Mastering Ajax, Part 11: JSON on the server side

Responding to and replying with JSON in your server-side scripts and programs

Skill Level: Advanced

Brett McLaughlin (brett@newInstance.com)
Author and Editor
O'Reilly Media Inc.

14 Aug 2007

In the last article, you learned how to take an object in JavaScript and convert it into a JSON representation. That format is an easy one to use for sending (and receiving) data that maps to objects, or even arrays of objects. In this final article of the series, you'll learn how to handle data sent to a server in the JSON format and how to reply to scripts using the same format.

JSON's real value

As discussed in the previous article in this series, JSON is a useful format for Ajax applications because it allows you to convert between JavaScript objects and string values quickly. Because Ajax apps are best suited to send plain text to server-side programs and receive plain text in return, an API that generates text is almost always preferable over an API that doesn't; further, JSON allows you to work with native JavaScript objects and not worry about how those objects will be represented.

The developerWorks Ajax resource center
Check out the Ajax Resource Center, your one-stop shop for information on the Ajax programming model, including articles and tutorials, discussion forums, blogs, wikis, events, and news. If it's happening, it's covered here.

XML provides the same textual benefits, but the APIs for converting JavaScript objects into XML (and there are a few) aren't as mature as the JSON APIs; you'll

sometimes have to take care in creating and working with your JavaScript objects to ensure that you're doing things that can work with the XML conversion API you've chosen. The same isn't true with JSON: it handles pretty much every conceivable object type and simply gives you back a nice JSON data representation.

So the biggest value of JSON is that you can work with JavaScript *as JavaScript*, not as a data-format language. All the things you learn about using JavaScript objects can be applied to your code without worrying about how those objects will be converted to text. Then, you make a simple JSON method call:

```
String myObjectInJSON =
myObject.toJSONString();
```

and you're ready to send the resulting text onto a server.

Getting JSON to the server

Sending JSON to the server isn't particularly difficult, but it is crucial, and you still have a few choices to make. However, once you've already chosen to use JSON, the choices are a lot simpler and quite a bit more limited, so there's not as much to think or worry about. The bottom line is you just need to get your JSON string to the server, preferably as quickly and as simply as possible.

Sending JSON in name/value pairs via GET

The easiest way to get your JSON data to the server is to convert it to text, and send it as the value of a name/value pair. Remember, your JSON-formatted data is just one fairly long object, and probably looks something like Listing 1:

Listing 1. A simple JavaScript object in JSON format

```
var people =  { "programmers": [    { "firstName": "Brett", "lastName":"McLaughlin",
"email": "brett@newInstance.com" },    { "firstName": "Jason", "lastName":"Hunter",
"email": "jason@servlets.com" },    { "firstName": "Elliotte", "lastName":"Harold",
"email": "elharo@macfaq.com" }   ],  "authors": [    { "firstName": "Isaac",
"lastName": "Asimov", "genre": "science fiction" },    { "firstName": "Tad",
"lastName": "Williams", "genre": "fantasy" },    { "firstName": "Frank",
"lastName": "Peretti", "genre": "christian fiction" }   ],  "musicians": [
{ "firstName": "Eric", "lastName": "Clapton", "instrument": "guitar" },
{ "firstName": "Sergei", "lastName": "Rachmaninoff", "instrument": "piano" }   ]  }
```

So you could send this to a server-side script as a name/value pair like this:

```
var url = "organizePeople.php?people=" + people.toJSONString();
xmlHttp.open("GET", url, true);
xmlHttp.onreadystatechange = updatePage;
xmlHttp.send(null);
```

This looks good, but there's one problem: you've got spaces and all sorts of characters in your JSON data string that a Web browser might try and interpret. To ensure that these characters don't mess things up on the server (or in transmission of your data to the server), add in the JavaScript `escape()` function, like this:

```
var url = "organizePeople.php?people=" + escape(people.toJSONString());
request.open("GET", url, true);
request.onreadystatechange = updatePage;
request.send(null);
```

This function handles whitespace, slashes, and anything else that might trip up browsers and converts them to Web-safe characters (for example, an empty space is converted to `%20`, which browsers don't treat as a space but instead pass on to a server unchanged). Then servers will (generally automatically) convert these back to what they are supposed to be after transmission.

The downside to this approach is twofold:

- You're sending potentially huge chunks of data using a GET request, which has a length limitation on the URL string. It's a big length, granted, but you never know how long an object's JSON string representation could get, especially if you're using pretty complex objects.
- You're sending all your data across the network as clear text, which is about as unsecure as you can manage to get when it comes to sending data.

To be clear, these are both limitations of GET requests, rather than anything related to JSON data specifically. However, they're very real concerns when you're sending more than just a user's first or last name, or maybe selections on a form. Once you start to deal with anything that might be even remotely confidential, or extremely lengthy, then you should look at using POST requests.

POSTing JSON data

When you decide you want to move to using a POST request for sending your JSON data to the server, you don't have to make a lot of code changes. Here's what you'd want:

```
var url = "organizePeople.php?timeStamp=" + new Date().getTime();
request.open("POST", url, true);
request.onreadystatechange = updatePage;
request.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
request.send(people.toJSONString());
```

Most of this code should be familiar to you from " *Mastering Ajax, Part 3*: Advanced requests and responses in Ajax," which focuses on sending POST requests. The request is opened using POST instead of GET, and the Content-Type header is set to let the server know what sort of data it should expect. In this case, that's `application/x-www-form-urlencoded`, which lets the server know you're just sending across text like it would get from a normal HTML form.

One other quick note is that the URL has the current time appended to it. That ensures that the request isn't cached after it's sent the first time but is recreated and resent each time this method is called; the URL will be subtly different because of the changing time stamp. This is a common trick to ensure that POSTing to a script actually repeatedly generates a new request each time, and the Web browser doesn't try and cache responses from the server.

JSON is just text

Whether you use GET or POST, the big deal here is that JSON is ultimately just text data. You can easily manipulate it and get it to the server because it doesn't require any special encoding, and every server-side script in the world can handle text data. If JSON were a binary format or had some strange textual encoding, that wouldn't be the case; as it is, though, JSON is just normal text data -- like a script would receive from a form submission, as you saw in the POST section and Content-Type header -- so you don't have a lot to worry about when sending it to a server.

Interpreting JSON on the server

Once you've written your client-side JavaScript code, allowed your users to interact with your Web forms and pages, and gathered information that you need to send on to a server-side program for processing, you're to the point where the server becomes the major player in your application (and probably well into what we all think of as "Ajax applications," assuming you've made the call to the server-side program you're using asynchronous). It's here where the choices you made on the client, like using JavaScript objects and then converting them to JSON strings, must be matched by decisions on the server, like which API to use to decode your JSON data.

The JSON two-step

Working with JSON on the server side is essentially a two-step process, no matter what language you're using on the server:

1. Find a JSON parser/toolkit/helper API for the language you're using to write your server-side programs.

2. Use the JSON parser/toolkit/helper API to take the request data from the client and turn it into something your script can understand.

That's really all that there is to it. Let's take each step in a little more detail.

Find a JSON parser

The best resource for finding a JSON parser or toolkit is the JSON Web site (see Resources for a link). In addition to learning a lot about the format itself, this page has links to JSON tools and parsers for everything from ASP to Erlang to Pike to Ruby. Simply find the language your scripts are written in and download a toolkit. Drop that or expand that or install that (there's a lot of variability when you could be using C#, or PHP, or Lisp on the server) so that your scripts and programs on the server can use the toolkit.

For example, if you're using PHP, you could simply upgrade to PHP 5.2 and be done with it; this recent version of PHP includes the JSON extension by default. In fact, that's almost certainly the best way to handle JSON when you're working with PHP. If you're using Java servlets, the `org.json` package, hosted at json.org, is a simple choice. In this case, you'd download json.zip from the JSON Web site and add the included source files to your project's build directory. Compile these files, and you're ready to go. Similar steps hold true for the other languages supported; your own expertise using the language you prefer is the best guide.

Using your JSON parser

Once you've got the resources available to your program, it's just a matter of finding the right method to call. For example, suppose you were using the JSON-PHP module with PHP:

```
// This is just a code fragment from a larger PHP server-side script
require_once('JSON.php');

$json = new Services_JSON();

// accept POST data and decode it
$value = $json->decode($GLOBALS['HTTP_RAW_POST_DATA']);

// Now work with value as raw PHP
```

With that, you've got all the data -- array format, multiple rows, single values, whatever you stuffed into your JSON data structure -- into a native PHP format, in the `$value` variable.

If you were using the `org.json` package in a servlet, you'd use something like this:

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
  throws ServletException, IOException {

  StringBuffer jb = new StringBuffer();
  String line = null;
  try {
    BufferedReader reader = request.getReader();
    while ((line = reader.readLine()) != null)
      jb.append(line);
  } catch (Exception e) { //report an error }

  try {
    JSONObject jsonObject = new JSONObject(jb.toString());
  } catch (ParseException e) {
    // crash and burn
    throw new IOException("Error parsing JSON request string");
  }

  // Work with the data using methods like...
  // int someInt = jsonObject.getInt("intParamName");
  // String someString = jsonObject.getString("stringParamName");
  // JSONObject nestedObj = jsonObject.getJSONObject("nestedObjName");
  // JSONArray arr = jsonObject.getJSONArray("arrayParamName");
  // etc...
}
```

You can refer to the `org.json` package documentation (links in the Resources section) for more details. *(Note: If you want more detail on the org.json or other JSON toolkits, e-mail me and let me know. Hearing from you helps me decide what to write about!)*.

In conclusion

You should have a handle now on how to deal with JSON from a technical standpoint on the server-side. This article and Part 10 of the series, beyond just being technical assists, should also convince you that JSON is a pretty flexible, powerful data format. Even if it's not what you use on every application, good Ajax and JavaScript programmers should have JSON in their toolbox and be able to select it for use any time a need arises.

I'd certainly like to hear more about your own experiences with JSON and what languages you're finding work well -- and not so well -- on the server for dealing with JSON data. Hop on the Java and XML newsgroup (links in Resources) and let me know. Enjoy JSON and the power of textual data formats.

Resources

Learn

- See all of the articles in the *Mastering Ajax* series. Part 3 focuses on sending and receiving advanced requests and responses in Ajax.
- The best resource for finding a JSON parser or toolkit is the JSON Web site, the virtual hub for all things JSON.
- Read the JavaDoc for the org.json package.
- xml.com is one of the easiest to understand online resources for everything XML. Start there if you're not already an experienced XML programmer.
- "Ajax for Java developers: Build dynamic Java applications" (Philip McCarthy, developerWorks, September 2005) is a look at Ajax from the server side, using a Java perspective.
- "Ajax for Java developers: Java object serialization for Ajax" (Philip McCarthy, developerWorks, October 2005) examines how objects can be sent over the network, and interact with Ajax, from a Java perspective.
- "Call SOAP Web services with Ajax, Part 1: Build the Web services client" (James Snell, developerWorks, October 2005) is a fairly advanced article on integrating Ajax with existing SOAP-based Web services.
- The DOM home page at the World Wide Web Consortium is the starting place for all things DOM-related.
- The DOM Level 3 Core Specification defines the core Document Object Model, from the available types and properties to the usage of the DOM from various languages.
- The ECMAScript language bindings for DOM are of great interest to any JavaScript programmer wanting to use the DOM from their code.
- "Ajax: A New Approach to Web Applications" is the article that coined the Ajax moniker and is required reading for all Ajax developers.
- *Head Rush Ajax* (Brett McLaughlin, O'Reilly Media, Inc.): Takes the ideas in this article and loads them into your brain, Head First style.
- *Java and XML, Second Edition* (Brett McLaughlin, O'Reilly Media, Inc.): Includes Brett's discussion of XHTML and XML transformations.
- *JavaScript: The Definitive Guide* (David Flanagan, O'Reilly Media, Inc.): Includes extensive instruction on working with JavaScript and dynamic Web pages; the upcoming edition adds two chapters on Ajax.
- *Head First HTML with CSS & XHTML* (Elizabeth and Eric Freeman, O'Reilly

Media, Inc.): Learn more about standardized HTML and XHTML and how CSS can be applied to HTML.

- "New to XML" page: Ready to learn about XML, but don't know where to start? Visit the XML zone's updated resource central for XML.
- The Ajax Resource Center here on developerWorks is your one-stop shop for all things Ajax. Browse the resources there and get started developing with Ajax today.

Discuss

- XML and Java Technology forum: Discuss your own experiences with JSON and the languages that you find work (or not) with Brett.
- developerWorks blogs: Get involved in the developerWorks community.

About the author

Brett McLaughlin

Brett McLaughlin has worked in computers since the Logo days. (Remember the little triangle?) In recent years, he's become one of the most well-known authors and programmers in the Java and XML communities. He's worked for Nextel Communications, implementing complex enterprise systems; at Lutris Technologies, actually writing application servers; and most recently at O'Reilly Media, Inc., where he continues to write and edit books that matter. Brett's upcoming book, *Head Rush Ajax* , brings the award-winning and innovative Head First approach to Ajax. His last book, *Java 1.5 Tiger: A Developer's Notebook* , was the first book available on the newest version of Java technology. And his classic *Java and XML* remains one of the definitive works on using XML technologies in the Java language.