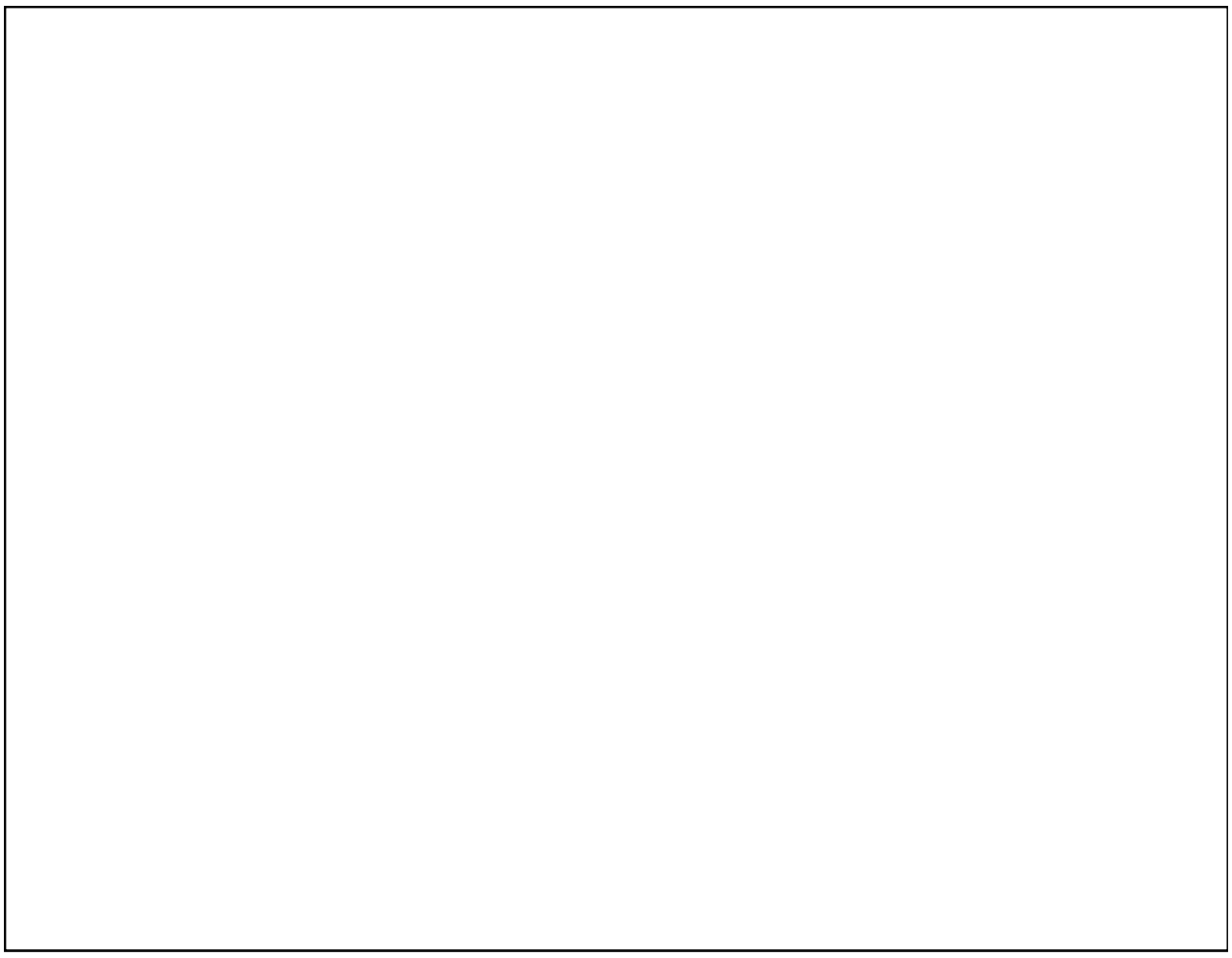
Sessions

- Why use sessions?
- Using sessions
- Cookies and URL rewriting

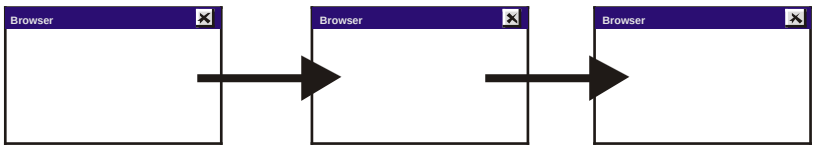
8 - 1

In this chapter, you will learn about various techniques so that your servlets can maintain state between requests.



HTTP is a Stateless Protocol

- Unlike FTP or Telnet, HTTP conversations do not maintain a long-lived connection between the client and server
- That's great for normal Web browsing, but not good for Web applications that need to maintain state between conversations (e.g. a shopping cart or a secure login)



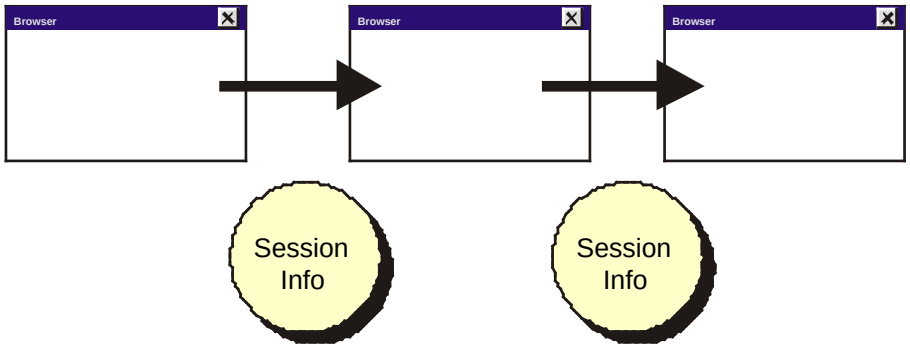
8 - 2

HTTP was originally designed to be a simple protocol, so the connection between browsers and the web server is transient -- essentially, the browser must open a new connection for each request. This is different than with Telnet for example, where the client and server maintain a long-lived connection.

In the early days of the Web, HTTP's statelessness was not an issue. But e-commerce and related applications need to maintain state across requests. The classic example is the shopping cart, which must "remember" its contents until the order is complete.

What is a Session?

- A session is a series of requests from the same user, running the same browser



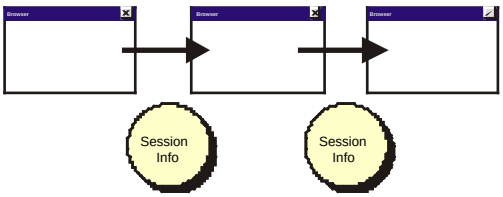
8 - 3

We will define a session as being a series of requests from the same browser instance. Note that if the user runs two different browsers, that's not considered a session, even if the browsers are accessing the same site. This definition also excludes the user running browsers on different computers to the same Web site.

For a session to work, the application must somehow remember the session information and be able to connect it with subsequent requests from the same browser instance.

Remembering State

- To manage sessions, Web programmers have developed a few techniques:
  - Hidden form fields
  - URL rewriting
  - Cookies
- These techniques work, but are somewhat tedious to program, may have privacy implications and are subject to how the user configures the browser



8 - 4

The hidden form field technique requires each page to contain an HTML form with one or more fields of type "hidden". The web application uses the hidden fields to maintain state information -- when the user wants to link to a new page, the application generates the new page dynamically, complete with the state information in a form field. This technique is tedious and requires all pages to be generated dynamically, since they need to contain the state information as part of the HTML itself.

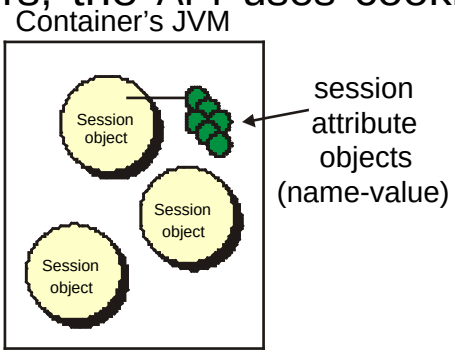
In URL rewriting, the application appends state information to the URL for each page in the application. This technique also requires dynamic page generation, since any hyperlinks in the page must be encoded on the fly to contain the state information.

Cookies are probably the most elegant solution. With cookies, the application instructs the browser to create a small file on the client's computer that contains the state information. On subsequent requests, the browser passes the information so the application can maintain state. The problem with cookies is that many people view them as an invasion of privacy, and set up their browser to reject cookies. Thus, a web application cannot always depend on cookies.

Note that a container can use other techniques to maintain state if available, for example SSL Sessions.

The Servlet Session API

- To make remembering state easier for the servlet programmer, the container provides an API that the servlet can use set or retrieve state
- For example, a login page could set the login information and other pages could retrieve that info
- Under the covers, the API uses cookies or URL rewriting



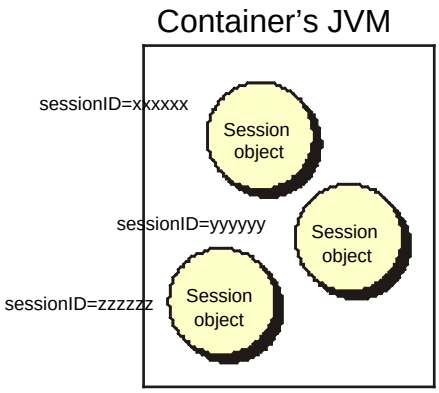
8 - 5

Instead of explicitly using the techniques shown on the last page, servlet programmers can use an API that masks most of the complexity of maintaining sessions. To use the servlet session support, servlets create objects and attach them by name to a session object that's maintained by the servlet container. When a subsequent request arrives from the same browser, the container matches the request with the appropriate session object -- the servlet can then retrieve the objects store previously.

Note that "under the covers", the API uses cookies or URL rewriting. Some containers, such as JRun, first try to use cookies and then fallback to URL rewriting if cookies don't work. While that sounds reasonable, it does require some extra programming to work as we will cover later.

How Does the Session API Work?

- Each session object is tagged with a unique *session ID*
- On each request, the container uses cookies or URL rewriting to pass the session ID
- The container can then use the sessionID to find the corresponding session object



The sessionIDs are like database key fields -- given the sessionID, the container can find the corresponding session object. If the user's browser supports cookies, the container uses a session cookie to store the ID (session cookies expire automatically when the session ends). If the URL rewriting is used, then the container, with help from the servlet, appends the sessionID as a parameter on each URL. We will see more on cookies and URL rewriting coming up.

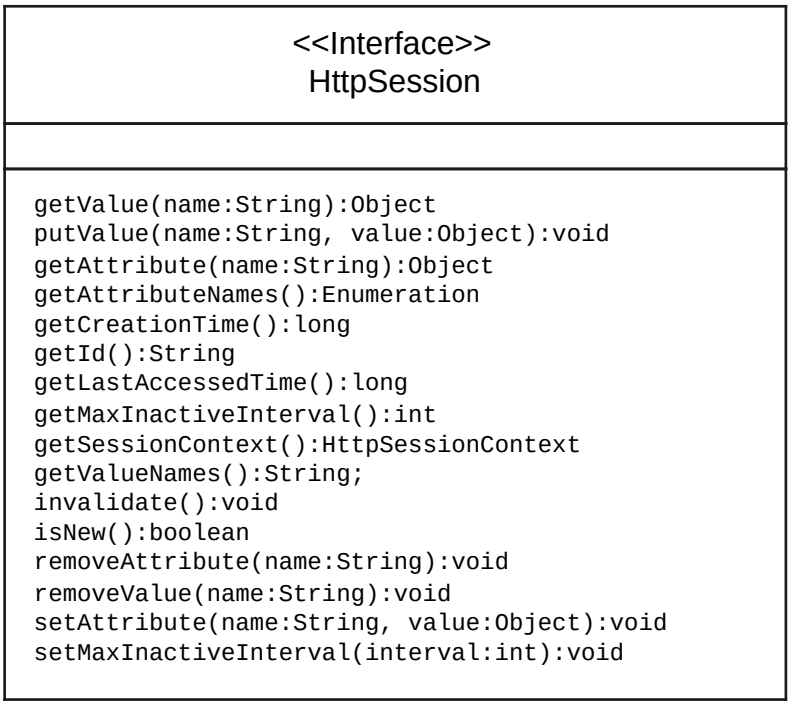
The Servlet Session API, cont'd

This page shows part of the API that let servlets maintain state.

The two most useful methods shown are the getSession methods. These methods give a servlet access to a session object, to which the servlet can attach state information. The getSession method with no arguments returns a session object, creating a new one if necessary. The other getSession method accepts a boolean. You pass true to create a new session or retrieve an existing one. If you pass false, that method returns null if there is no current session.

You can use getSession (false) to determine if there's a current session -- for example, if you use a session to maintain login information, lack of a session means that the user isn't logged in (perhaps they previously set a bookmark on a page and skipped the login page).

The Servlet Session API, cont'd



This figure shows additional APIs for manipulating sessions. HttpSession is the type of the object returned by the getSession methods shown on the previous page.

The two most useful methods here are getAttribute and setAttribute. The setAttribute method lets you attach an object by name to the session, while getAttribute lets you retrieve the object from the session, given its name.

Note that there are various APIs that let you examine or configure session timeouts, invalidate the session, and so forth.

The getValue and putValue methods are deprecated and replaced by getAttribute and setAttribute.

Creating or Retrieving a Session

- You can request a new or existing session object by calling the getSession method of the HttpServletRequest object

First Page

```
//create a new session
  HttpSession session =  request.getSession (true);
```

Subsequent Page

```
//retrieve existing session
  HttpSession session =  request.getSession (false);
  if ( session == null ) error();
```


This page shows a common pattern for using sessions in an application with multiple servlets. The first servlet creates a session and then attaches objects to it (not shown here). Subsequent servlets then retrieve the session object and then retrieve the named objects from the session (not shown).

In this application, after the first servlet, the lack of a session is considered an error. Perhaps the first servlet processes a login page that establishes the user's credentials. If the user goes directly to a subsequent page without visiting the first page, there will be no session and this program would generate an error response, perhaps redirecting the request back to the login servlet.

Saving State in a Session

- To write information into the session, use the setAttribute method
- Note that the *value* must be an Object!

```
1    int correctAnswers = 0;
2    int wrongAnswers = 0;
3    . . .
4    Integer correctInteger =
5        new Integer ( correctAnswers );
6    Integer wrongInteger =
7        new Integer ( wrongAnswers );
8
9    session.setAttribute ( "CorrectAnswers"
10                        , correctInteger );
11   session.setAttribute ( "WrongAnswers"
12                        , wrongInteger );
```

8 - 10

This fragment of a servlet is part of an online quiz-talking application. In lines 1 and 2, it defines integers that keep track of the counts of correct and incorrect answers. Then in code omitted here for brevity, it scores a quiz and updates the integer variables. Then the servlet creates two Integer objects in lines 4 to 7 and then attaches them in lines 9 through 12 to a previously retrieved session object. The servlet assigns the names CorrectAnswers and WrongAnswers to the Integer objects.

Since the data attached to a session must extend Object, this servlet uses the Integer class to "wrap" the integer primitive variables that hold the count of correct and incorrect answers.

Retrieving State from a Session

- To retrieve information from the session, use the getAttribute method

```
1    . . .
2
3    Integer correctInteger = (Integer)
4        theSession.getAttribute ( "CorrectAnswers" );
5    Integer wrongInteger =  (Integer)
6        theSession.getAttribute ( "WrongAnswers" );
7
8    int correct = correctInteger.intValue();
9    int wrong = wrongInteger.intValue();
```


This is a fragment of another servlet that's part of the quiz-taking application. This servlet retrieves the session object (no shown, but same as before) and then can retrieve the Integer wrapper objects for the count of correct and incorrect answers. Note that to retrieve the objects, this servlet uses the same names as assigned in the previous servlet.

Configuring Sessions

- Most containers let you configure session characteristics via a GUI or by editing configuration files
- For example, you can configure whether the container implements session via cookies or URL rewriting
- Many containers will attempt to use cookies and "fall back" to URL rewriting if the browser does not have cookies enabled

The way the session API is actually implemented is up to the container. Most container vendors provide tools or configuration files that let the system administrator control how sessions work. For example, if your application runs in an environment where cookies are disabled in browsers, you could configure the container to use URL rewriting instead.

Using URL Rewriting

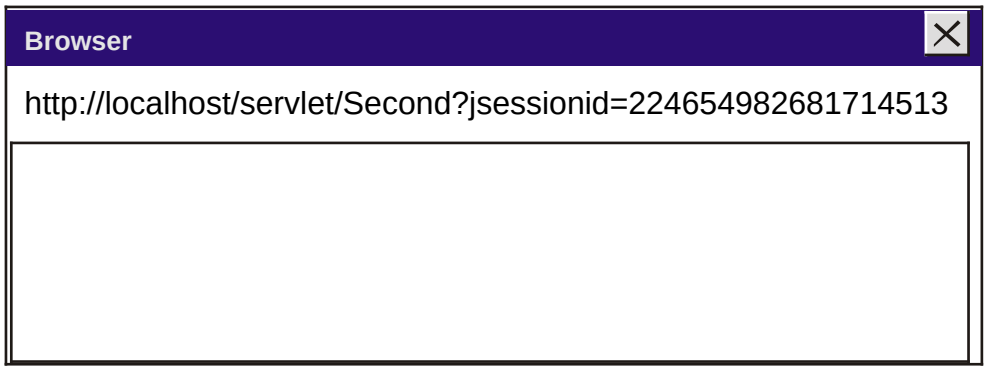
- If your user's browsers do not support cookies, then you should use URL rewriting
- To use URL rewriting, you must encode all HTML hyperlink URLs generated by servlets
- You should also encode any URLs used for redirection

```
String url =
  res.encodeURL ( "http://localhost/servlet/Second" );
out.println ( "<a href=\"" + url + "\">Next</a> " );
```


The container can use either session cookies or URL rewriting to "remember" the session ID. If cookies are used, then the servlet has no extra work to do. But if URL rewriting is required, then the servlet must ensure that all URLs in the generated HTML are encoded with the session ID. To make that easier, the servlet API provides the encodeURL and encodeRedirectedURL methods. Here we show using encodeURL to encode the URL for a hyperlink in a page generated from a servlet.

Using URL Rewriting, cont'd

- With URL rewriting, each URL has the session information appended to the actual URL



8 - 14

This figure shows what an encoded URL looks like. Note that the sessionID "magic number" is appended to the URL of this servlet, which is the second servlet in a Web application.

Using Cookies Explicitly

- Sometimes you may need to create and access cookies directly instead of using the Servlet session API
- For example, you may want to store information for a longer time period than the *session* cookies used by the API, or you might want to save info that's not associated with a session
- The Servlet API contains a rich set of methods to let you manipulate cookies directly

8 - 15

The session API uses so-called session cookies that automatically expire at the end of the session. If you need to store information on a client computer that lasts longer, you can use the cookie API we will cover next.

Note that some users will disable cookies on their browser due to privacy concerns. If so, the techniques shown here will not work.

Cookie Example: Setting a Cookie

- This example sets a cookie that will not expire for 24 hours

```
1    public void doGet ( HttpServletRequest req
2                      , HttpServletResponse res )
3        throws ServletException, IOException
4    {
5        . . .
6        String val = "123ABC";
7        Cookie cookie = new Cookie ( "mycookie"
8                                   , val );
9        cookie.setMaxAge ( 3600 * 24 );
10       res.addCookie ( cookie );
11       . . .
12   }
```


This fragment of a servlet shows setting a relatively long-lived cookie that stores a String value under the name "mycookie" (how original!)

Note that you actually add cookies to the response instead of "setting" them -- that lets you set multiple cookies with different names.

Cookie Example: Retrieving a Cookie

```
1    public void doGet ( HttpServletRequest req
2                      , HttpServletResponse res )
3        throws ServletException, IOException
4    {
5        . . .
6        String val = null;
7        Cookie[] cookies = req.getCookies();
8        for ( int i = 0; i < cookies.length; i++)
9        {
10           if ( cookies[i].getName().equals ( "mycookie" ) )
11           {
12              val = cookies[i].getValue();
13              break;
14           }
15       }
16       . . .
17   }
```

8 - 17

This fragment is the complement to the code shown on the last page -- it retrieves the cookie with name "mycookie". Actually, there's no easy way to find a cookie by name. Instead, you query an array of all cookies in the request (line 7) and then iterate, searching for the cookie by name (line 10). The servlet can then retrieve the cookie's value (line 12).

Lab Exercise

- **Title:** Lab 4: Sessions
- **Approximate time:** 45 minutes

Chapter Summary

In this chapter, you learned:

- The different ways to maintain state
- How to use the servlet session API