# Contents

Chapter 1: What Are Comet and Reverse Ajax? ........................................ 1

The Trouble with HTTP .......................................................................... 2

Some Common Use Cases......................................................................... 5
  Monitoring and Data Feeds ................................................................  5
  Progress Updates ...............................................................................  6
  Chat and Collaboration ......................................................................  8

Summary .................................................................................................. 9

Chapter 2: Simple Ways to Achieve Push .............................................. 11

The Magnetic Poetry Application.......................................................... 11
  Creating New Words .......................................................................... 15
  Reading Words ................................................................................... 15
  Updating Words .................................................................................. 17
  Deleting Words ................................................................................... 18

Introducing Push Using Polling ............................................................. 19

Improving Efficiency Using Piggybacking .......................................... 25

Summary ................................................................................................ 30

Chapter 3: Introducing Comet............................................................... 33

Implementing a Comet Feed Using XHR ............................................. 33

Script Tags, Iframes, and Comet .......................................................... 40
Long Polling.............................................................................................................. 41

Issues with Naive Comet Implementations......................................................... 47
  Request Limits in the Browser ........................................................................... 48
  Server-Side Performance Concerns ............................................................... 50
  Network Infrastructure ................................................................................... 50

Summary............................................................................................................. 51

Chapter 4: Comet the Easy Way........................................................................ 53

The Emergence of Comet Tools......................................................................... 53

Direct Web Remoting........................................................................................ 54
  DWR in Action.................................................................................................. 55
  DWRServlet .................................................................................................... 58

DWR and Comet.................................................................................................. 62
  Magnetic Poetry Meets DWR on the Client Side ............................................. 62
  Magnetic Poetry and DWR on the Server Side .............................................. 64
  Routing Magnetic Poetry Events................................................................. 66
  Wrapping Up This Implementation ............................................................. 68

Summary............................................................................................................. 69

Chapter 5: Scaling Comet in Java.................................................................... 71

Thread Management for the Web ..................................................................... 71
  wait/notify ....................................................................................................... 72
  Difficulties in Using wait/notify with Comet .............................................. 76

Jetty 6.................................................................................................................. 77
  Using Jetty Continuations ............................................................................. 77
  Understanding the Continuation Mechanism ............................................. 80
  Drawbacks of Continuations ....................................................................... 82
Comet and Reverse Ajax: The Next-Generation Ajax 2.0

by Dave Crane and Phil McCarthy

This is a small book about a big subject.

As a technology, Ajax was small enough to be described in a few sentences, but it catalyzed huge changes in how we use web technologies (and communities, and business models). The full ramifications of those changes are still unfolding.

And in the middle of this change and upheaval, along comes Comet. Comet, simply put, allows you to send information from the server to the browser without the browser having to ask for it first. That’s it—simple and itself very catalytic! Comet is still in its early days, but we believe that it’s going to have a big impact on the way the Web unfolds over the next few years.

We’re lucky enough to have our names on the front of this book, in exchange for which we spent the prescribed number of late nights in our lonely garrets, putting electronic pen to paper. However, a host of talented people behind us have made this possible. We’d like to extend our thanks to Tom Welsh, Richard Dal Porto, and Heather Lang of Apress for keeping us on schedule (and being patient when we weren’t!) and for turning our rough drafts into flowing prose. We’d also like to thank Joe Walker of the DWR project, Greg Wilkins of the Jetty Web Server project, and Dylan Schiemann of the Dojo toolkit for answering our questions and for being generally supportive of our efforts to write this book—and, of course, for their broader support of the Ajax and Comet communities and turning out such interesting Open Source code in the first place.

Our friends, families, colleagues, and household pets have also been extremely patient and understanding, and we’d like to thank everyone in the Crane and McCarthy households, at Historic Futures, and Skillsmatter for their support.
Chapter 1: What Are Comet and Reverse Ajax?

The term “Comet” was coined by Alex Russell of the Dojo project to describe exchanges between a client and a server in which the server, rather than the client, initiates the contact. Joe Walker of the Direct Web Remoting (DWR) project refers to a similar mechanism as “Reverse Ajax.” Much like when the term “Ajax” was coined in 2005, the name “Comet” has served as a rallying point around a number of previously disconnected technological projects, such as the nonblocking I/O introduced into Java in 2002, message queue technologies, and, further back, HTTP 1.1’s persistent connections and the push technologies of the late 1990s.

These technologies have in common an interest in initiating communication between a client and a server from the server’s end. Conventional web-based applications are all about client-led communication, but there has been a repeated need to discuss server-led communication within the web development community and to provide a name for it. To understand the phenomenon of Comet and Reverse Ajax, we need to consider why there is a need for it and why it is so out of the ordinary as to require a label of its own.

In this short book, you’re going to address two tasks. You’re going to learn the techniques being used to deliver Comet and Reverse Ajax in today’s cutting-edge web toolkits. You’re also going to cut your way through the various tangled incarnations of Comet, Reverse Ajax, and push to figure out why developers persist in trying to turn the HTTP request-response sequence on its head. What business need is there that only Comet (a.k.a. Reverse Ajax) can deliver? And is Comet always the best way to meet these needs?
WHAT IS AJAX?

We’re assuming that you’ve heard the term “Ajax” before, but for those in need of a pithy, twenty-second, concise definition, here goes.

In most web applications, the client sends a request to the server only when refreshing the entire page, that is, all of the visual real estate that the user is looking at. “Ajax” refers to the ability to programmatically talk to the server in the background, without triggering a full-screen refresh.

Ajax may not sound like a big deal, but it’s had big implications for the design of web applications, their workflow and usability, and even their business models. We are interested in Comet partly because we believe that it will have a similarly catalytic effect on the evolution of web-based technologies.

The Trouble with HTTP

To understand Comet, first you need to understand HTTP. As web developers, we’re all somewhat familiar with HTTP—mostly as a part of the infrastructure that we take for granted and generally don’t need to pay much attention to. Let’s stop to give it our full attention for a moment.

HTTP was designed as a protocol for retrieving documents from remote servers, as illustrated in Figure 1-1. As such, it has two important characteristics:

- Communication between the client and the server is always initiated by the client and never by the server.
- Connections between the client and server are transient, and the server does not maintain any long-term state information regarding the client.

At least, this was the state of play with version 1.0 of the HTTP specification. By version 1.1, more application-like features, such as
conversational state and persistent connections, were being talked about. We’ll get to those shortly.

*Figure 1-1. In a conventional HTTP request and response, the client initiates the communication.*

Comet challenges that first assumption and allows the server to decide when it should contact the client, as illustrated in Figure 1-2. According to the ground rules of HTTP then, Comet is already kicking up a storm.

*Figure 1-2. In a Comet or Reverse Ajax exchange communication is initiated by the server.*
Figure 1-2 illustrates a single Comet exchange between server and client. Unlike the classic HTTP exchange depicted in Figure 1-1, the communication is only one-way, and it starts at the server. This is analogous to Steps 3 and 4 in Figure 1-1. Typically, the client won’t respond to the server as part of this exchange, although within the larger life cycle of the application, the client will probably also talk to the server by initiating conventional HTTP requests.

Although Comet doesn’t agree with HTTP, a number of workarounds can be used to implement Comet. In fact, we shouldn’t really be bothered about breaking the ground rules of HTTP at all. If you look at the second rule stated previously, you can see that that is challenged by another common piece of infrastructure that we take for granted, namely the HTTP session. HTTP was never designed to preserve conversational state on the server, and in fact, the continuity of a session is ensured by the client (by passing a header or cookie to the server every time it makes a request to remind the server who it is).

At the time of its introduction, the HTTP session was seen as a clever hack of the HTTP model and as a catalyst that opened up many new use cases for the web, spawning the first generation of web applications. The concept of HTTP sessions is now well supported by all but the simplest of web servers and by all major web programming tools. Perhaps in time, Comet will become a standard part of the infrastructure that we can take for granted too. As you’ll see in Chapters 6 and 7, work is already underway in reengineering web servers to better support Comet. For now, though, know that Reverse Ajax will suffice, so let’s consider the reasons why you want to make use of this technique.
Some Common Use Cases

Let’s assume for now that Comet can be made to work. Before starting to look at the technical details, we should perhaps ask why you’re considering Comet at all. As you’ll see in Chapter 2, there are several technical ways to address the problem, and you need to understand the nature of the problem correctly in order to pick the most suitable solution. Why, then, should you want the server to be able to contact the client? There are, in fact, several common use cases, so let’s look at each one in turn.

Monitoring and Data Feeds

Most applications are designed to let the user actively engage with a domain model, for instance, by querying and updating it. On a desktop PC, applications that interact with the domain model include word processors, spreadsheets, file system browsers, and most of the functionality of e-mail clients. On the web, we include e-commerce applications and search engines in this category.

However, in a smaller but important class of application, the domain model is active, and the client takes on the role of a dashboard or monitor. E-mail clients function this way when they automatically check for new mail, as do utilities such as battery monitors. Within vertical industries, there is often strong demand for monitoring applications of this type, including applications to monitor specialized hardware in science/engineering and security applications, and stock ticker and other market data feeds in the financial arena. Message queue technologies, a standard part of the enterprise developer’s toolkit, have been developed around these types of applications.

If we were to sketch the communication pattern between client and server for such an application, we might come up with something very similar to Figure 1-2.
Progress Updates

A second category in which Comet has a useful role to play is communicating progress on long-running server-side activities. In most web applications, contact with the server initiates server-side activity that is relatively brief, typically the execution of some business logic followed by a commit of the results to a database. In these cases, it is reasonable to make the user wait until the activity is completed before offering any feedback.

In some situations, however, contacting the server will initiate a longer running process. In this case, the process is best executed in a different thread, as illustrated in Figure 1-3. In this case, the user ought to be kept up to date as the long-running process unfolds, and the server may need to send several messages up to the client, possibly stating what percentage of the task is complete or listing key milestones.
When reporting progress on a long-running server-side task, the connection may be kept open while the task executes, with response data being drip-fed to the client as significant milestones are reached.
Chat and Collaboration

In the applications that we have described so far, a single user has sole access to the domain model. While this is still true of the majority of desktop applications, on the web, multiple users frequently share a larger domain model (e.g., e-commerce and photo-sharing sites and chat systems). In these types of applications, the majority of traffic between client and server is still client-driven, but situations will arise in which one user has modified the shared model in such a way that it will affect other users’ views of the model, as illustrated in Figure 1-4.

Figure 1-4: Mixing conventional Ajax and Reverse Ajax in a collaborative application
The sequence of events in this situation combines conventional Ajax HTTP calls with reverse Ajax. When one user submits an update, in a client-initiated exchange, the server may decide that other clients need to receive that update immediately. Reverse Ajax is then used to communicate these updates.

You don’t always need Comet to deal with this situation. If the urgency of communicating the changes to the other users is low, you can simply wait for them to refresh their views and issue a warning if they try to commit updates that are no longer appropriate. Alternately, you may elect to notify them by an alternate route, such as sending e-mail.

These approaches may work for photo-sharing sites, for example, in which the timing of receiving an update is not critical. However, in other collaborative applications, for example, live chat systems and auctions, the entire workflow depends on instantaneous updates, so Comet has a significant role to play.

Summary

We’ve outlined three common scenarios in web application development in which we perceive a need for Comet. In the next chapter, you’ll look at ways of implementing Comet and see how they fit the requirements that we’ve outlined here.
Chapter 2: Simple Ways to Achieve Push

In Chapter 1, we identified three common use cases that could benefit from using Comet. In this chapter, we’ll cover some simple techniques that might address these use cases, without having to resort to Comet. In Chapter 3, you’ll move on to look at simple implementations of Comet itself. If you want to really understand Comet, then you’ll need to evaluate the alternatives and recognize the situations in which Comet is the best solution.

The Magnetic Poetry Application

As you’re starting to delve into the nitty-gritty aspects of coding at this point, an example application would be useful. The application that you’ll work with in this section (and through much of this book) is an online version of a magnetic ’fridge poetry set, in which words can be placed onto a surface and rearranged to make (hopefully) humorous or insightful phrases.

To add a Web 2.0–style twist to our application, we’ve decided to share the workspace among all users who are logged on. In terms of the use cases described in the “Common Use Cases” section of Chapter 1, you’re creating a collaborative application in which multiple users will be manipulating a shared domain model at the same time.

You’ll see the implementation details of our application in more detail as we proceed. For now, Figure 2-1 presents a screenshot of the application.