9.1 The Need for Session Tracking

HTTP is a "stateless" protocol: each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server does not automatically maintain contextual information about the client. Even with servers that support persistent (keep-alive) HTTP connections and keep sockets open for multiple client requests that occur in rapid succession, there is no built-in support for maintaining contextual information. This lack of context causes a number of difficulties. For example, when clients at an online store add an item to their shopping carts, how does the server know what's already in the carts? Similarly, when clients decide to proceed to checkout, how can the server determine which previously created shopping carts are theirs? These questions seem very simple, yet, because of the inadequacies of HTTP, answering them is surprisingly complicated.

There are three typical solutions to this problem: cookies, URL rewriting, and hidden form fields. The following subsections quickly summarize what would be required if you had to implement session tracking yourself (without using the built-in session-tracking API) for each of the three ways.

Cookies

You can use cookies to store an ID for a shopping session; with each subsequent connection, you can look up the current session ID and then use that ID to extract information about that session from a lookup table on the server machine. So, there would really be two tables: one that associates session IDs with user tables, and the user tables themselves that store user-specific data. For example, on the initial request a servlet could do something like the following:

```
String sessionID = makeUniqueString();
HashMap sessionInfo = new HashMap();
HashMap globalTable = findTableStoringSessions();
globalTable.put(sessionID, sessionInfo);
Cookie sessionCookie = new Cookie("JSESSIONID", sessionID);
sessionCookie.setPath("/");
response.addCookie(sessionCookie);
```

Then, in later requests the server could use the `globalTable` hash table to associate a session ID from the `JSESSIONID` cookie with the `sessionInfo` hash table of user-specific data.

Using cookies in this manner is an excellent solution and is the most widely used approach for session handling. Still, it is nice that servlets have a higher-level API that handles all this plus the following tedious tasks:

- Extracting the cookie that stores the session identifier from the other cookies (there may be many cookies, after all).
- Determining when idle sessions have expired, and reclaiming them.
- Associating the hash tables with each request.
- Generating the unique session identifiers.

URL Rewriting

With this approach, the client appends some extra data on the end of each URL. That data identifies the session, and the server associates that identifier with user-specific data it has

stored. For example, with `http://host/path/file.html;jsessionid=a1234`, the session identifier is attached as `jsessionid=a1234`, so `a1234` is the ID that uniquely identifies the table of data associated with that user.

URL rewriting is a moderately good solution for session tracking and even has the advantage that it works when browsers don't support cookies or when the user has disabled them. However, if you implement session tracking yourself, URL rewriting has the same drawback as do cookies, namely, that the server-side program has a lot of straightforward but tedious processing to do. Even with a high-level API that handles most of the details for you, you have to be very careful that every URL that references your site and is returned to the user (even by indirect means like `Location` fields in server redirects) has the extra information appended. This restriction means that you cannot have any static HTML pages on your site (at least not any that have links back to dynamic pages at the site). So, every page has to be dynamically generated with servlets or JSP. Even when all the pages are dynamically generated, if the user leaves the session and comes back via a bookmark or link, the session information can be lost because the stored link contains the wrong identifying information.

Hidden Form Fields

As discussed in Chapter 19 (Creating and Processing HTML Forms), HTML forms can have an entry that looks like the following:

```
<INPUT TYPE="HIDDEN" NAME="session" VALUE="a1234">
```

This entry means that, when the form is submitted, the specified name and value are automatically included in the `GET` or `POST` data. This hidden field can be used to store information about the session but has the major disadvantage that it only works if every page is dynamically generated by a form submission. Clicking on a regular (`<A HREF...>`) hypertext link does not result in a form submission, so hidden form fields cannot support general session tracking, only tracking within a specific series of operations such as checking out at a store.

Session Tracking in Servlets

Servlets provide an outstanding session-tracking solution: the `HttpSession` API. This high-level interface is built on top of cookies or URL rewriting. All servers are required to support session tracking with cookies, and most have a setting by which you can globally switch to URL rewriting.

Either way, the servlet author doesn't need to bother with many of the implementation details, doesn't have to explicitly manipulate cookies or information appended to the URL, and is automatically given a convenient place to store arbitrary objects that are associated with each session.