Comet Programming: Using Ajax to Simulate Server Push

By Rob Gravelle


[next]

Comet is a Web application model that enables web servers to send data to the client without having to explicitly request it. Developers can utilize Comet techniques to create event-driven Web apps. Comet is actually an umbrella term for multiple techniques for achieving client-server interaction. All methods have in common that they rely on browser-native technologies such as JavaScript, rather than on proprietary plug-ins. This article will examine one method of implementing Comet called Ajax Long Polling and demonstrate how it can be used to refresh page components and keep cached data in synch with the server.

Older Models for Refreshing Browser Content

In the early days of the World Wide Web, the browser would make multiple server requests: one for the page content and one for each page component. Examples of page components include images, CSS files, scripts, Java applets, and any other server-hosted resource referenced in the page. Downloading a page in chunks of data is known as a page-by-page model. One of the main drawbacks to this early model was that you couldn't easily refresh any one page element. JavaScript allowed for image swapping by setting the `src` property of an image, but in most cases, the browser would have to refresh the entire page. Even this did not always work as intended, as browser and server caching could cause the same element to display on subsequent requests!

Ajax (asynchronous JavaScript and XML), went a long way towards eliminating these limitations. It allowed far greater control over page content by providing the ability to send server requests for as little, or as much data as the browser needed to update. In addition, its asynchronous nature supported multiple simultaneous calls - even while other elements downloaded. One problem that Ajax did not adequately solve was the issue of data synchronization between the client and server. Since the browser would not know if something had changed on the server, Web applications typically polled the server on a periodic basis to ask if new information was available. Although better than whole page refreshes, high polling frequencies could still waste server resources and bandwidth.

In 1995, Netscape Navigator added a feature called "server push". Taking advantage of server-side programming and the content type "multipart/x-mixed-replace" feature of the MIME standard, it allowed servers to send new

versions of an image or HTML page to that browser, as part of a multipart HTTP response. Since 2004, Gecko-based browsers such as Firefox accept multipart responses to XHR as well. On the server side, each component was encoded as a separate portion of the multipart response; on the client, the callback function provided to the XHR `onreadystatechange` function to be called as each component arrived. Unfortunately, this functionality was only included in Gecko-based browsers.

Comet Techniques Overview

Comet techniques come in two flavors: streaming and long polling. In a Web application using streaming Comet, the browser opens a single persistent connection to the server for all Comet events and handles them incrementally on the browser side. Specific examples of streaming Comet techniques include the Hidden IFrame and the XMLHttpRequest Server Push. In long polling, the client makes an Ajax request to the server, which is kept open until the server has new data to send to the browser. Upon receiving the server response, the browser initiates a new long polling request in order to obtain the next data set. Long polling can be achieved using either Ajax or script tag techniques.

Comet streaming techniques are considered to be somewhat inferior to long polling for several reasons. The Hidden IFrame technique has two major flaws: the lack of a reliable way to handle errors and the impossibility of tracking the state of the request calling process. The XMLHttpRequest Server Push technique, is even more limited in applicability, as it only works with Gecko-based browsers. Long polling, on the other hand, is easier to implement on the browser side, and works in every browser that supports JavaScript and Ajax.

Implementing Long Polling

Comet is quite useful in applications where data needs to be synchronized between the server and the client. In the "Building an Ajax Caching Manager" article, we were storing data on the client-side in order to minimize redundant server calls. The danger in doing so is that the cached data may become stale because the client has no way of knowing whether or not data has changed on the server. Traditionally, the solution would be to poll the server on a periodic basis and ask it whether or not data has been modified. Unfortunately, doing so increases network traffic, which was the reason for implementing caching in the first place! Long polling would decrease server calls because the connection stays open until the server has an update to announce. Chat apps, such as phpFreeChat, are ideal candidates for long polling, because they are written using non-proprietary technologies (a main tenant of Comet), and they require frequent updating. Other prime candidates for long polling include e-merchant sites, where you need to keep prices and product availability up-to-date.

The latter is what we're going to construct here, using Ajax long polling. The "CD Sales Page" is a website that sells my alter-ego's latest single: a cover of The Police's classic Synchronicity II song. Unlike myself, who is a bona fide geek and conservative professional, this Rob Gravelle is a "Guitar Hero" who doesn't need a video game to conquer the most intricate rhythms and blazing solos. The main HTML page will contain an <INPUT> field to select the number of copies to purchase and a button to send the order. Beneath the <FORM>, a message will display the number of CDs left in stock. We will use long polling to keep this number up-to-date. Here is the HTML code:

view plain | print | ?

```
1  <html>
2  <head>
3  <title>CD Sales Page</title>
4  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
5  <script language="JavaScript" src="prototype.js">
```