How-To: Build a Web Application with Ajax Part 1

Ajax Overview

The term AJAX, originally coined by Jesse James Garrett of Adaptive Path in his essay AJAX: A New Approach To Web Applications, is an acronym for "Asynchronous JavaScript And XML." That's a bit of a mouthful, but it's simply describing a technique that uses JavaScript to refresh a page's contents from a web server without having to reload the entire page. This is different from the traditional method of updating web pages, which requires the browser to refresh the entire page in order to display any changes to the content.

Similar techniques have been around in one form or another (often achieved with the help of some clever hacks) for quite a while. But the increasing availability of the XMLHttpRequest class in browsers, the coining of the catchy term AJAX, and the advent of a number of high-profile examples such as Google Maps, Gmail, Backpack, and Flickr, have allowed these kinds of highly interactive web applications to begin to gain traction in the development world.

As the term AJAX has become more widespread, its definition has expanded to refer more generally to browser-based applications that behave much more dynamically than old-school web apps. This new crop of AJAX web applications make more extensive use of interaction techniques like edit-in-place text, drag-and-drop, and CSS animations or transitions to effect changes within the user interface. This tutorial will explain those techniques, and show you how to develop AJAX web applications of your own.

This tutorial is an excerpt from my new book, *Build Your Own AJAX Web Applications*. In the three chapters presented here, we'll discuss the basics of AJAX and learn how it ticks, before delving into the wonderful world of XMLHttpRequest. After we've played around with it, exploring its inner workings, making requests, and updating our

application page asynchronously, we begin to develop our first true AJAX application.

It's going to be quite a ride, so I hope you're ready for some adventure! If you'd rather read these chapters to offline, download the .pdf version of them. But now, let's get a solid grounding in AJAX.

Chapter 1. AJAX: the Overview

He's escaping, idiot! Dispatch War Rocket Ajax! To bring back his body!

AJAX Web Applications

AJAX can be a great solution for many web development projects — it can empower web apps to step up and take over a lot of the ground that previously was occupied almost exclusively by desktop applications.

All the same, it's important to keep in mind that AJAX is not a sort of magic fairy dust that you can sprinkle on your app to make it whizzy and cool. Like any other new development technique, AJAX isn't difficult to mis-use, and the only thing worse than a horrible, stodgy, old-school web app is a horrible, poorly executed AJAX web app.

When you apply it to the right parts of your web application, in the right ways, AJAX can enhance users' experience of your application significantly. AJAX can improve the interactivity and speed of your app, ultimately making that application easier, more fun, and more intuitive to use.

Often, AJAX applications are described as being "like a desktop application in the browser." This is a fairly accurate description — AJAX web apps are significantly more responsive than traditional, old-fashioned web applications, and they can provide levels of interactivity similar to those of desktop applications.

But an AJAX web app is still a remote application, and behaves differently from a desktop application that has access to local

storage. Part of your job as an AJAX developer is to craft applications that feel responsive and easy to use despite the communication that must occur between the app and a distant server. Fortunately, the AJAX toolbox gives you a number of excellent techniques to accomplish exactly that.

The Bad Old Days

One of the first web development tasks that moved beyond serving simple, static HTML pages was the technique of building pages dynamically on the web server using data from a back-end data store.

Back in the "bad old days" of web development, the only way to create this dynamic, database-driven content was to construct the entire page on the server side, using either a CGI script (most likely written in Perl), or some server component that could interpret a scripting language (such as Microsoft's Active Server Pages). Even a single change to that page necessitated a round trip from browser to server — only then could the new content be presented to the user.

In those days, the normal model for a web application's user interface was a web form that the user would fill out and submit to the server. The server would process the submitted form, and send an entirely new page back to the browser for display as a result. So, for example, the completion of a multi-step, web-based "wizard" would require the user to submit a form — thereby prompting a round-trip between the browser and the server — for each step.

Granted, this was a huge advance on static web pages, but it was still a far cry from presenting a true "application" experience to end-users.

Prehistoric AJAX

Early web developers immediately began to look for tricks to extend the capabilities of that simple forms-based model, as they strove to create web applications that were more responsive and interactive. These hacks, while fairly ad hoc and crude, were the first steps web

developers took toward the kind of interactivity we see in today's AJAX applications. But, while these tricks and workarounds often provided serviceable, working solutions, the resulting code was not a pretty sight.

Nesting Framesets

One way to get around the problem of having to reload the entire page in order to display even the smallest change to its content was the hideous hack of nesting framesets within other framesets, often several levels deep. This technique allowed developers to update only selected areas of the screen, and even to mimic the behavior of tab-style navigation interfaces in which users' clicking on tabs in one part of the screen changed content in another area.

This technique resulted in horrible, unmaintainable code with profusions of pages that had names like EmployeeEditWizardMiddleLowerRight.asp.

The Hidden `iframe`

The addition of the `iframe` in browsers like Internet Explorer 4 made things much less painful. The ability to hide the iframe completely led to the development of another neat hack: developers would make HTTP requests to the server using a hidden iframe, then insert the content into the page using JavaScript and DHTML. This provided much of the same functionality that's available through modern AJAX, including the ability to submit data from forms without reloading the page — a feat that was achieved by having the form submit to the hidden iframe. The result was returned by the server to the `iframe`, where the page's JavaScript could access it.

The big drawback of this approach (beyond the fact that it was, after all, a hack) was the annoying burden of passing data back and forth between the main document and the document in the iframe.

Remote Scripting

Another early AJAX-like technique, usually referred to as remote scripting, involved setting the `src` attribute of a `<script>` tag to load pages that contained dynamically generated JavaScript.

This had the advantage of being much cleaner than the hidden `iframe` hack, as the JavaScript generated on the server would load right into the main document. However, only simple GET requests were possible using this technique.

What Makes AJAX Cool

This is why AJAX development is such an enormous leap forward for web development: instead of having to send everything to the server in a single, huge mass, then wait for the server to send back a new page for rendering, web developers can communicate with the server in smaller chunks, and selectively update specific areas of the page based on the server's responses to those requests. This is where the word asynchronous in the AJAX acronym originated.

It's probably easiest to understand the idea of an asynchronous system by considering its opposite — a synchronous system. In a synchronous system, everything occurs in order. If a car race was a synchronous system, it would be a very dull affair. The car that started first on the grid would be the first across the finish line, followed by the car that started second, and so on. There would be no overtaking, and if a car broke down, the traffic behind would be forced to stop and wait while the mechanics made their repairs.

Traditional web apps use a synchronous system: you must wait for the server to send you the first page of a system before you can request the second page, as shown in Figure 1.1.

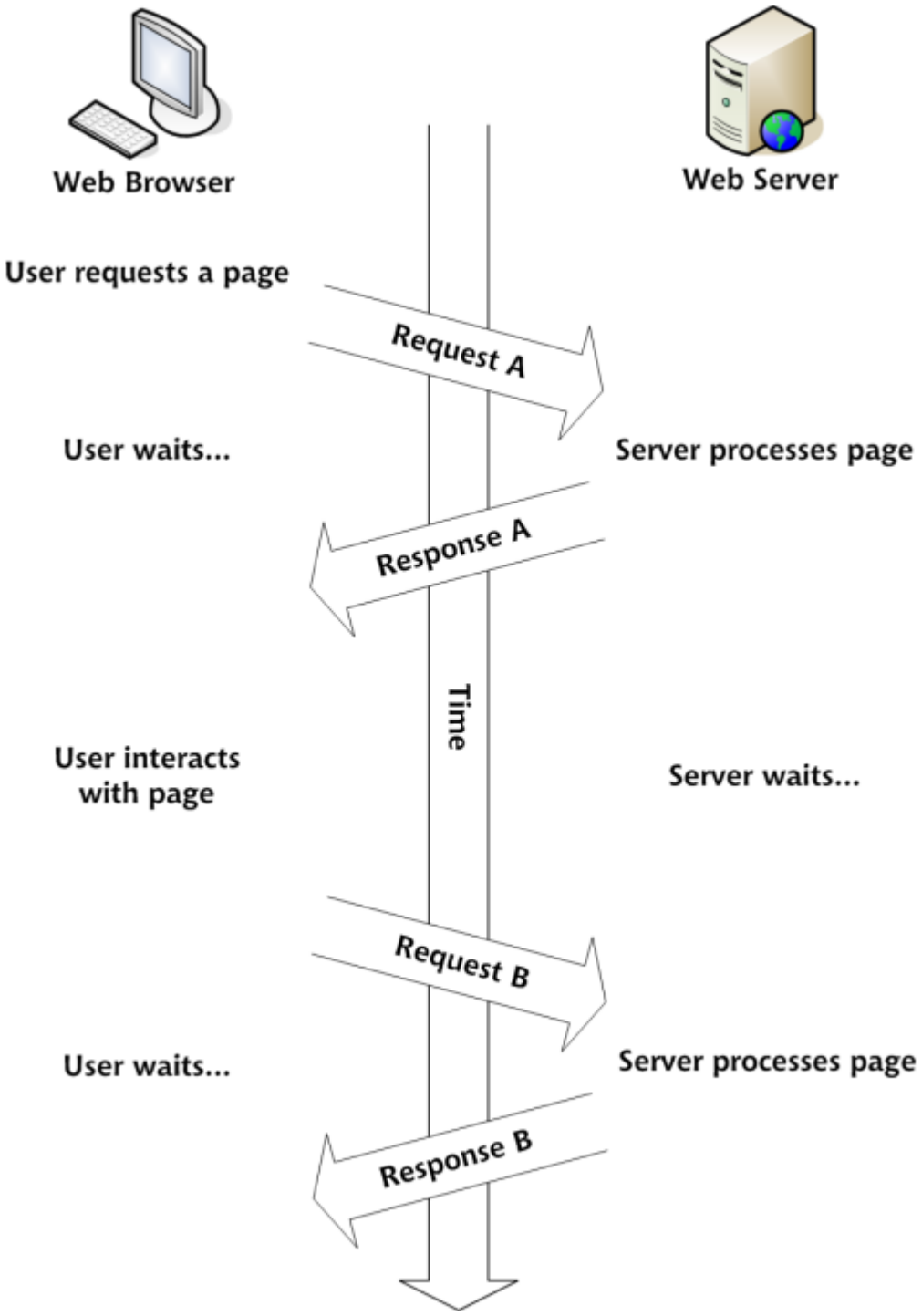

Figure 1.1. A traditional web app is a synchronous system

An asynchronous car race would be a lot more exciting. The car in pole position could be overtaken on the first corner, and the car that starts from the back of the grid could weave its way through the field and cross the finish line in third place. The HTTP requests from the

browser in an AJAX application work in exactly this way. It's this ability to make lots of small requests to the server on a needs-basis that makes AJAX development so cool. Figure 1.2 shows an AJAX application making asynchronous requests to a web server.

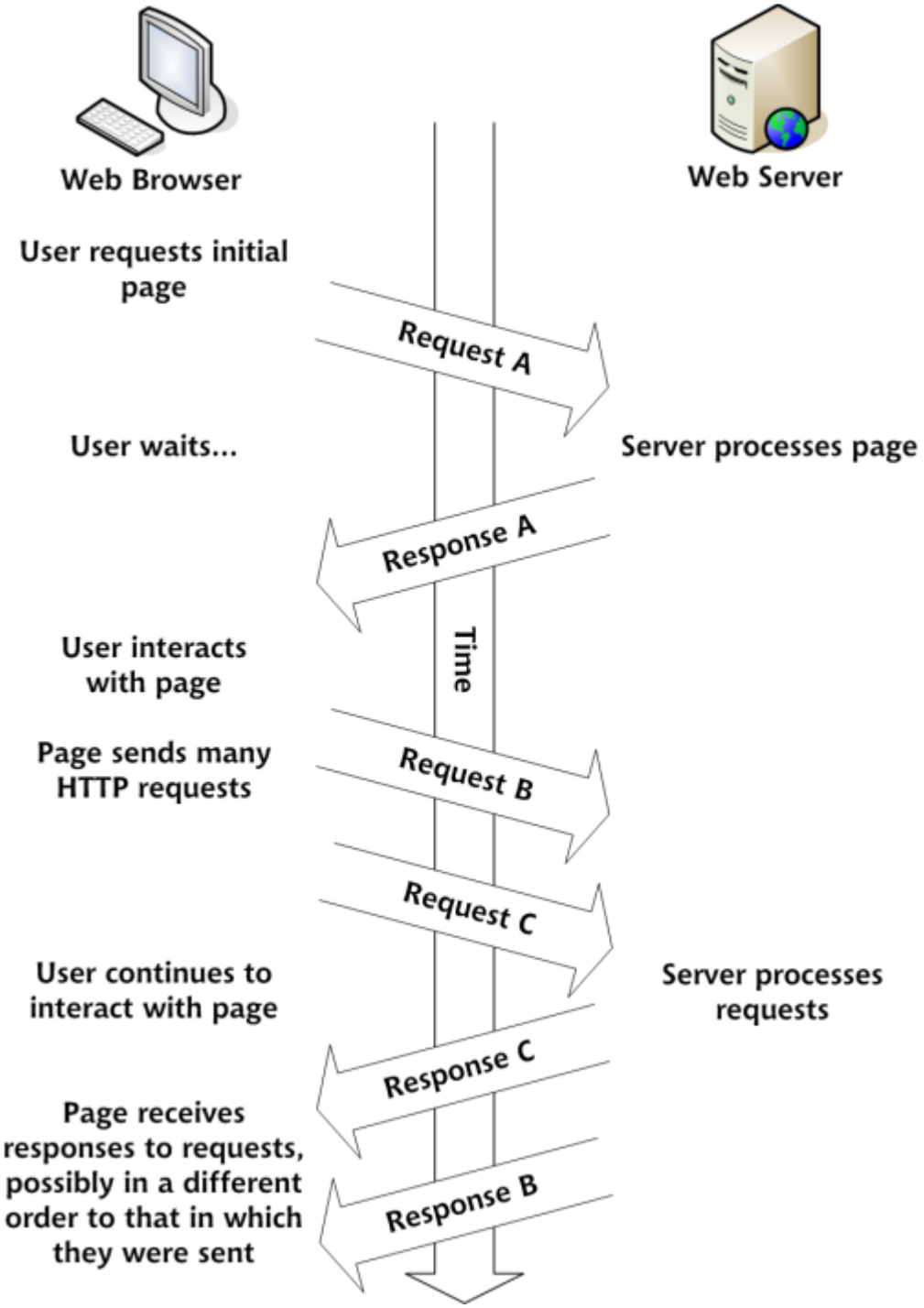

Figure 1.2. An AJAX web app is an asynchronous system

The end result is an application that feels much more responsive, as users spend significantly less time waiting for requests to process, and don't have to wait for an entire new web page to come across the wire, and be rendered by their browsers, before they can view the results.

AJAX Technologies

The technologies that are used to build AJAX web applications encompass a number of different programming domains, so AJAX development is neither as straightforward as regular applications development, nor as easy as old-school web development.

On the other hand, the fact that AJAX development embraces so many different technologies makes it a lot more interesting and fun. Here's a brief listing of the technologies that work together to make an AJAX web application:

- XML
- the W3C DOM
- CSS
- XMLHttpRequest
- JavaScript

Through the rest of this chapter, we'll meet each of these technologies and discuss the roles they play in an AJAX web application.

Data Exchange and Markup: XML

XML (XML stands for Extensible Markup Language — not that anyone ever calls it that outside of textbooks.) is where AJAX gets its letter "X." This is fortunate, because tech acronyms are automatically seen as being much cooler if they contain the letter "X." (Yes, I am kidding!)

Data Exchange Lingua Franca

XML often serves as the main data format used in the asynchronous HTTP requests that communicate between the browser and the server

in an AJAX application. This role plays to XML's strengths as a neutral and fairly simple data exchange format, and also means that it's relatively easy to reuse or reformat content if the need arises.

There are, of course, numerous other ways to format your data for easy exchange between the browser and the server (such as CSV (comma separated values), JSON (JavaScript object notation), or simply plain text), but XML is one of the most common.

XML as Markup

The web pages in AJAX applications consist of XHTML markup, which is actually just a flavor of XML. XHTML, as the successor to HTML, is very similar to it. It's easily picked up by any developer who's familiar with old-school HTML, yet it boasts all the benefits of valid XML. There are numerous advantages to using XHTML:

- It offers lots of standard tools and script libraries for viewing, editing, and validating XML.
- It's forward-compatible with newer, XML-compatible browsers.
- It works with either the HTML Document Object Model (DOM) or the XML DOM.
- It's more easily repurposed for viewing in non-browser agents.

Some of the more pedantic folks in the development community insist that people should not yet be using XHTML. They believe very strongly that XHTML, since it is actual XML, should not be used at all unless it can be served with a proper HTTP `Content-Type` header of `application/xhtml+xml` (`text/xml` and `application/xml` would also be okay, though they're less descriptive) for which, at present, there is still limited browser support. (Internet Explorer 6 and 7 do not support it at all.)

In practice, you can serve XHTML to the browser with a `Content-Type` of `text/html`, as all the mainstream browsers render correctly all XHTML documents served as text/html. Although browsers will treat your code as plain old HTML, other programs can still interpret it

as XML, so there's no practical reason not to "future-proof" your markup by using it.

If you happen to disagree with me, you can choose instead to develop using the older HTML 4.01 standard. This is still a viable web standard, and is a perfectly legitimate choice to make in developing your web application.

XHTML and this Book

Most of the code examples in this book will use XHTML 1.0 Strict. The iframe element is not available in Strict, so the few code examples we show using the iframe will be XHTML 1.0 Transitional.

The World Wide Web Consortium maintains an FAQ on the differences between HTML and XHTML.

W3C Document Object Model

The Document Object Model (DOM) is an object-oriented representation of XML and HTML documents, and provides an API for changing the content, structure, and style of those documents.

Originally, specific browsers like Netscape Navigator and Internet Explorer provided differing, proprietary ways to manipulate HTML documents using JavaScript. The DOM arose from efforts by the World Wide Web Consortium (W3C) to provide a platform- and browser-neutral way to achieve the same tasks.

The DOM represents the structure of an XML or HTML document as an object hierarchy, which is ideal for parsing by standard XML tools.

DOM Manipulation Methods

JavaScript provides a large API for dealing with these DOM structures, in terms of both parsing and manipulating the document. This is one of the primary ways to accomplish the smaller, piece-by-piece changes to a web page that we see in an AJAX application. (Another

method is simply to change the `innerHTML` property of an element. This method is not well documented in any standard, though it's widely supported by mainstream browsers.)

DOM Events

The other important function of the DOM is that it provides a standard means for JavaScript to attach events to elements on a web page. This makes possible much richer user interfaces, because it allows you to give users opportunities to interact with the page beyond simple links and form elements.

A great example of this is drag-and-drop functionality, which lets users drag pieces of the page around on the screen, and drop them into place to trigger specific pieces of functionality. This kind of feature used to exist only in desktop applications, but now it works just as well in the browser, thanks to the DOM.

Presentation: CSS

CSS (Cascading Style Sheets) provides a unified method for controlling the appearance of user interface elements in your web application. You can use CSS to change almost any aspect of the way the page looks, from font sizes, colors, and spacing, to the positioning of elements.

In an AJAX application, one very good use of CSS is to provide user-interface feedback (with CSS-driven animations and transitions), or to indicate portions of the page with which the user can interact (with changes to color or appearance triggered, for example, by mouseovers). For example, you can use CSS transitions to indicate that some part of your application is waiting for an HTTP request that's processing on the server.

CSS manipulation figures heavily in the broader definition of the term AJAX — in various visual transitions and effects, as well as in drag-and-drop and edit-in-place functionality.

Communication: `XMLHttpRequest`

`XMLHttpRequest`, a JavaScript class with a very easy-to-use interface, sends and receives HTTP requests and responses to and from web servers. The `XMLHttpRequest` class is what makes true AJAX application development possible. The HTTP requests made with `XMLHttpRequest` work just as if the browser were making normal requests to load a page or submit a form, but without the user ever having to leave the currently loaded web page.

Microsoft first implemented `XMLHttpRequest` in Internet Explorer 5 for Windows as an ActiveX object. The Mozilla project provided a JavaScript-native version with a compatible API in the Mozilla browser, starting in version 1.0. (It's also available in Firefox, of course.) Apple has added `XMLHttpRequest` to Safari since version 1.2.

The response from the server — either an XML document or a string of text — can be passed to JavaScript to use however the developer sees fit — often to update some piece of the web application's user interface.

Putting it All Together: JavaScript

JavaScript is the glue that holds your AJAX application together. It performs multiple roles in AJAX development:

- controlling HTTP requests that are made using `XMLHttpRequest`
- parsing the result that comes back from the server, using either DOM manipulation methods, XSLT, or custom methods, depending on the data exchange format used
- presenting the resulting data in the user interface, either by using DOM manipulation methods to insert content into the web page, by updating an element's `innerHTML` property, or by changing elements' CSS properties

Because of its long history of use in lightweight web programming (and at the hands of inexperienced programmers), JavaScript has not been seen by many traditional application developers as a "serious programming language," despite the fact that, in reality, it's a fully-featured, dynamic language capable of supporting object-oriented programming methodologies.

The misperception of JavaScript as a "toy language" is now changing rapidly as AJAX development techniques expand the power and functionality of browser-based applications. As a result of the advent of AJAX, JavaScript now seems to be undergoing something of a renaissance, and the explosive growth in the number of JavaScript toolkits and libraries available for AJAX development is proof of the fact.

Summary

In this chapter, we had a quick overview of AJAX and the technologies that make it tick. We looked at some of the horrible coding contortions that developers had to endure back in the bad old days to create something resembling an interactive UI, and we saw how AJAX offers a huge improvement on those approaches. With a decent command of the building blocks of AJAX — XML, the DOM, CSS, XMLHttpRequest, and JavaScript, which ties them all together — you have everything you need to start building dynamic and accessible AJAX sites.

Chapter 2. Basic XMLHttpRequest

I can't wait to share this new wonder, The people will all see its light, Let them all make their own music, The priests praise my name on this night.

— Rush, Discovery

It's `XMLHttpRequest` that gives AJAX its true power: the ability to make asynchronous HTTP requests from the browser and pull down content in small chunks.

Web developers have been using tricks and hacks to achieve this for a long time, while suffering annoying limitations: the invisible iframe hack forced us to pass data back and forth between the parent document and the document in the `iframe`, and even the "remote scripting" method was limited to making GET requests to pages that contained JavaScript.

Modern AJAX techniques, which use XMLHttpRequest, provide a huge improvement over these kludgy methods, allowing your app to make both GET and POST requests without ever completely reloading the page.

In this chapter, we'll jump right in and build a simple AJAX web application — a simple site-monitoring application that pings a page on a web server to a timed schedule. But before we start making the asynchronous HTTP requests to poll the server, we'll need to simplify the use of the XMLHttpRequest class by taking care of all of the little browser incompatibilities, such as the different ways XMLHttpRequest objects are instantiated, inside a single, reusable library of code.

A Simple AJAX Library

One approach to simplifying the use of the `XMLHttpRequest` class would be to use an existing library of code. Thanks to the increasing popularity of AJAX development, there are literally dozens of libraries, toolkits, and frameworks available that make `XMLHttpRequest` easier to use.

But, as the code for creating an instance of the `XMLHttpRequest` class is fairly simple, and the API for using it is easy to understand, we'll just write a very simple JavaScript library that takes care of the basic stuff we need.

Stepping through the process of creating your own library will ensure you know how the `XMLHttpRequest` class works, and will help you get more out of those other toolkits or libraries when you do decide to use them.

Starting our `Ajax` Class

We'll start by creating a basic class, called `Ajax`, in which we'll wrap the functionality of the `XMLHttpRequest` class.

I've Never done Object Oriented Programming in JavaScript — Help!

In this section, we'll start to create classes and objects in JavaScript. If you've never done this before, don't worry — it's quite simple as long as you know the basics of object oriented programming.

In JavaScript, we don't declare classes with complex syntax like we would in Java, C++ or one of the .NET languages; we simply write a constructor function to create an instance of the class. All we need to do is:

- provide a constructor function — the name of this function is the name of your class
- add properties to the object that's being constructed using the keyword this, followed by a period and the name of the property
- add methods to the object in the same way we'd add properties, using JavaScript's special function constructor syntax

Here's the code that creates a simple class called `HelloWorld`:

```
function HelloWorld() {

  this.message = 'Hello, world!';

  this.sayMessage = function() {

    window.alert(this.message);

  };

}
```

JavaScript's framework for object oriented programming is very lightweight, but functions surprisingly well once you get the hang of it.

More advanced object oriented features, such as inheritance and polymorphism, aren't available in JavaScript, but these features are rarely needed on the client side in an AJAX application. The complex business logic for which these features are useful should always be on the web server, and accessed using the `XMLHttpRequest` class.

In this example, we create a class called `HelloWorld` with one property (`message`) and one method (`sayMessage`). To use this class, we simply call the constructor function, as shown below:

```
var hw = new HelloWorld();

hw.sayMessage();

hw.message = 'Goodbye';

hw.sayMessage();
```

Here, we create an instance of `HelloWorld` (called `hw`), then use this object to display two messages. The first time we call `sayMessage`, the default "Hello, world!" message is displayed. Then, after changing our object's `message` property to "Goodbye," we call `sayMessage` and "Goodbye" is displayed.

Don't worry if this doesn't make too much sense at the moment. As we progress through the building of our `Ajax` class, it will become clearer.

Here are the beginnings of our `Ajax` class's constructor function:

```
Example 2.1. ajax.js (excerpt)



function Ajax() {

  this.req = null;
```

```
  this.url = null;

  this.method = 'GET';

  this.async = true;

  this.status = null;

  this.statusText = '';

  this.postData = null;

  this.readyState = null;

  this.responseText = null;

  this.responseXML = null;

  this.handleResp = null;

  this.responseFormat = 'text', // 'text', 'xml',
or 'object'

  this.mimeType = null;

}
```

This code just defines the properties we'll need in our `Ajax` class in order to work with `XMLHttpRequest` objects. Now, let's add some methods to our object. We need some functions that will set up an `XMLHttpRequest` object and tell it how to make requests for us.

Creating an `XMLHttpRequest` Object

First, we'll add an `init` method, which will create an `XMLHttpRequest` object for us.
Unfortunately, `XMLHttpRequest` is implemented slightly differently in Firefox (in this book, whenever I explain how something works in

Firefox, I'm referring to all Mozilla-based browsers, including Firefox, Mozilla, Camino, and SeaMonkey), Safari, and Opera than it was in Internet Explorer's original implementation (interestingly, Internet Explorer version 7 now supports the same interface as Firefox, which promises to simplify AJAX development in the future), so you'll have to try instantiating the object in a number of different ways if you're not targeting a specific browser. Firefox and Safari create `XMLHttpRequest` objects using a class called `XMLHttpRequest`, while Internet Explorer versions 6 and earlier use a special class called `ActiveXObject` that's built into Microsoft's scripting engine. Although these classes have different constructors, they behave in the same way.

Cross-browser Code

Fortunately, most modern browsers (Internet Explorer 6, Firefox 1.0, Safari 1.2, and Opera 8, or later versions of any of these browsers) adhere to web standards fairly well overall, so you won't have to do lots of browser-specific branching in your AJAX code.

This usually makes a browser-based AJAX application faster to develop and deploy cross-platform than a desktop application. As the power and capabilities available to AJAX applications increase, desktop applications offer fewer advantages from a user-interface perspective.

The `init` method looks like this:

Example 2.2. ajax.js (excerpt)

```
this.init = function() {
  if (!this.req) {
```

```
    try {
      // Try to create object for Firefox, Safari,
IE7, etc.
      this.req = new XMLHttpRequest();
    }
    catch (e) {
      try {
        // Try to create object for later versions
of IE.
        this.req = new
ActiveXObject('MSXML2.XMLHTTP');
      }
      catch (e) {
        try {
          // Try to create object for early
versions of IE.
          this.req = new
ActiveXObject('Microsoft.XMLHTTP');
        }
        catch (e) {
          // Could not create an XMLHttpRequest
object.
```

```
            return false;

          }

        }

      }

    }

    return this.req;

};
```

The `init` method goes through each possible way of creating an `XMLHttpRequest` object until it creates one successfully. This object is then returned to the calling function.

Degrading Gracefully

Maintaining compatibility with older browsers (by "older" I mean anything older than the "modern browsers" I mentioned in the previous note) requires a lot of extra code work, so it's vital to define which browsers your application should support.

If you know your application will receive significant traffic via older browsers that don't support the `XMLHtmlRequest` class (e.g., Internet Explorer 4 and earlier, Netscape 4 and earlier), you will need either to leave it out completely, or write your code so that it degrades gracefully. That means that instead of allowing your functionality simply to disappear in less-capable browsers, you code to ensure that users of those browsers receive something that's functionally equivalent, though perhaps in a less interactive or easy-to-use format.

It's also possible that your web site will attract users who browse with JavaScript disabled. If you want to cater to these users, you should provide an alternative, old-school interface by default, which you can then modify on-the-fly — using JavaScript — for modern browsers.

Continue with Part 2: Send a Request via HTTP…

Courtesy: https://www.sitepoint.com/build-your-own-ajax-web-apps/

Modified: 2021.10.04.7.10.AM

Dököll Solutions,. Inc.