Reverse Ajax, Part 1: Introduction to Comet

Streaming and long polling for responsive communication between your server and client

Skill Level: Intermediate

Mathieu Carbou (mathieu.carbou@gmail.com)
Java Web Architect
Ovea

19 Jul 2011

Web development has changed considerably in the past few years. These days, we expect fast, dynamic applications accessible from the web. In this new series, learn how to develop event-driven web applications using Reverse Ajax techniques to achieve a better user experience. The examples on the client side will use the jQuery JavaScript library. In this first article, explore different Reverse Ajax techniques. With downloadable examples, learn about Comet with streaming and long polling methods.

Introduction

Web development has evolved considerably in the past few years. We're beyond the static web pages linked together, which caused browser refreshing and waiting for pages to load. Now, the demand is for completely dynamic applications accessible from the web. These applications often need to be as fast as possible and provide nearly real-time components. In this new five-part series, learn how to develop event-driven web applications using Reverse Ajax techniques.

In this first article, learn about Reverse Ajax, polling, streaming, Comet, and long polling. Learn how to implement different Reverse Ajax communication techniques, and explore the advantages and disadvantages of each method. You can download the source code to follow along with the examples in this article.

Ajax, Reverse Ajax, and WebSockets

Asynchronous JavaScript and XML (Ajax), a browser feature accessible in JavaScript, allows a script to make an HTTP request to a website behind the scenes, without the need for a page reload. Ajax has been around more than a decade. Though the name includes XML, you can transfer nearly anything in an Ajax request. The most commonly used data is JSON, which is close to JavaScript syntax and consumes less bandwidth. Listing 1 shows an example of an Ajax request to retrieve a place's name from its postal code.

Listing 1. Example Ajax request

```
var url = 'http://www.geonames.org/postalCodeLookupJSON?postalcode='
    + $('#postalCode').val() + '&country='
    + $('#country').val() + '&callback=?';
$.getJSON(url, function(data) {
    $('#placeName').val(data.postalcodes[0].placeName);
});
```

You can see this example work in listing1.html in the downloadable source code for this article.

Reverse Ajax is essentially a concept: being able to send data from the server to the client. In a standard HTTP Ajax request, data is sent to the server. Reverse Ajax can be simulated to issue an Ajax request, in specific ways that are covered in this article, so the server can send events to the client as quickly as possible (low-latency communication).

WebSockets, which comes from HTML5, is a much more recent technique. Many browsers already support it (Firefox, Google Chrome, Safari, and others). WebSockets enables bidirectional, full-duplex communication channels. The connection is opened through a sort of HTTP request, called a WebSockets handshake, with some special headers. The connection is kept alive, and you can write and receive data in JavaScript, as if you were using a raw TCP socket. WebSockets will be covered more in Part 2 of this series.

Reverse Ajax techniques

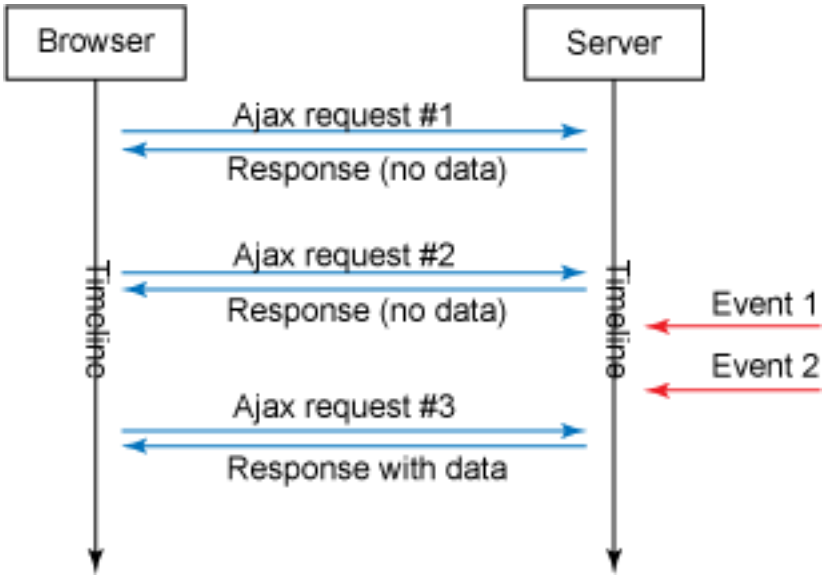
The goal of Reverse Ajax is to let the server push information to the client. Ajax requests are stateless by default, and can only be opened from the client to the server. You can bypass this limitation by using the techniques to simulate responsive communication between the server and client.

HTTP polling and JSONP polling

Polling involves issuing a request from the client to the server to ask for some data. This is obviously a mere Ajax HTTP request. To get the server events as soon as possible, the polling interval (time between requests) must be as low as possible. There's a drawback: if this interval is reduced, the client browser is going to issue many more requests, many of which won't return any useful data, and will consume bandwidth and processing resources for nothing.

The timeline in Figure 1 shows how some polling requests issued by the client but no information is returned. The client must wait for the next polling to get the two events received by the server.

Figure 1. Reverse Ajax with HTTP polling



JSONP polling is essentially the same as HTTP polling. The difference, however, is that with JSONP you can issue cross-domain requests (requests not in your domain). JSONP is used in Listing 1 to get a place name from a postal code. A JSONP request can usually be recognized by its callback parameter and returned content, which is executable JavaScript code.

To implement polling in JavaScript, you can use `setInterval` to periodically issue Ajax requests, as shown in Listing 2:

Listing 2. JavaScript polling

```
setInterval(function() {
    $.getJSON('events', function(events) {
        console.log(events);
    });
}, 2000);
```

The polling demo in the article source code shows the bandwidth consumption by the polling method. The interval is low, but you can see some requests returning no

event. Listing 3 shows the output of the sample polling.

Listing 3. Sample polling demo output

```
[client] checking for events...
[client] no event
[client] checking for events...
[client] 2 events
[event] At Sun Jun 05 15:17:14 EDT 2011
[event] At Sun Jun 05 15:17:14 EDT 2011
[client] checking for events...
[client] 1 events
[event] At Sun Jun 05 15:17:16 EDT 2011
```

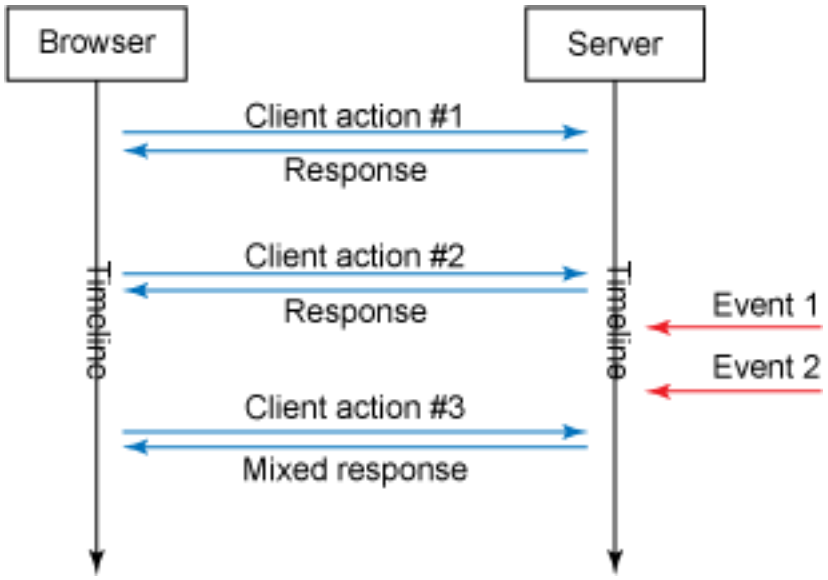
Polling in JavaScript has advantages and disadvantages.

- Advantages: It's really easy to implement and does not require any special features on the server side. It also works in all browsers.
- Disadvantage: This method is rarely employed because it does not scale at all. Imagine the quantity of lost bandwidth and resources in the case of 100 clients each issuing polling requests for 2 seconds, where 30% of the requests returned no data.

Piggyback

Piggyback polling is a much more clever method than polling since it tends to remove all non-needed requests (those returning no data). There is no interval; requests are sent when the client needs to send a request to the server. The difference lies in the response, which is split into two parts: the response for the requested data and the server events, if any occurred. Figure 2 shows an example.

Figure 2. Reverse Ajax with piggyback polling



When implementing the piggyback technique, typically all Ajax requests targeting the server might return a mixed response. An implementation sample is in the article download and in Listing 4 below.

Listing 4. Sample piggyback code

```
$('#submit').click(function() {
    $.post('ajax', function(data) {
        var valid = data.formValid;
        // process validation results
        // then process the other part of the response (events)
        processEvents(data.events);
    });
});
```

Listing 5 shows some piggyback output.

Listing 5. Sample piggyback output

```
[client] checking for events...
[server] form valid ? true
[client] 4 events
[event] At Sun Jun 05 16:08:32 EDT 2011
[event] At Sun Jun 05 16:08:34 EDT 2011
[event] At Sun Jun 05 16:08:34 EDT 2011
[event] At Sun Jun 05 16:08:37 EDT 2011
```

You can see the result of the form validation and the events appended to the response. Again, there are advantages and disadvantages to this method.

- Advantages: With no requests returning no data, since the client controls when it sends requests, you have less resource consumption. It also

works in all browsers and does not require special features on the server side.

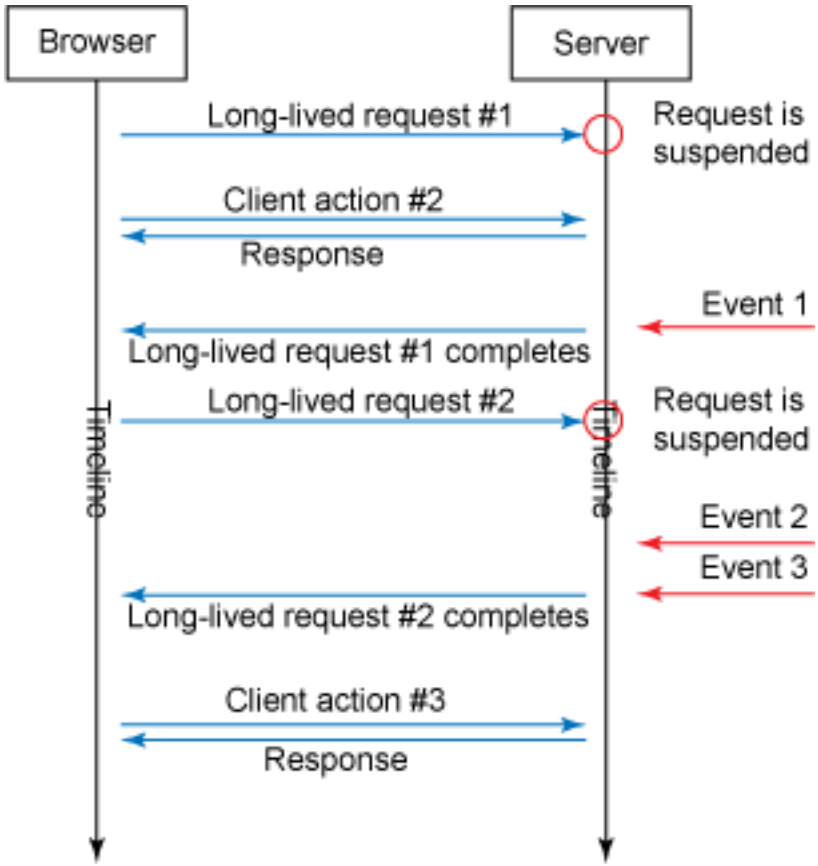
- Disadvantage: You have no clue when the events accumulated on the server side will be delivered to the client because it requires a client action to request them.

Comet

Reverse Ajax with polling or piggyback is very limited: it does not scale and does not provide low-latency communication (when events arrive in the browser as soon as they arrive on the server). *Comet* is a web application model where a request is sent to the server and kept alive for a long time, until a time-out or a server event occurs. When the request is completed, another long-lived Ajax request is sent to wait for other server events. With Comet, web servers can send the data to the client without having to explicitly request it.

The big advantage of Comet is that each client always has a communication link open to the server. The server can push events on the clients by immediately committing (completing) the responses when they arrive, or it can even accumulate and send bursts. Because a request is kept open for a long time, special features are required on the server side to handle all of these long-lived requests. Figure 3 shows an example. (Part 2 in this series will explain the server constraints in more detail.)

Figure 3. Reverse Ajax with Comet



Implementations of Comet can be separated into two types: those using streaming mode, and those using long polling.

Comet using HTTP streaming

In streaming mode, one persistent connection is opened. There will only be a long-lived request (#1 in Figure 3) since each event arriving on the server side is sent through the same connection. Thus, it requires on the client side a way to separate the different responses coming through the same connection. Technically speaking, two common techniques for streaming include Forever Iframes (hidden IFrames) or the multi-part feature of the `XMLHttpRequest` object used to create Ajax requests in JavaScript.

Forever Iframes

The Forever Iframes technique involves a hidden Iframe tag put in the page with its `src` attribute pointing to the servlet path returning server events. Each time an event is received, the servlet writes and flushes a new script tag with the JavaScript code inside. The iframe content will be appended with this script tag that will get executed.

- Advantages: Simple to implement, and it works in all browsers supporting iframes.
- Disadvantages: There is no way to implement reliable error handling or to track the state of the connection, because all connection and data are handled by the browser through HTML tags. You then don't know when the connection is broken on either side.

Multi-part XMLHttpRequest

The second technique, which is more reliable, is to use the multi-part flag supported by some browsers (such as Firefox) on the `XMLHttpRequest` object. An Ajax request is sent and kept open on the server side. Each time an event comes, a multi-part response is written through the same connection. Listing 6 shows an example.

Listing 6. Sample JavaScript code to set up a multi-part streaming request

```
var xhr = $.ajaxSettings.xhr();
xhr.multipart = true;
xhr.open('GET', 'ajax', true);
xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
        processEvents($.parseJSON(xhr.responseText));
    }
};
xhr.send(null);
```

On the server side, things are a little more complicated. You must first set up the multi-part request, and then suspend the connection. Listing 7 shows how to suspend an HTTP streaming request. (Part 3 of this series will cover the APIs in more detail.)

Listing 7. Suspending an HTTP streaming request in a servlet using Servlet 3 API

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
    // start the suspension of the request
    AsyncContext asyncContext = req.startAsync();
    asyncContext.setTimeout(0);

    // send the multipart separator back to the client
    resp.setContentType("multipart/x-mixed-replace;boundary=\""
        + boundary + "\"");
    resp.setHeader("Connection", "keep-alive");
    resp.getOutputStream().print("--" + boundary);
    resp.flushBuffer();

    // put the async context in a list for future usage
    asyncContexts.offer(asyncContext);
}
```

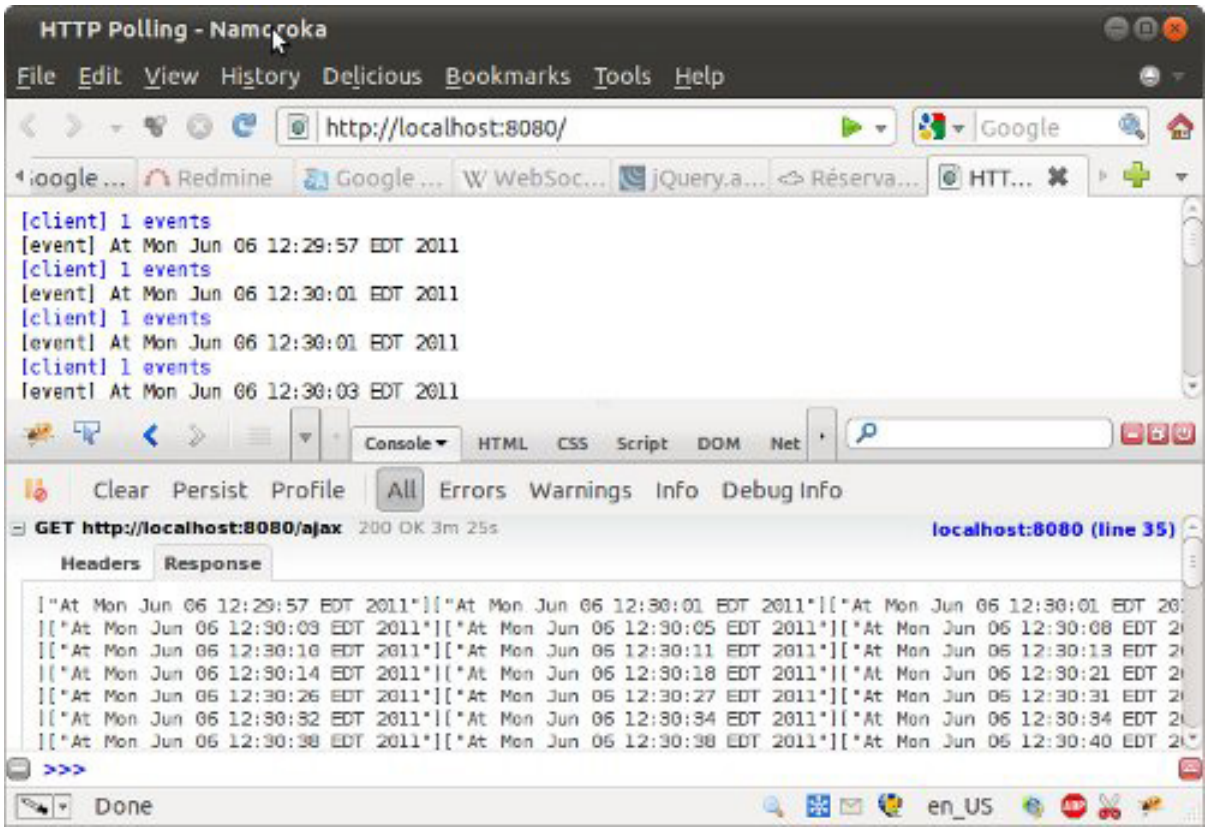
Now, each time an event occurs you can iterate over all suspended connections and write the data to them, as shown in Listing 8:

Listing 8. Send events to a suspended multi-part request using Servlet 3 API

```
for (AsyncContext asyncContext : asyncContexts) {
    HttpServletResponse peer = (HttpServletResponse)
        asyncContext.getResponse();
    peer.getOutputStream().println("Content-Type: application/json");
    peer.getOutputStream().println();
    peer.getOutputStream().println(new JSONArray()
        .put("At " + new Date()).toString());
    peer.getOutputStream().println("--" + boundary);
    peer.flushBuffer();
}
```

The files you can download with this article demonstrate HTTP streaming in the Comet-streaming folder. When you run the sample and open the home page, you'll see that events appear immediately asynchronously as soon as they arrive on the server. Also, if you open the Firebug console, you can see that there is only one Ajax request opened. If you look deeper, you'll see JSON responses appended in the Response tab, as shown in Figure 4:

Figure 4. FireBug view of an HTTP streaming request



As usual, there are advantages and disadvantages.

- Advantage: Only one persistent connection is opened. This is the Comet technique that saves the most bandwidth usage.
- Disadvantage: The multi-part flag is not supported by all browsers. Some widely used libraries, such as CometD in Java, reported issues in buffering. For example, chunks of data (multi-parts) may be buffered and sent only when the connection is completed or the buffer is full, which can create higher latency than expected.

Comet using HTTP long polling

The long polling mode involves techniques that open a connection. The connection is kept open by the server, and, as soon as an event occurs, the response is committed and the connection is closed. Then, a new long-polling connection is reopened immediately by the client waiting for new events to arrive.

You can implement HTTP long polling by using script tags or a mere `XMLHttpRequest` object.

Script tags

As with iframes, the goal is to append a script tag in your page to get the script executed. The server will: suspend the connection until an event occurs, send the script content back to the browser, and then reopen another script tag to get the next events.

- Advantages: Because it's based on HTML tags, this technique is very easy to implement and works across domains (by default, `XMLHttpRequest` does not allow requests on other domains or sub-domains).
- Disadvantages: Similar to the iframe technique, error handling is missing, and you can't have a state or the ability to interrupt a connection.

XMLHttpRequest long polling

The second, and recommended, method to implement Comet is to open an Ajax request to the server and wait for the response. The server requires specific features on the server side to allow the request to be suspended. As soon as an event occurs, the server sends back the response in the suspended request and closes it, exactly like you close the output stream of a servlet response. The client then consumes the response and opens a new long-lived Ajax request to the server, as shown in Listing 9:

Listing 9. Sample JavaScript code to set up long polling requests

```
function long_polling() {
    $.getJSON('ajax', function(events) {
        processEvents(events);
        long_polling();
    });
}

long_polling();
```

On the back end, the code also uses the Servlet 3 API to suspend the request, as in HTTP streaming, but you don't need all the multi-part handling code. Listing 10 shows an example.

Listing 10. Suspending a long polling Ajax request

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
    AsyncContext asyncContext = req.startAsync();
    asyncContext.setTimeout(0);
    asyncContexts.offer(asyncContext);
}
```

When an event is received, simply take all of the suspended requests and complete them, as shown in Listing 11:

Listing 11. Completing a long polling Ajax request when an event occurs

```
while (!asyncContexts.isEmpty()) {
    AsyncContext asyncContext = asyncContexts.poll();
    HttpServletResponse peer = (HttpServletResponse)
        asyncContext.getResponse();
    peer.getWriter().write(
        new JSONArray().put("At " + new Date()).toString());
    peer.setStatus(HttpServletResponse.SC_OK);
    peer.setContentType("application/json");
    asyncContext.complete();
}
```

In the accompanying downloadable source files, the comet-long-polling folder contains a long polling sample web application that you can run using the mvn `jetty:run` command.

- Advantages: It's easy to implement on the client side with a good error-handling system and timeout management. This reliable technique also allows a round-trip between connections on the server side, since connections are not persistent (a good thing, when you have a lot of clients on your application). It also works on all browsers; you only make use of the `XMLHttpRequest` object by issuing a simple Ajax request.

- Disadvantage: There is no main disadvantage compared to other techniques. But, like all techniques we've discussed, this one still relies on a stateless HTTP connection, which requires special features on the server side to be able to temporarily suspend it.

Recommendations

Because all modern browsers support the Cross-Origin Resource Sharing (CORS) specification, which allows XHR to perform cross-domain requests, the need for script-based and iframe-based techniques becomes deprecated.

The best way to implement and use Comet for Reverse Ajax is through the `XMLHttpRequest` object, which provides a real connection handle and error handling. Considering that not all browsers support the multi-part flag, and multi-part streaming can be subject to buffering issues, it is recommended that you use Comet through HTTP long polling with the `XMLHttpRequest` object (a simple Ajax request that is suspended on the server side). All browsers supporting Ajax also support this method.

Conclusion

This article provided an introduction to Reverse Ajax techniques. It explored different ways to implement Reverse Ajax communication, and it explained the advantages and drawbacks for each implementation. Your particular situation and the requirements of your application will influence which method is best for you. Generally speaking though, Comet with Ajax long-polling requests is the way to go if you want the best compromise of: low-latency communication; timeout and error detection; simplicity; and good support from all browsers and platforms.

Stay tuned for Part 2 in this series, which will explore a third Reverse Ajax technique: WebSockets. Though not all browsers support it yet, WebSockets will definitely be a very good communication medium for Reverse Ajax. WebSockets will remove all constraints relative to the stateless characteristic of an HTTP connection. Part 2 will also cover the server-side constraints induced by Comet and WebSocket techniques.

Downloads

Description	Name	Size	Download method
Article source code	reverse_ajaxpt1_source.zip	17KB	HTTP

Information about download methods

Resources

Learn

- On Wikipedia, read about:
  - Ajax
  - Reverse Ajax
  - Comet
  - WebSockets
- "Exploring Reverse AJAX" (Google Maps .Net Control blog, August 2006): Get an introduction to some Reverse-Ajax techniques.
- "Cross-domain communications with JSONP, Part 1: Combine JSONP and jQuery to quickly build powerful mashups" (developerWorks, February 2009): See how you can combine an obscure cross-domain call technique (JSONP) and a flexible JavaScript library (jQuery) to build powerful mashups surprisingly quickly.
- "Cross-Origin Resource Sharing (CORS)" specification (W3C, July 2010): Learn more about this mechanism, which allows XHR to perform cross-domain requests.
- "Build Ajax applications with Ext JS" (developerWorks, July 2008): Get an overview of the object-oriented JavaScript design concepts behind Ext JS, and shows how to use the Ext JS framework for rich Internet application UI elements.
- "Compare JavaScript frameworks" (developerWorks, February 2010): Get an overview of the frameworks that greatly enhance JavaScript development.
- "Mastering Ajax, Part 2: Make asynchronous requests with JavaScript and Ajax" (developerWorks, January 2006): Learn how to use Ajax and the `XMLHttpRequest` object to create a request/response model that never leaves users waiting for a server to respond.
- "Create Ajax applications for the mobile Web" (developerWorks, March 2010): Understand how to build cross-browser smartphone Web applications using Ajax.
- "Where and when to use Ajax in your applications" (developerWorks, February 2008): See how you can use Ajax to improve your Web sites while avoiding bad user experiences.
- "Improve the performance of Web 2.0 applications" (developerWorks, December 2009): Explore different browser-side cache mechanisms.

- "Introducing JSON" (JSON.org): Get an introduction to JSON syntax.
- developerWorks Web development zone: Find articles covering various Web-based solutions.
- developerWorks podcasts: Listen to interesting interviews and discussions for software developers.
- developerWorks technical events and webcasts: Stay current with developerWorks technical events and webcasts.

Get products and technologies

- Get ExtJS, the cross-browser JavaScript library for building rich internet applications.
- XAMPP provides easy installation of Apache, PHP, MySQL and other goodies.
- Try out IBM software for free. Download a trial version, log into an online trial, work with a product in a sandbox environment, or access it through the cloud. Choose from over 100 IBM product trials.

Discuss

- Create your developerWorks profile today and setup a watchlist on Reverse Ajax. Get connected and stay connected with developerWorks community.
- Find other developerWorks members interested in web development.
- Share what you know: Join one of our developerWorks groups focused on web topics.
- Roland Barcia talks about Web 2.0 and middleware in his blog.
- Follow developerWorks' members' shared bookmarks on web topics.
- Get answers quickly: Visit the Web 2.0 Apps forum.
- Get answers quickly: Visit the Ajax forum.

About the author

Mathieu Carbou

Mathieu Carbou, a Java web architect and consultant at Ovea, provides services and development solutions. He is a committer and leader of several open source projects, a speaker, and a leader of Montreal's Java User Group. Mathieu has a strong background in code design and best practices, and is a specialist of event-driven web development from the client side to the back end. He focuses on providing event-driven and messaging solutions in highly scalable web

applications. Check out his blog.