow better interoperability between clients and RESTful web services. The second example is an introduction into using the dojox.rpc.Rest method.

Using a simple service

Because the response from the service changes based on the query string, the service does require that you write it in a dynamic web application technology (see

Resources to learn more about dynamic web applications). The example shown in Listing 5 is a simple PHP script that has an array of names. Based on what the user provides, it filters out the list of names and adds them to the Extensible Markup Language (XML) response.

Listing 5. A simple service written in PHP

```
<?php
header("Content-type: text/xml");

$data = array(
    "Bilbo Baggins",
    "Frodo Baggins",
    "Samwise (Sam) Gamgee",
    "Meriadoc (Merry) Brandybuck",
    "Peregrin (Pippin) Took");

$resultXML = '';
$resultsXML .= '<suggestions>';

foreach ($data as $d) {
    $pattern = '/'.$_GET['s'].'/';
    if (preg_match($pattern, $d)) {
        $resultsXML .= '<item>'.$d.'</item>';
    }
}

$resultsXML .= '</suggestions>';

print($resultsXML);
```

You can write a simple script in any other language that returns a similar response to test the JavaScript code.

Making the call to the URL from the Dojo Toolkit could not be easier. Simply set up the arguments for the call, as shown in Listing 6, and pass them to the service.

Listing 6. Setting up the arguments for dojo.xhrGet

```
   var args = {
      url:"mockService.php?s=" + dojo.byId("searchBox").value,
      handleAs:"xml",
      preventCache:true,
      load:function(data) {
          // handle the data...
      },
      error:function(error) { target.innerHTML = "Error:" + error; }
```

The arguments are explained in further detail in Table 1.

Table 1. Arguments for dojo.xhrGet

Argument	Description
url	The arguments include the URL (remember, this URL is not a RESTful URL).
handleAs	One of json, text, or xml. The PHP script responds with XML, so that is used here.

preventCache	Use true if you don't want to cache the data. Caching data leads to faster execution, but it's not desirable if the results are completely dynamic.
load	The callback function to execute when the data is returned by the service.
error	The callback function to execute if an error occurs.

When you have the arguments set up correctly, pass them to the dojo.xhrGet method. The complete code is shown in Listing 7.

Listing 7. The complete JavaScript code

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Test web page</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="stylesheet" type="text/css" href="styles/main.css" />
<script
    src="http://ajax.googleapis.com/ajax/libs/dojo/1.5/dojo/dojo.xd.js"
    type="text/javascript"></script>
<script type="text/javascript">
//<![CDATA[
    // You could move all this code to a JavaScript file and include it...
    dojo.require("dojox.xml.parser");

    function update() {

        if (dojo.byId("searchBox").value.length < 3 )
            return;

        var target = dojo.byId("suggestions");

        var args = {
                url:"mockService.php?s=" + dojo.byId("searchBox").value,
                handleAs:"xml",
                preventCache:true,
                load:function(data) {
                    // handle the data...
                },
                error:function(error) { target.innerHTML = "Error:" + error; }
        };
        var ajaxCall = dojo.xhrGet(args);
    }
//]]>
</script>
</head>
<body>
<div id="wrapper">
<form>
<div id="search"><input id="searchBox" type="text"
    maxlength="2048" name="criteria" title="Search" onkeyup="update()" /></div>
<div id="suggestions"></div>
</form>
</div>
</body>
</html>
```

In addition to calling standard Ajax-style services, you can also use the Dojo Toolkit to call RESTful web services.

Using a RESTful service

Most RESTful web services follow conventions for the way a URL is constructed for a specific type of request, in addition to the HTTP method used for the request. If you build a RESTful service that fits these conventions, you can use the dojo.rpc.Rest object to make the calls for you. The dojo.rpc.Rest object simplifies the traditional service calls even further.

Listing 8 is an example of calling a RESTful web service using the Dojo Toolkit.

Listing 8. Calling a RESTful web service using dojo.rpc.Rest

```
// Calling this access the URL hobbits/1 (see Table 2)
var service = dojox.rpc.Rest("hobbits");
service("1");
```

Table 2 lists examples of proper RESTful URLs and HTTP actions that should go with each type of service. Note that not all browsers support all the HTTP methods, so you should test and verify for your needs. For a massive audience, using GET and POST may be the best option.

Table 2. RESTful conventional URLs

Action	HTTP method	Example URL
Find an object	GET	http://www.example.com/hobbits/1
Find all objects	GET	http://www.example.com/hobbits/
Delete an object	DELETE	http://www.example.com/hobbits/1
Create an object	POST	http://www.example.com/hobbits/
Update a new object	PUT	http://www.example.com/hobbits/1

If you want to create a full implementation of a RESTful web service, frameworks can help guide you to build URLs that conform to the RESTful URL conventions. See Resources to find more information about various frameworks for different languages.

Displaying the results elegantly

Now that you are calling the service with the Dojo code, you can update the contents of the `div` element to include the query results. This allows the users to see the suggestions as they type.

To update the value of the `div` element that contains the suggestions, use the dojo.byId method to get a reference for the `div` by the ID and set the `innerHTML` property as shown in Listing 9. The dojo.byId method is an alias for the traditional JavaScript document.getElementById method.

Listing 9. Updating the contents of the element

```
    // the full load function...
    load:function(data) {
        var rootEl = data.documentElement;
        var resultHTML = "<ul>";
        for (var i = 0; i < rootEl.childNodes.length;i++) {
            resultHTML += "<li>" +
            dojox.xml.parser.textContent(rootEl.childNodes[i]) +
            "</li>";
        }
        resultHTML+="</ul>";
        target.innerHTML = resultHTML;
    },
```

Now that you have the code working, you can view your index.html page in a browser. When you type a value—for example, `Bag`—the suggestions automatically appear in the `div` element. Although not covered in this article, you should use Cascading Style Sheets (CSS) to make the `div` elements flow together nicely in a drop-down box with suggestions based on the user's input.

Handling long-running services

There are two major concerns when handling long-running services from a web browser:

- User experience
- Reliability

User experience

User experience is affected by long-running services if nothing apparent is changed in the browser while the service is executing. If your web page contains a button that makes an Ajax or RESTful service call and displays the results, you should do something with the browser to show the user that something is happening. Otherwise, you run the risk of a user submitting a form multiple times or becoming impatient with your site and leaving.

To combat the user experience issue, implement an animation (the Dojo Toolkit has methods for those) or disable the submit button so the user cannot submit multiple times. With animated visual (for example, a spinning clock), the user has a sense that something is happening and is much more willing to wait for the process to complete.

Reliability

Services that are expected to run for a long time can be problematic to call using Ajax, but there can be valid cases for using them. Examples of long-running services are those that aggregate data, generate documents, or archive files.

In situations such as these, it's not a good practice to simply call the service and wait—especially if the service could process for more than a few seconds. If your

connection is disrupted or if your browser is closed, you cannot necessarily rely on the callback mechanism to fire when the service is complete.

If you have control over the service, consider returning a unique identifier to your callers that provides them with a way of later asking a different service method about the status of the request. Your browser can either put this number into storage in a cookie locally, or the browser and services can work together to persist the number for the user.

This method of calling services allows you to begin the long-running process on the server side. In the browser, you can implement polling to ask the service tier for the status of the request. (You can the Dojo Toolkit's Timer object for this.) This method provides the user with both flexibility and stability.

Summary

Among other things, the Dojo Toolkit enables you to make Ajax calls in your web application to provide RIA functions. Using either one of the CDNs or downloading the Dojo Toolkit files yourself, you can take advantage of prewritten and tested functions.

The Dojo Toolkit provides methods for calling both plain web services using Ajax and RESTful web services. The toolkit allows you to process JSON, XML, and text responses from Ajax services.

Resources

Learn

- Find examples and application programming interface (API) documentation at Dojocampus.org.
- Read "RESTful web Services: The basics" (developerWorks, Nov 2008) to get an introduction to the basic principles of REST.
- Explore CakePHP and REST services and JAX-RS Java API for RESTful web Services.
- Check out "Mastering Grails: RESTful Grails" (developerWorks, Sep 2008) to learn various ways to get Grails to produce XML instead of the usual HTML.
- Visit developerWorks' New to Web development page to start learning about dynamic web applications and more.

Get products and technologies

- Learn more about The Dojo Toolkit.
- Download the Eclipse IDE for JavaScript web Developers.
- Download dojo.js from the Dojo site.
- Download IBM product evaluation versions or explore the online trials in the IBM SOA Sandbox and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Create your My developerWorks profile today and setup a watchlist on Dojo or RESTful web services. Get connected and stay connected with My developerWorks.
- Find other developerWorks members interested in web development.
- Share what you know: Join one of our developerWorks groups focused on web topics.
- Roland Barcia talks about Web 2.0 and middleware in his blog.
- Follow developerWorks' members' shared bookmarks on web topics.
- Get answers quickly: Visit the Web 2.0 Apps forum.
- Get answers quickly: Visit the Ajax forum.

About the author

Nathan A. Good

Nathan A. Good lives in the Twin Cities area of Minnesota. Professionally, he does software development, software architecture, and systems administration. When he's not writing software, he enjoys building PCs and servers, reading about and working with new technologies, and trying to get his friends to make the move to open source software. He's written and co-written many books and articles, including *Professional Red Hat Enterprise Linux 3, Regular Expression Recipes: A Problem-Solution Approach*, and *Foundations of PEAR: Rapid PHP Development.*