8.4 Sending and Receiving Cookies

To send cookies to the client, a servlet should use the `Cookie` constructor to create one or more cookies with designated names and values, set any optional attributes with `cookie.set`*Xxx* (readable later by `cookie.get`*Xxx*), and insert the cookies into the HTTP response headers with `response.addCookie`.

To read incoming cookies, a servlet should call `request.getCookies`, which returns an array of `Cookie` objects corresponding to the cookies the browser has associated with your site (`null` if there are no cookies in the request). In most cases, the servlet should then loop down this array calling `getName` on each cookie until it finds the one whose name matches the name it was searching for, then call `getValue` on that `Cookie` to see the value associated with the name. Each of these topics is discussed in more detail in the following subsections.

Sending Cookies to the Client

Sending cookies to the client involves three steps (summarized below with details in the following subsections).

1. **Creating a `Cookie` object.** You call the `Cookie` constructor with a cookie name and a cookie value, both of which are strings.
2. **Setting the maximum age.** If you want the browser to store the cookie on disk instead of just keeping it in memory, you use `setMaxAge` to specify how long (in seconds) the cookie should be valid.
3. **Placing the `Cookie` into the HTTP response headers.** You use `response.addCookie` to accomplish this. If you forget this step, no cookie is sent to the browser!

Creating a Cookie Object

You create a cookie by calling the `Cookie` constructor, which takes two strings: the cookie name and the cookie value. Neither the name nor the value should contain white space or any of the following characters:

```
 [ ] ( ) = , " / ? @ : ;
```

For example, to create a cookie named `userID` with a value `a1234`, you would use the following.

```
Cookie c = new Cookie("userID", "a1234");
```

Setting the Maximum Age

If you create a cookie and send it to the browser, by default it is a session-level cookie: a cookie that is stored in the browser's memory and deleted when the user quits the browser. If you want the browser to store the cookie on disk, use `setMaxAge` with a time in seconds, as below.

```
c.setMaxAge(60*60*24*7); // One week
```

Since you could use the session-tracking API (Chapter 9) to simplify most tasks for which you use session-level cookies, you almost always use the `setMaxAge` method when using the `Cookie` API.

Setting the maximum age to 0 instructs the browser to delete the cookie.

Core Approach

When you create a `Cookie` object, you should normally call `setMaxAge` before sending the cookie to the client.

Note that `setMaxAge` is not the only `Cookie` characteristic that you can modify. The other, less frequently used characteristics are discussed in Section 8.6 (Using Cookie Attributes).

Placing the Cookie in the Response Headers

By creating a `Cookie` object and calling `setMaxAge`, all you have done is manipulate a data structure in the server's memory. You haven't actually sent anything to the browser. If you don't send the cookie to the client, it has no effect. This may seem obvious, but a common mistake by beginning developers is to create and manipulate `Cookie` objects but fail to send them to the client.

Core Warning

Creating and manipulating a `Cookie` object has no effect on the client. You must explicitly send the cookie to the client with `response.addCookie`.

To send the cookie, insert it into a `Set-Cookie` HTTP response header by means of the `addCookie` method of `HttpServletResponse`. The method is called `addCookie`, not `setCookie`, because any previously specified `Set-Cookie` headers are left alone and a new header is set. Also, remember that the response headers must be set before any document content is sent to the client.

Here is an example:

```
Cookie userCookie = new Cookie("user", "uid1234");
userCookie.setMaxAge(60*60*24*365); // Store cookie for 1 year
response.addCookie(userCookie);
```

Reading Cookies from the Client

To send a cookie *to* the client, you create a `Cookie`, set its maximum age (usually), then use `addCookie` to send a `Set-Cookie` HTTP response header. To read the cookies that come back *from* the client, you should perform the following two tasks, which are summarized below and then described in more detail in the following subsections.

1. **Call `request.getCookies`.** This yields an array of `Cookie` objects.

2. **Loop down the array, calling `getName` on each one until you find the cookie of interest.** You then typically call `getValue` and use the value in some application-specific way.

Call request.getCookies

To obtain the cookies that were sent by the browser, you call getCookies on the HttpServletRequest. This call returns an array of Cookie objects corresponding to the values that came in on the Cookie HTTP request headers. If the request contains no cookies, getCookies should return null. Note, however, that an old version of Apache Tomcat (version 3.x) had a bug whereby it returned a zero-length array instead of null.

Loop Down the Cookie Array

Once you have the array of cookies, you typically loop down it, calling getName on each Cookie until you find one matching the name you have in mind. Remember that cookies are specific to your host (or domain), not your servlet (or JSP page). So, although your servlet might send a single cookie, you could get many irrelevant cookies back. Once you find the cookie of interest, you typically call getValue on it and finish with some processing specific to the resultant value. For example:

```
String cookieName = "userID";
Cookie[] cookies = request.getCookies();
if (cookies != null) {
  for(int i=0; i<cookies.length; i++) {
    Cookie cookie = cookies[i];
    if (cookieName.equals(cookie.getName())) {
      doSomethingWith(cookie.getValue());
    }
  }
}
```

This is such a common process that, in Section 8.8, we present two utilities that simplify retrieving a cookie or cookie value that matches a designated cookie name.