Armed with AJAX

By Russ McGuire - russ.mcguire@gmail.com

Last month we discussed client-side programming, especially JavaScript. A few months ago we discussed web services. We've also spent most of our time studying different aspects of server-side programming. This month, we're going to tie all of those concepts together in looking at a very powerful programming technique known as AJAX.

What is AJAX?

AJAX is an acronym standing for Asynchronous JavaScript and XML. Let's look at each part of this name to understand what's involved:

Asynchronous: Asynchronous literally means Not Synchronous. A synchronous system works in a fixed, ordered way. One thing happens, followed by another, in pre-determined fashion. Therefore; asynchronous means that the program can operate with independent actions not waiting to go in a specific order.

JavaScript: We studied JavaScript at length in last month's column. JavaScript is a client-side programming language that runs inside the web browser. (Note that the capabilities represented by AJAX can actually be achieved with any client-side programming language – not just JavaScript.)

XML: eXtensible Markup Language is a set of rules for encoding a document in machine-readable form. XML is extensible, meaning that specific types of documents can be defined using the structure of XML. These document types are defined with a Document Type Definition (DTD). XML is organized as markup and content, with markup tags contained in < brackets >. If you're familiar with HTML, this is probably starting to sound familiar, and in fact HTML has become a subset of XML defined by a DTD. The first line in each HTML page at ccmag.com is: <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> which tells us that this website is using XHTML 1.0 Transitional which is defined in a DTD at the w3.org website.

The important thing about XML as it relates to AJAX is that it provides a language for two programs to talk to each other. (Note that the capabilities represented by AJAX can actually be achieved without using XML – as long as the messages can be understood by each program.)

So, what does it mean when we put all these things together?

It means that there are programs running inside a web page which are communicating in an asynchronous manner with server-side programs. This means that the web page can change dynamically, probably based on user interaction, and the content of the page can be updated with information coming from the server.

Why use AJAX?

A popular example of AJAX is Google's Gmail service. Gmail uses AJAX to mimic the capabilities of a standalone client program (like Microsoft Outlook) within a web application. When you click on Inbox, a portion of the page redraws listing the messages in the inbox, and when you click on an individual message, a portion of the page redraws displaying the message.

Technically, behind the scenes, Gmail is using AJAX to fetch the list of messages or the body of the message and display it. But to the user, the end result is simplicity and ease of use. It's comfortable and natural, not like most web interfaces. That's the beauty of AJAX, and why more and more web sites are using it.

How to deploy AJAX

AJAX requires you to write two programs – the client side (probably in JavaScript) and the server side (in our case, we use PHP).

Writing the client side program is easiest using a framework like jQuery. JQuery includes a full suite of methods and functions to simplify implementing AJAX. The simplest is the .load() method which fills the specified page element with content retrieved from the specified url. This method is good for loading a page with static data – for example in implementing a tabbed interface or an interface for stepping through multiple pages of content.

A more likely implementation would use a function like .get() or .post() to pass data to the server as if a form had been submitted. The returned data can then be copied into an element on the web page. This can be used, for example, to load a specific Bible passage into a section of the web page when the user makes a selection without having to reload the entire web page.

An even more sophisticated approach is to use the .getJSON() function to retrieve back from the server structured data that can be used to manipulate elements on the page. For example, I've used this approach to populate known churches in a pull-down list (<SELECT> element) based on a city and state input by the user. (JSON stands for JavaScript Object Notation and is a simpler alternative to XML.)

Writing the server side program is pretty straightforward. In general, it works just like any other web server program, simply outputting the text to be returned to the client side program. For the .load() method, the server side "program" could even be as simple as just an HTML document. For the .get() or .post() function, the server side program would be very similar to a normal form handling program. For a .getJSON() function, the server-side program has to take the extra step of formatting the output as JSON objects. Even in the most complex implementations, the server side program is typically the easiest part of an AJAX implementation.

One really important limitation to be aware of is that, for good security reasons, you generally can only use AJAX to communicate with a server side program in the same domain as the client side web page. If you're trying to access a web service in another domain (for example the Living Stones web service we discussed in June), this is easily resolved by writing a simple server side program on your web server that passes along the request to the web service and then passes along the response to the client side AJAX program.

The most challenging aspect of implementing AJAX is testing and troubleshooting. Since we can't "see" the interaction between the client and server components, it can be very challenging to figure out why an AJAX solution isn't working. But what fun is programming without a few challenges to work through?

I hope and pray that this article will help you in making your websites better serve your visitors to the glory of God.

Russ McGuire is an executive for a Fortune 100 company and the founder/co-founder of three technology start-ups. His latest entrepreneurial venture is Hschooler.net (http://hschooler.net), a social network for Christian families (especially homeschoolers) which is being built and run by three homeschooled students under Russ' direction.