Five Ajax best practices

Tips and tools for better Ajax development

Nathan A. Good

Learn five best practices you can apply to your everyday Asynchronous JavaScript + XML (Ajax) development. The article introduces data formats, error handling, and some tools used to develop Rich Internet Applications (RIAs) that use Ajax. Discover how using these practices can help you write more efficient and robust Ajax code.

This article is introduces you to five practices that you can adopt while doing Ajax development in your web applications:

1. Minimize calls
2. Make your data tiny
3. Preload components
4. Handle errors elegantly
5. Use existing tools

These practices help you write JavaScript code that is more robust and executes your Ajax code faster, which is better for your users.

Ajax overview

Ajax is a relatively new term used to describe technologies that have existed for quite some time: JavaScript code, XML, and objects that can make asynchronous calls over HTTP. Ajax is often used to prevent an entire HTML page from being submitted and reloaded, especially when a user performs an action that doesn't require the entire page to be reloaded.

Over the past few years Ajax has been used more widely as the tools for developing Ajax-enabled sites have improved. Use the practices in this article to build better web applications using Ajax and JavaScript code.

Minimize calls

It may sound obvious, but one thing you can do to increase the performance of your web application using Ajax is to minimize the number of calls that you make.
One way that you can minimize the number of calls is to aggregate the calls made into fewer calls. If your data is relatively small (see "Make your data tiny"), your adversary on most networks is latency. Latency is the time that it takes for your browser to actually acquire the connection to the server and services, and it can sometimes be a significant part of a connection. The total latency that users experience depends on a number of factors, including the caching settings of their browsers and DNS clients as well as their physical connections.

There is no easy formula or single code snippet that you can read to learn how to minimize the calls in your web application. However, a simple exercise can demonstrate an effective way of engineering the number of Ajax calls from the client to the server. Consider an example web application where users go to purchase used motorcycles (see Figure 1).

**Figure 1. A simple web page to search for used motorcycles**

First, the user selects the year of the motorcycle. Then, the user selects the make of the motorcycle. Finally, the user selects the motorcycle model. All the while, Ajax is running in the background, updating the drop-down boxes in the web application to filter the lists for the user, thus making it easier to make a selection.

To start the exercise, create a simple diagram with one box for your client and one for your server. Draw lines for the Ajax calls that your browser makes to get the data for the user from the server, as shown in Figure 2.
You can improve the design by combining the calls to get makes and models into one call. Instead of making one call for the makes and then an additional call for the models, you cache the models so that when the user selects the make, your new code simply retrieves the list of available models from the cache. Obtaining the data from a locally-cached resource is much faster than obtaining the same data from a service. By avoiding additional service calls, you avoid the latency of service calls. The new communication looks like Figure 3

Figure 3. Ajax calls after combining the get makes and get models calls

So far, the redesign drops one call from the communication between the browser and the server. You can further reduce the number of calls by using the code shown in Listing 1, where a few key lines can store the data that is retrieved in an array for later lookup.
Listing 1. Storing your data locally in a cache

```javascript
var choices = new Array();
function fillChoiceBoxes(year) {
    // see resources for links to dojo toolkit...
    if (dojo.indexOf(choices, year) == -1) {
        // go get the
        } else {
            // make the ajax call and fill the choices.
            choices[year] = result; // result of ajax call.
        }
    // calling a function to fill the values...
    fillSelect(dojo.byId('makes'), choices[year]);
}
```

If the user is considering two different models and is going back and forth, the web application uses the locally-cached data instead of making the additional service call. Cache only data that is static—at least for the time that the user's session lasts. Don't introduce a host of issues by caching data that shouldn't be cached.

As demonstrated by this example, you can minimize calls by reducing the number of round trips from the client to the server and caching data where possible.

**Make your data tiny**

To increase the performance of dealing with data, make the data that you transmit from the server to the client as small as possible. To effectively do this, you must have control over the service layer to the point where you can dictate the formatting of messages sent from the services to the client.

XML is a common format for messages between the client and the server for many good reasons. One of those reasons is there is no shortage of libraries or frameworks that provide XML serialization for you.

However, XML is a verbose format for messages when compared to JavaScript Serialized Object Notation (JSON), which is a more concise message format. There are many libraries currently available to enable you to easily build messages to transport your objects in JSON from the services to the client.

Many client-side libraries, such as the Dojo Toolkit, allow you to define which transport format your services use. If your service responses use JSON, you can use the same client objects by providing a parameter.

Consider the code in Listing 2, which demonstrates the representation of a motorcycle object using XML.
Listing 2. The motorcycle data using XML

```xml
<motorcycle>
  <year>2010</year>
  <make>Motocool</make>
  <model>Uberfast</model>
</motorcycle>
```

Now consider Listing 3, which shows the same data using JSON. Note that is almost 25% smaller (if the whitespace is removed).

Listing 3. The motorcycle data using JSON

```json
{
  "motorcycle" : {
    "year" : "2010",
    "make" : "Motocool",
    "model" : "Uberfast"
  }
}
```

Because the data is smaller, not only does it take less time to transmit from the services to the client (or vice versa), but it also may take less time to parse because the string is smaller.

Design the data that you are transporting to contain as few characters as possible.

Preload components

You can take advantage of browser caching by loading components such as JavaScript files and images that you use with your Ajax calls. It's important to note that preloading JavaScript files and images provides a benefit only to those users who have caching turned on, but that is a large percentage of users.

To preload external JavaScript files, include the JavaScript file on a page earlier in the workflow, but only do this if that page is smaller and has fewer resources than the page you're trying to optimize. For example, preloading is useful when you have a relatively light page that introduces a workflow to the user. Consider the motorcycle purchasing example from the Minimize Calls section. You can preload the heavier JavaScript code that contains all of the Ajax code for the page containing the drop-down boxes in an earlier page in the flow.

If you update images as a result of making Ajax calls, preloading images provides a good benefit. With the images preloaded, users don't have to wait for the browser to retrieve the image when they hover their mouse over an element, make a choice from a drop-down box, or click a button. Even though the Ajax occurs asynchronously, the image itself takes some time to transmit from the server to the client and doesn't appear in the user's browser until the image is completely downloaded.

In the example shown in Listing 4, the images used when a user selects a make of motorcycle from the list are preloaded using standard JavaScript code.
Listing 4. Preloading images using standard JavaScript code

```html
<html>
<head><title>Preload example</title></head>
<body>
<!-- web page... -->
<script type="text/javascript" language="javascript">
  var img = new Image();
  img.src = "http://path/to/motocool.jpg";
</script>
</body>
</html>
```

JavaScript placement is important when preloading images for the page. You do not want to affect the loading of the web page by putting JavaScript code in the HTML where it might slow the page loading down. As a general rule, you should put JavaScript code that is inside `<script>` elements last on your HTML page because the browser is relatively limited regarding how many resources it can download at one time. Putting your scripts toward the end of your HTML page, if possible, helps your browser load images and other resources faster.

In HTML 5, you can use the new `async` attribute for the `<script>` tag. This tells your browser to run the JavaScript code asynchronously so that it can execute while other things are happening in your web page.

**Handle errors elegantly**

Every function you define in your JavaScript code should assume hostile input because code that is written defensively performs better than handling errors using `try... catch` statements. For instance, if you need to do a calculation with a JavaScript function based on the user’s input, check the input before doing the calculation, as shown in Listing 5.

Listing 5. Checking user input

```javascript
function calculateDistance(source, dest) {
  if (!isNaN(source) && !isNaN(dest)) {
    dojo.byId("errors").innerHTML = "Please provide a valid number. ";
  }
}
```

Even if you code defensively, it’s a good idea to use `try... catch` statements and error callbacks where applicable. Listing 6 demonstrates the use of a `try... catch` statement in JavaScript code to catch an error.

Listing 6. Using a `try... catch` statement to handle errors

```javascript
function calculateDistance(source, dest) {
  try {
    // do some calculations...
  } catch (error) {
    dojo.byId("errors").innerHTML = "An error occurred while adding the numbers";
  }
}
```

Listing 7 demonstrates the use of an error callback while calling the `xhrGet()` method provided in the Dojo Toolkit. The error argument is optional, so it’s tempting to skip defining an error handler.
Listing 7. Using an error callback with \texttt{xhrGet( )}

```javascript
var args = {
  url: "/moin_static185/js/dojo/trunk/dojo/..dojo/NoSuchFile",
  handleAs: "text",
  preventCache: true,
  load: function(data) {
    // do something when successful...
  },
  error: function(error) {
    dojo.byId("errors").innerHTML = "An error occurred while getting the data..";
  }
}
var ajx = dojo.xhrGet(args);
```

How to handle errors on your web page can be as much of a business problem as it is a technical problem. Make sure you ask your clients what message they want users to see if something goes wrong because any message you show the users can have an impact on the business. Your clients can help provide meaningful default behavior, where applicable, if an exception occurs.

Finally, don't display the description of the error in a JavaScript alert as shown in Listing 8. Your users aren't software engineers, so chances are anything that you show your users in an alert box is meaningless information to them. In addition to the inconvenience of providing messages that users can probably do nothing about, alert boxes require users to make the extra action of dismissing the box to get back to the web page.

Listing 8. Avoid JavaScript alert boxes in error handling

```javascript
function calculateDistance(source, dest) {
  try {
    // do some calculations...
  } catch (error) {
    // Bad:
    // alert(error.message);
    // Better:
    dojo.byId("errors").innerHTML = "An error occurred while calculating data...";
  }
}
```

Use existing tools

Finally, as a best practice, avoid the Not Invented Here (NIH) syndrome as much as possible. By using existing tools (frameworks and platforms), you take advantage of their resources. Most mature, commonly used tools have been thoroughly tested for performance and cross-browser compatibility. They also have more features than you might have time to implement on your own project.

There are many good tools that provide methods for making Ajax calls—in addition to supporting many other functions and features, such as animation. Some of those tools are listed in Table 1.

**Table 1. JavaScript tools that provide methods for making Ajax calls**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
</table>

Five Ajax best practices
Dojo Toolkit

The Dojo Toolkit is a suite of JavaScript tools available for free. It provides methods for making Ajax calls to normal web pages as well as Representational State Transfer (REST) services. The Dojo Toolkit methods support XML, JSON, and plain text for message formats.

Google Web Toolkit (GWT) Designer

Google recently acquired Instantiations Developer Tools and relaunched it as a suite of free products. One of them is GWT Designer, which you can install on top of an existing installation of Eclipse. You can use the designer to help you build interfaces using the GWT. The GWT lets you build complex web applications that use Ajax, which enables web applications to be nearly as complex as native applications. Similar to Rich Ajax Platform (RAP), GWT is more than a JavaScript framework, but rather a Java™-based toolkit that compiles to Ajax-enabled HTML.

jQuery

jQuery is another JavaScript library that provides a full complement of Ajax functions. jQuery also supports different message formats and other Ajax-based methods, such as getScript(), which downloads a JavaScript file (a derivation of the Preload Components best practice) and executes it.

Prototype

Prototype is also a JavaScript framework that you can use to make easy Ajax calls. Using methods such as the Ajax.PeriodicalUpdater, you can update values in your Ajax page on a periodic basis, which lets you implement controls such as progress bars or other status for long-running service processes.

Rich Ajax Platform (RAP)

Unlike the other frameworks listed in this table, RAP is a full platform that enables you to build better Ajax-enabled sites using the Eclipse integrated development environment (IDE) and Java (not JavaScript) code, similar to building a Swing or Standard Widget Toolkit (SWT) application. For Java programmers who don’t want to deal with the complexity of HTML, CSS, and JavaScript code, a platform tool such as RAP can be a good alternative.

The RAP documentation warns not to install it as a plug-in onto an existing installation of Eclipse. Rather, download the Eclipse for Rich Client Platform (RCP) and RAP Developers bundle from the Eclipse site (see Related topics) and install it into a separate location. A cheat sheet demonstrates how to import a sample project.

Summary

Using Ajax in your web applications can provide your users with smoother web application interfaces. Ajax already provides some improvement over transmitting full HTML pages, but adopting the best practices in this article can help you begin building better Ajax applications.
Related topics

- Read examples and API documentation for the Dojo Toolkit at Dojocampus.org.
- Learn more about the Dojo Toolkit.
- Get GWT Designer from the GWT site.
- Explore jQuery.
- Check out Prototype.
- Download Eclipse IDE for RCP and RAP Developers.
- 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®.

© Copyright IBM Corporation 2010
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)