9.3 The Session-Tracking API

Although the session attributes (i.e., the user data) are the pieces of session information you care most about, other information is sometimes useful as well. Here is a summary of the methods available in the `HttpSession` class.

public Object getAttribute(String name)

This method extracts a previously stored value from a session object. It returns `null` if no value is associated with the given name.

public Enumeration getAttributeNames()

This method returns the names of all attributes in the session.

public void setAttribute(String name, Object value)

This method associates a value with a name. If the object supplied to `setAttribute` implements the `HttpSessionBindingListener` interface, the object's `valueBound` method is called after it is stored in the session. Similarly, if the previous value implements `HttpSessionBindingListener`, its `valueUnbound` method is called.

public void removeAttribute(String name)

This method removes any values associated with the designated name. If the value being removed implements `HttpSessionBindingListener`, its `valueUnbound` method is called.

public void invalidate()

This method invalidates the session and unbinds all objects associated with it. Use this method with caution; remember that sessions are associated with users (i.e., clients), not with individual servlets or JSP pages. So, if you invalidate a session, you might be destroying data that another servlet or JSP page is using.

public void logout()

This method logs the client out of the Web server and invalidates *all* sessions associated with that client. The scope of the logout is the same as the scope of the authentication. For example, if the server implements single sign-on, calling `logout` logs the client out of all Web applications on the server and invalidates all sessions (at most one per Web application) associated with the client. For details, see the chapters on Web application security in Volume 2 of this book.

public String getId()

This method returns the unique identifier generated for each session. It is useful for debugging or logging or, in rare cases, for programmatically moving values out of memory and into a database (however, some J2EE servers can do this automatically).

public boolean isNew()

This method returns `true` if the client (browser) has never seen the session, usually

because the session was just created rather than being referenced by an incoming client request. It returns `false` for preexisting sessions. The main reason for mentioning this method is to steer you away from it: `isNew` is much less useful than it appears at first glance. Many beginning developers try to use `isNew` to determine whether users have been to their servlet before (within the session timeout period), writing code like the following:

```
HttpSession session = request.getSession();
if (session.isNew()) {
  doStuffForNewbies();
} else {
  doStuffForReturnVisitors();  // Wrong!
}
```

Wrong! Yes, if `isNew` returns `true`, then as far as you can tell this is the user's first visit (at least within the session timeout). But if `isNew` returns `false`, it merely shows that they have visited the Web application before, not that they have visited your servlet or JSP page before.

public long getCreationTime()

This method returns the time in milliseconds since midnight, January 1, 1970 (GMT) at which the session was first built. To get a value useful for printing, pass the value to the `Date` constructor or the `setTimeInMillis` method of `GregorianCalendar`.

public long getLastAccessedTime()

This method returns the time in milliseconds since midnight, January 1, 1970 (GMT) at which the session was last accessed by the client.

public int getMaxInactiveInterval()

public void setMaxInactiveInterval(int seconds)

These methods get or set the length of time, in seconds, that a session should go without access before being automatically invalidated. A negative value specifies that the session should never time out. Note that the timeout is maintained on the server and is *not* the same as the cookie expiration date. For one thing, sessions are normally based on in-memory cookies, not persistent cookies, so there *is* no expiration date. Even if you intercepted the `JSESSIONID` cookie and sent it out with an expiration date, browser sessions and server sessions are two distinct things. For details on the distinction, see the next section.