F5 White Paper

Cookies, Sessions, and Persistence

Cookies and sessions are the most useful hack invented, allowing HTTP to become stateful and applications to work on the web. But it is persistence that ties the two together and makes the web what it is today.

by Lori MacVittie
Technical Marketing Manager, Application Services

Contents

Introduction

HTTP (HyperText Transfer Protocol) was designed to support a stateless, request-response model of transferring data from a server to a client. Its first version, 1.0, supported a purely 1:1 request to connection ratio (that is, one request-response pair was supported per connection).

Its second version, 1.1, expanded that ratio to be N:1—that is, many requests per connection. This was done in order to address the growing complexity of web pages, including of the many objects and elements that need to be transferred from the server to the client.

Somewhere along the line, HTTP became more than just a simple mechanism for transferring text and images from a server to a client; it became a platform for applications. The ubiquity of the browser, cross-platform nature, and ease with which applications could be deployed without the heavy cost of supporting multiple operating systems and environments was certainly appealing.

Unfortunately, HTTP was not designed to be an application transport protocol. It was designed to transfer documents. Despite the fact that both documents and application protocols are generally text-based, the resemblance ends there. Applications require some way to maintain their state, while documents do not. Applications are built on logical flows and processes, both of which require that the application know where the user is during this time, and that requires state.

HTTP is stateless, so it seems obvious that HTTP would not be appropriate for delivering applications. But it has become the de facto application transport protocol of the web. In what is certainly one of the most widely accepted, useful hacks in technical history, HTTP was given the means by which state could be tracked throughout the use of an application. That "hack" is where sessions and cookies come into play.

Sessions

Transforming the Stateless into the Stateful

Sessions are the way in which web and application servers maintain state. These simple chunks of memory are associated with every TCP connection made to a web or application server, and serve as in-memory storage for information in HTTP-based applications.

When a user connects to a server for the first time, a corresponding session is created and associated with the connection. Developers then use that session as a place to store bits of application-relevant data. This data can range from important information such as a customer ID to less consequential data such as how you like to see the front page of the site displayed.

The best example of the usefulness of sessions is shopping carts, because nearly all of us have shopped online at one time or another. Items in a shopping cart remain over the course of a "session" because every item in your shopping cart is represented in some way in the session on the server. Another good example is wizard-style product configuration or customization applications. These "mini" applications enable you to browse a set of options and select them; at the end, you are usually shocked by the estimated cost of all the bells and whistles you added. As you click through each "screen of options," the other options you chose are stored in the session so they can be easily retrieved, added, or deleted.

The problem is that sessions are tied to connections, and also to connections left idle for too long time out. Also, the definition of "too long" for connections is quite a bit different than when it is applied to sessions. The default configuration for Apache, for example, is to close a connection once it has been idle—that is, no more requests have been made—for 15 seconds. Conversely, a session in Apache, by default, will remain in memory for 300 seconds, or 5 minutes. Obviously the two are at odds with one another, because once the connection times out, what good is the session if it's associated with the connection?

You might think you could simply increase the connection time-out value to match the session and address this disparity. Increasing the time-out means that you're potentially going to expend memory to maintain a connection that may or may not be used. This can decrease the total concurrent user capacity of your server as well as ultimately impede its performance. And you certainly don't want to decrease the session timeout to match the connection time out, because most people take more than five minutes to shop around or customize their new toy.

Thus, what you end up with is sessions that remain as memory on the server even after its associated connection has been terminated due to inactivity, chewing up valuable resources and potentially angering users for whom your application just doesn't work.

Luckily, this problem is solved through the use of cookies.

Cookies

The Trail of Crumbs Leads Home

Cookies are bits of data stored on the client by the browser. Cookies can, and do, store all sorts of interesting tidbits about you, your applications, and the sites you visit. The term "cookie" is derived from "magic cookie," a well-known concept in UNIX computing that inspired both the idea and the name. Cookies are created and shared between the browser and the server via the HTTP Header, Cookie.

Cookie: `JSESSIONID=9597856473431`
Cache-Control: `no-cache`
Host: `127.0.0.2:8080`
Connection: `Keep-Alive`

The browser automatically knows it should store the cookie in the HTTP header in a file on your computer, and it keeps track of cookies on a per domain basis. The cookies for any given domain are always passed to the server by the browser in the HTTP headers, so developers of web applications can retrieve those values simply by asking for them on the server-side of the application.

The way in which the session/connection length problem is solved is through a cookie. Almost all modern web applications generate a "session ID" and pass it along as a cookie. This enables the application to find the session on the server even after the connection from which the session was created is closed. Through this exchange of session IDs, state is maintained even for a stateless protocol like HTTP.

But what happens when the use of a web application outgrows the capability of a single web or application server? Usually a load balancer, or in today's architectures an Application Delivery Controller (ADC), is introduced in order to scale the application such that all users are satisfied with the availability and performance.

The problem with this is that load balancing algorithms are generally concerned only with distributing requests across servers. Load balancing techniques are based on industry standard algorithms like round robin, least connections, or fastest response time. None of them are stateful, and it is possible for the same user to have each request made to an application be distributed to a different server. This makes all the work done to implement state for HTTP useless, because the data stored in one server's session is rarely shared with other servers in the "web farm."

This is where the concept of persistence comes in handy.

Persistence

The Tie that Binds

Persistence— otherwise known as stickiness—is a technique implemented by Application Delivery Controllers that ensures requests from a single user are always distributed to the server on which they started.

Persistence has long been used in load balancing SSL-enabled sites because once the negotiation process—a compute intensive one—has been completed and keys exchanged it would significantly degrade performance to start the process again. Thus, ADCs implemented SSL session persistence to ensure that users were always directed to the same server to which they first connected.

With the release of Microsoft Internet Explorer 5.01 , SSL persistence was suddenly rendered inoperable. The browser would automatically renegotiate SSL sessions every two minutes, necessarily changing the session ID and making it impossible to persist connections based on the session ID. The "fix" supplied by Microsoft required changes to the registry, something most burgeoning e-commerce sites considered a non-viable solution for their customers. It was therefore necessary to find an alternative solution in order to continue to support persistence. Load balancing vendors came to the rescue and offered a viable workaround: cookie-based persistence.

Rather than rely on the SSL session ID, the load balancer would insert a cookie to uniquely identify the session the first time a client accessed the site and then refer to that cookie in subsequent requests to persist the connection to the appropriate server.

The concept of cookie-based persistence has since been applied to application sessions, using session ID information generated by web and application servers to ensure that user requests are always directed to the same server during the same session. Without this capability, applications requiring load balancing would need to find another way to share session information or resort to increasing session and connection time outs to the point that the number of servers needed to support its user base would quickly grow unmanageable.

Although the most common form of persistence is implemented using session IDs passed in the HTTP header, ADCs today can persist on other pieces of data as well. Any data that can be stored in a cookie or derived from the IP, TCP, or HTTP headers can be used to persist a session. In fact, any data within an application message that uniquely identifies the user can be used by an intelligent ADC to persist a connection between the browser and a server.

Conclusion

HTTP may be a stateless protocol, but we have managed through the use of technology to force-fit state into the ubiquitous protocol. Through the use of persistence and Application Delivery Controllers, it is possible to architect highly available, performant web applications without breaking the somewhat brittle integration of cookies and sessions required to maintain state.

These features are what give HTTP state, though its implementation and execution remains stateless. Without cookies, sessions, and persistence, we surely would have found a stateful protocol on which to build our applications. Instead, features and functionality found in Application Delivery Controllers mediate between browsers (clients) and servers to provide this functionality, extending the useful of HTTP beyond static web pages to the Rich Internet Applications (RIA) of today.

F5 Networks, Inc.
Corporate Headquarters
401 Elliott Avenue West
Seattle, WA 98119
+1-206-272-5555 Phone
(888) 88BIGIP Toll-free
+1-206-272-5556 Fax
www.f5.com
info@f5.com

F5 Networks
Asia-Pacific
+65-6533-6103 Phone
+65-6533-6106 Fax
info.asia@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
+44 (0) 1932 582 000 Phone
+44 (0) 1932 582 001 Fax
emeainfo@f5.com

F5 Networks
Japan K.K.
+81-3-5114-3200 Phone
+81-3-5114-3201 Fax
info@f5networks.co.jp

WP-COOKIES-SESSIONS-PERSISTENCE 07/08