Mastering Ajax, Part 10: Using JSON for data transfer

Work natively with JavaScript objects

Skill Level: Introductory

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

27 Mar 2007

Plain text and XML are both data formats that you can use for sending and receiving information in your asynchronous applications. This installment of Mastering Ajax looks at another useful data format, JavaScript Object Notation (JSON), and how it makes moving data and objects around in your applications easier.

If you've been reading this series for long, you've received an extensive education in data formats. Earlier articles examined how plain text and simple name/value pairs work great in many asynchronous applications. You can assemble data that looks like this:

```
firstName=Brett&lastName=McLaughlin&email=brett@newInstance.com
```

There's no need to do anything more. In fact, Web veterans will recognize this as the format for information sent through GET requests.

Then the series looked at XML. Obviously, XML has received its fair share of press (both negative and positive), and it's no surprise that it shows up in Ajax applications. You can review earlier articles in this series to see how XML provides an alternative data format:

```
<request>
  <firstName>Brett</firstName>
```

```
   <lastName>McLaughlin</lastName>
   <email>brett@newInstance.com</email>
 </request>
```

This is the same data you saw above, but this time it's stored in an XML format. There's nothing remarkable going on here; it's just a different data format that allows you to use XML instead of plain text and name/value pairs.

This article takes on yet another data format, called *JavaScript Object Notation,* or *JSON.* JSON looks a bit like what you've seen already, and a bit like nothing you've ever seen before. It will give you another choice, and having choices is a good thing.

The power to choose

Before digging into the details of the JSON format, you should understand why it's worth spending another couple of articles on another data format (yes, the next article in this series will deal with JSON as well), especially when you already know how to work with both XML and name/value pairs in plain text. In a nutshell, though, it's simple: The more choices you have, and the more options you have for any problem, the better your chances are of finding the *best* solution to a problem, rather than just *a* solution.

Recapping name/value pairs and XML

This series has already touched a good bit on those situations in which name/value pairs make sense, as well as those in which XML might be a better choice. To recap quickly, your first thought should always be to use name/value pairs. Doing so is almost always the simplest solution to the problems in most asynchronous applications, and it doesn't require anything but a very basic knowledge of JavaScript.

You really shouldn't even worry about using an alternate data format unless you have some sort of constraint that moves you towards XML. Obviously, if you're sending data to a server-side program that requires XML-formatted input, then you will want to use XML as your data format. In most cases, though, XML turns out to be a better choice for the servers that need to send your application multiple pieces of information; in other words, XML is more commonly used for responses *to* Ajax applications, rather than requests *from* Ajax applications.

Adding JSON to the mix

When you're using either name/value pairs or XML, you're essentially taking data from the application using JavaScript and stuffing it into a data format. In these cases, JavaScript is functioning largely as a data manipulation language, moving and manipulating data from Web forms and putting it into a format that can be easily sent to a server-side program.

However, there are times when you'll use JavaScript as more than just a format language. In these cases, you're actually using objects in the JavaScript language to represent data, and going beyond just shuttling data from Web forms into requests. In these situations, it's *more* work to extract data from the objects in JavaScript and then stuff that data into name/value pairs or XML. This is where JSON shines: It allows you to easily turn JavaScript objects into data that can be sent as part of a request (synchronous or asynchronous).

The bottom line is that JSON isn't some sort of magic bullet; it is, however, a great option for some very specific situations. Rather than assuming that you won't ever encounter those situations, read this article and the next in the series to learn about JSON, so you'll be equipped when you do run into those types of problems.

JSON basics

At its simplest, JSON allows you to transform a set of data represented in a JavaScript object into a string that you can easily pass from one function to another, or -- in the case of asynchronous applications -- from a Web client to a server-side program. The string looks a little odd (you'll see some examples in just a moment), but it's easily interpreted by JavaScript, and JSON allows you to represent structures more complex than name/value pairs. For instance, you can represent arrays and complex objects, rather than just simple lists of keys and values.

Simple JSON examples

At its very simplest, you can represent what is essentially a name/value pair in JSON like this:

```
{ "firstName": "Brett" }
```

This is pretty basic and actually takes up more space than the equivalent name/value pair in clear text:

```
firstName=Brett
```

However, JSON illustrates its value when you start to string together multiple name/value pairs. First, you can create what is essentially a *record* of data, with multiple name/value pairs, like this:

```
{ "firstName": "Brett", "lastName":"McLaughlin", "email": "brett@newInstance.com" }
```

There's still not much advantage here over name/value pairs in terms of syntax, but

in this case JSON is significantly easier to use, and has some readability advantages. For example, it's clear that all three of the values above are part of the same record; the brackets establish that the values have some connection.

Arrays of values

When you need to represent a set of values, JSON starts to be not only more readable, but less verbose. Say, for example, that you want to have a list of people. In XML, you'd be stuck with lots of opening and closing tags, and if you were using typical name/value pairs -- the kind we've looked at in earlier articles in this series -- then you'd have to come up with a proprietary data format, or perhaps modify the key names to something like `person1-firstName`.

With JSON, you can simply group multiple bracketed records:

```
{ "people": [
  { "firstName": "Brett", "lastName":"McLaughlin", "email": "brett@newInstance.com" },
  { "firstName": "Jason", "lastName":"Hunter", "email": "jason@servlets.com" },
  { "firstName": "Elliotte", "lastName":"Harold", "email": "elharo@macfaq.com" }
]}
```

This isn't too hard to understand. In this case, there's a single variable, named `people`, and the value is the array containing three items, each of which is a person record with a first name, a last name, and an e-mail address. The example above illustrates how you can throw records together, and also group the items into a single value with brackets around it. Of course, you could use the same syntax, but have multiple values (each with multiple records):

```
{ "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" }
 ]
}
```

The most obvious thing to note here is your ability to represent multiple values that each in turn have multiple values. What you should also notice, though, is that the actual name/value pairs in the records change across the different main entries (programmers, authors, and musicians). JSON is completely dynamic and lets you change the way you represent data in the middle of your JSON structure.

To put it another way, there is no predefined set of constraints that you need to follow when working with JSON-formatted data. So you can change how you represent things, or even represent the same thing in different ways, all within the same data structure.

Using JSON in JavaScript

After you've got a handle on the JSON format, it's simple to use it within JavaScript. JSON is a native JavaScript format, meaning simply that you don't need any special API or toolkit to work with JSON data within JavaScript.

Assigning JSON data to a variable

For example, you can simply create a new JavaScript variable and then directly assign a string of JSON-formatted data to it:

```
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" }
   ]
  }
```

There's nothing complicated going on here; now `people` contains the JSON formatted data we've been looking at throughout this article. However, this doesn't do much, as the data still isn't in a format that is obviously useful.

Accessing the data

While it might not be obvious, that lengthy string above is just an array, and once you've got that array in a JavaScript variable, you can access it easily. In fact, you can simply separate the array with period delimiters. So, to access the last name of the first entry of the programmers list, you would use code like this in your JavaScript:

```
people.programmers[0].lastName;
```

Take note that the indexing is zero-based. So this begins with the data in the

`people` variable; it then moves to the item called `programmers` and pulls off the first record (`[0]`); finally, it accesses the value for the `lastName` key. The result is the string value "McLaughlin".

Here are a few more examples using the same variable.

```
people.authors[1].genre                 // Value is "fantasy"
people.musicians[3].lastName         // Undefined. This refers to the fourth entry,
 and there isn't one
people.programmers.[2].firstName // Value is "Elliotte"
```

With this little bit of syntax, you can work with any variety of JSON-formatted data, all without any extra JavaScript toolkits or APIs.

Modifying JSON data

Just as you can access data with the dot and bracket notation shown above, you can easily modify data in the same way:

```
people.musicians[1].lastName = "Rachmaninov";
```

That's all you need to do to change data in a variable once you've converted from a string to JavaScript objects.

Converting back to strings

Of course, all that data modification isn't of much value if you can't easily convert back into the textual format mentioned in this article. That's also trivial in JavaScript:

```
String newJSONtext = people.toJSONString();
```

That's all there is to it! Now you've got a string of text that you can use anywhere you like -- you could use it as the request string in an Ajax application, for instance.

Perhaps even more importantly, you can convert *any* JavaScript object into JSON text. You don't have to work only with variables that were originally assigned a JSON-formatted string. To transform an object named `myObject`, you would simply issue the same sort of command:

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

This is the biggest difference between JSON and other data formats this series has

explored. With JSON, you call a simple function and get your data, formatted and ready for use. With other data formats, it's your job to handle the conversion between the raw data and the formatted data. Even when using an API like the Document Object Model that provides a function that converts from its own data structure into text, you've got to learn the API and use that API's objects, rather than native JavaScript objects and syntax.

The end result is that when you're already working with lots of JavaScript objects, JSON is almost certainly a good bet for you to easily convert your data into a format that is simple to send in your requests to server-side programs.

Conclusion

This series has spent a lot of time looking at data formats, mostly because your asynchronous applications are ultimately almost all about data. If you have a variety of tools and techniques that will let you send and receive all kinds of data -- and do it in the best, most efficient manner for each data type -- then you're well on your way to being an Ajax pro. Add JSON to what you've already got available in using XML and plain text, and you're equipped for even more complex data structures in JavaScript.

The next article in this series goes beyond just sending data, and dives into how server-side programs can receive -- and deal with -- JSON-formatted data. I'll also look at how server-side programs can send back data in the JSON format across scripts and server-side components, making it possible to mix and match XML, plain text, and JSON requests and responses. The goal is flexibility, and you're well on your way to being able to put all these tools together, in almost any combination you can imagine.

Resources

Learn

- Mastering Ajax: Read the previous articles in this series.
- JSON.org: Visit the JSON Web site for more on this data format, including links to several JSON API implementations.
- A number of developerWorks articles cover more advanced JSON topics:
  - "Convert XML to JSON in PHP," Senthil Nathan, Edward J. Pring, and John Morar (January 2007)
  - "Cache in with JSON," Bakul L. Patel (October 2006)
  - "Generate JSON from XML to use with Ajax," Jack D. Herrington (September 2006)
- developerWorks XML zone: Learn all about XML at the developerWorks XML zone.
- xml.com: Also a good resource for everything XML.
- "Build dynamic Java applications," Philip McCarthy (developerWorks, September 2005): A look at Ajax from the server side, using a Java perspective.
- "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," James Snell (developerWorks, October 2005): A fairly advanced article on integrating Ajax with existing SOAP-based Web services.
- The DOM Home Page at the World Wide Web Consortium: 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: Of great interest to any JavaScript programmers wanting to use the DOM from their code.
- "Ajax: A new approach to Web applications," Jesse James Garrett (*Adaptive Path,* February 2005): The article that coined the Ajax moniker. Required reading for all Ajax developers.

Get products and technologies

- *Head Rush Ajax,* Brett McLaughlin (O'Reilly Media, 2006): 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., 2001): Includes Brett's discussion of XHTML and XML transformations.
- *JavaScript: The Definitive Guide,* David Flanagan (O'Reilly Media, Inc., 2001): 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., 2005): Learn more about standardized HTML and XHTML, and how CSS can be applied to HTML.
- IBM trial software: Build your next development project with software available for download directly from developerWorks.

Discuss

- developerWorks blogs: Get involved in the developerWorks community.

About the author

Brett McLaughlin

Brett McLaughlin has been working in computers since the Logo days (remember the little triangle?). He currently specializes in building application infrastructure using Java and Java-related technologies. He has spent the last several years implementing these infrastructures at Nextel Communications and Allegiance Telecom, Inc. Brett is one of the co-founders of the Java Apache project, Turbine, which builds a reusable component architecture for Web application development using Java servlets. He is also a contributor of the EJBoss project, an open source EJB application server, and Cocoon, an open source XML Web-publishing engine. Contact Brett at brett@oreilly.com