Chapter 7

How to work with sessions and cookies

Objectives

Applied

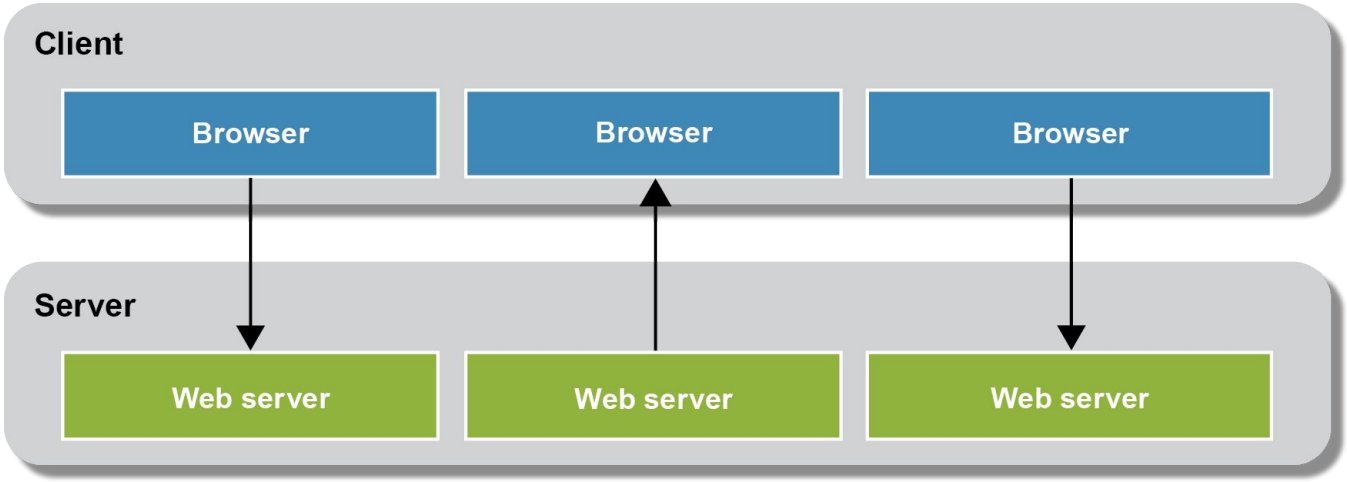
1. Provide for session tracking by using cookies.
2. Write code that stores a cookie on a user's browser, gets that cookie, and deletes that cookie.
3. Pass parameters by using URL rewriting.
4. Pass parameter by using hidden fields within a form.

Objectives (continued)

Knowledge

1. Describe the way HTTP works without session tracking.
2. Describe the way cookies are used for session tracking.
3. Describe why it's generally considered a best practice to use cookies for session tracking instead of using URL encoding.
4. Distinguish between persistent cookies and per-session cookies.
5. Distinguish between the use of URL rewriting and the use of hidden fields as ways to pass parameters.

Why session tracking is difficult with HTTP

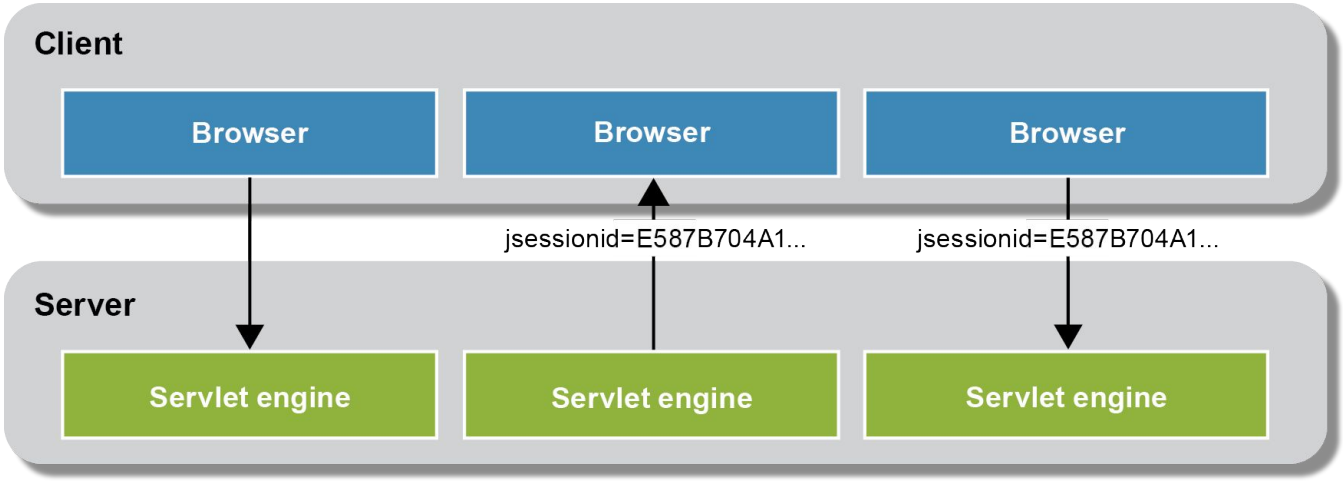


First HTTP Request:
The browser requests a page.

First HTTP Response:
The server returns the requested page and drops the connection.

Following HTTP Requests:
The browser requests a page. The web server has no way to associate the browser with its previous request.

How Java keeps track of sessions



First HTTP Request:

The browser requests a JSP or servlet. The servlet engine creates a session object and assigns an ID for the session.

First HTTP Response:

The server returns the requested page and the ID for the session.

Following HTTP Requests:

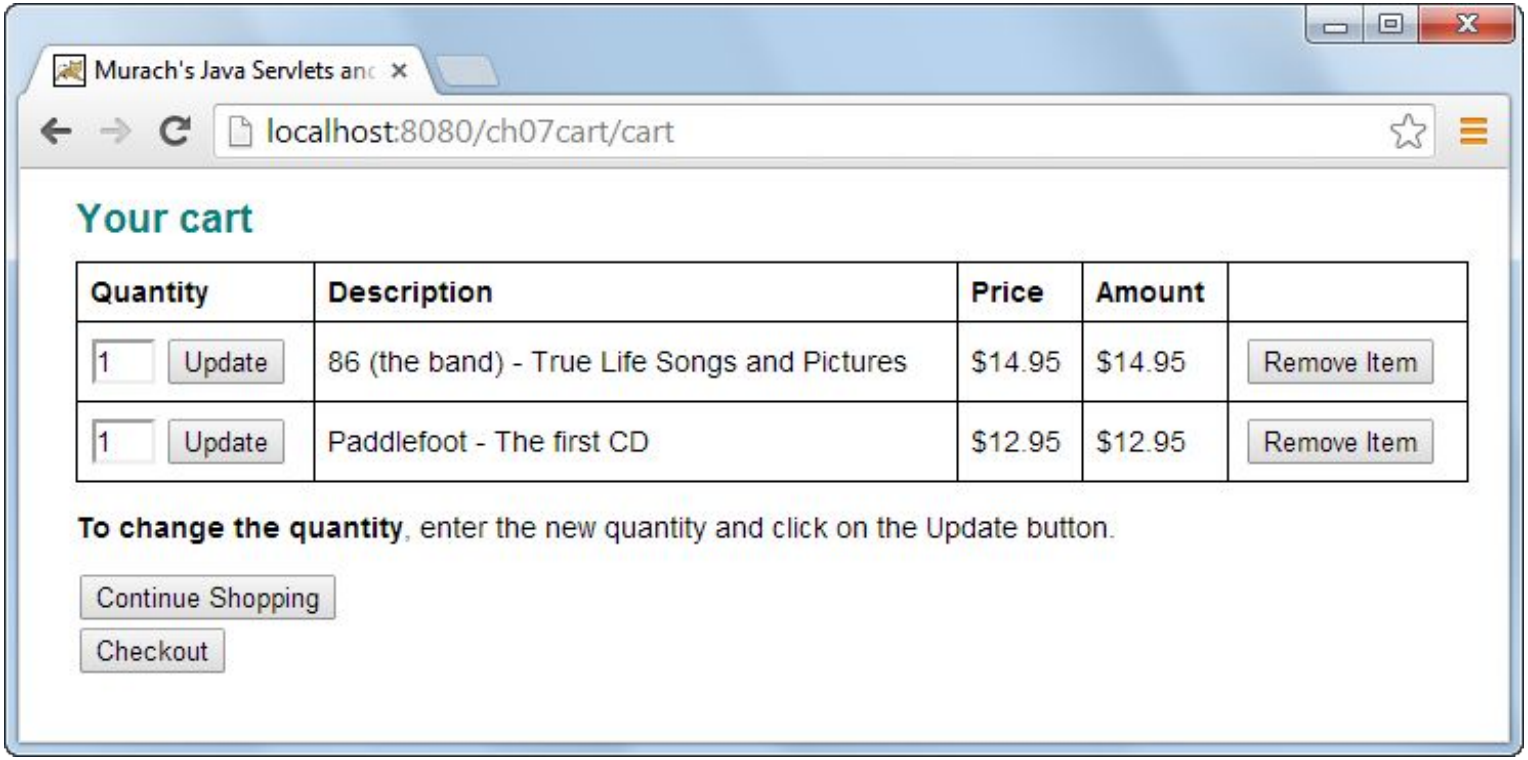
The browser requests a JSP or servlet. The servlet engine uses the session ID to associate the browser with its session object.

Session tracking and cookies

- HTTP is a *stateless protocol*. Once a browser makes a request, it drops the connection to the server. To maintain *state*, a web application must use *session tracking*.
- By default, the servlet API uses a cookie to store a session ID in each browser. The browser passes the cookie to the server with each request.
- To store the data for each session, the server creates a *session object*.
- To provide session tracking when cookies are disabled in the browser, you can use *URL encoding* to store the session ID in the URL for each page of an application. However, this is not considered a best practice.
- *Persistent cookies* are stored on the user's PC. *Per-session cookies* are deleted when the session ends.

The Index page

The Cart page



A method of the request object

Method	Description
`getSession()`	Returns the HttpSession object.

Three methods of the session object

Method	Description
`setAttribute(String name, Object o)`	Stores any object in the session as an attribute.
`getAttribute(String name)`	Returns the value of the specified attribute as an Object type.
`removeAttribute(String name)`	Removes the specified attribute from this session.

How to get a session object

```
HttpSession session = request.getSession();
```

How to set a String object as an attribute

```
session.setAttribute("productCode", productCode);
```

How to get a String object

```
String productCode = (String) session.getAttribute("productCode");
```

How to set a user-defined object as an attribute

```
Cart cart = new Cart(productCode);
session.setAttribute("cart", cart);
```

How to get a user-defined object

```
Cart cart = (Cart) session.getAttribute("cart");
if (cart == null)
    cart = new Cart();
```

How to remove an object

```
session.removeAttribute("productCode");
```

Sessions

- A session object is created when a browser makes the first request and destroyed when the session ends.
- A session ends when a specified amount of time elapses without another request or when the user exits the browser.

More methods of the session object

Method	Description
`getAttributeNames()`	Returns a java.util.Enumeration object that contains the names of all session attributes.
`getId()`	Returns a string for the unique Java session identifier.
`isNew()`	Returns a true value if the client does not yet know about the session.
`setMaxInactiveInterval(int seconds)`	By default, the maximum inactive interval for the session is set to 1800 seconds (30 minutes). To create a session that won't end until the user closes the browser, supply a negative integer such as –1.
`invalidate()`	Invalidates the session and unbinds any objects that are bound to it.

How to get the names of the attributes for a session

```
Enumeration names = session.getAttributeNames();
while (names.hasMoreElements()) {
    System.out.println((String) names.nextElement());
}
```

How to get the ID for a session

```
String jSessionId = session.getId();
```

How to set the inactive interval for a session

```
session.setMaxInactiveInterval(60*60*24);  // one day
session.setMaxInactiveInterval(-1);   // until the browser is closed
```

How to invalidate the session and unbind any objects

```
session.invalidate();
```

Three tabs accessing the same session object

How to provide thread-safe access to the session object

- Each servlet creates one session object that exists for multiple requests that come from a single client.
- If the client has one browser window open, access to the session object is thread-safe.
- If the client has multiple browser windows open, two threads from the same client could access the session object at the same time. As a result, the session object isn't thread-safe.
- Since the servlet specification doesn't guarantee that it will always return the same session object, you can't make the session object thread-safe by synchronizing on it. Instead, you can synchronize on the session ID string for the session object.

How to synchronize access to the session object

```
Cart cart;
final Object lock = request.getSession().getId().intern();
synchronized(lock) {
    cart = (Cart) session.getAttribute("cart");
}
```

How to synchronize access to the session object

```
final Object lock = request.getSession().getId().intern();
synchronized(lock) {
    session.setAttribute("cart", cart);
}
```

Examples of cookies

```
jsessionid=D1F15245171203E8670487F020544490
user_id=87
email=jsmith@hotmail.com
```

How cookies work

- A cookie is a name/value pair stored in a browser.
- A web application on the server creates a cookie and sends it to the browser. The browser on the client saves the cookie and sends it back to the server every time it accesses a page from that server.
- Cookies can be set to persist for up to 3 years.
- Some users disable cookies in their browsers.
- Browsers generally accept only 20 cookies from each site and 300 cookies total. They can limit each cookie to 4 kilobytes.
- A cookie can be associated with one or more subdomain names.

Typical uses for cookies

- **To allow users to skip login and registration forms** that gather data like user name, password, address, or credit card data.
- **To customize pages** that display information like weather reports, sports scores, and stock quotations.
- **To focus advertising** like banner ads that target the user's interests.

Cookies

- A per-session cookie that holds the session ID is automatically created for each session. It is used to relate the browser to the session object.
- You can also create and send other cookies to a user's browser, and use them to access user-specific data that's stored in a file or database.

Constructor of the Cookie class

Constructor	Description
`Cookie(String name, String value)`	Creates a cookie with the specified name and value.

The methods of the Cookie class

Method	Description
`setMaxAge(int maxAgeInSeconds)`	To create a persistent cookie, set the cookie's maximum age to a positive number. To create a per-session cookie, set the cookie's maximum age to –1.
`setPath(String path)`	To allow the entire application to access the cookie, set the cookie's path to "/".
`getName()`	Returns a string for the name of the cookie.
`getValue()`	Returns a string that contains the value of the cookie.

A method of the response object

Method	Description
`addCookie(Cookie c)`	Adds the specified cookie to the response.

A method of the request object

Method	Description
`getCookies()`	Returns an array of Cookie objects that the client sent with this request. If no cookies were sent, this method returns a null value.

How to create a cookie and set it in a response

```
Cookie c = new Cookie("userIdCookie", userId);
c.setMaxAge(60*60*24*365*2);   // set age to 2 years
c.setPath("/");                // allow access by entire app
response.addCookie(c);
```

How to get a cookie value from a request

```
Cookie[] cookies = request.getCookies();
String cookieName = "userIdCookie";
String cookieValue = "";
for (Cookie cookie: cookies) {
    if (cookieName.equals(cookie.getName()))
        cookieValue = cookie.getValue();
}
```

JSP that shows all cookies for the current server

JSP code that displays all cookies

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<table>
  <tr>
    <th>Name</th>
    <th>Value</th>
  </tr>
  <c:forEach var="c" items="${cookie}">
  <tr>
    <td>${c.value.name}</td>
    <td>${c.value.value}</td>
  </tr>
  </c:forEach>
</table>
```

Servlet code that deletes all persistent cookies

```
Cookie[] cookies = request.getCookies();
for (Cookie cookie : cookies) {
    cookie.setMaxAge(0); //delete the cookie
    cookie.setPath("/"); //allow the download application to access it
    response.addCookie(cookie);
}
```

JSP code that displays the value of a cookie

```
<p>Email cookie value: ${cookie.emailCookie.value}</p>
```

Four methods of the Cookie class

Method	Description
`setPath(String path)`	To make a cookie available to the entire application, you can set the path to a slash (/).
`setDomain(String pattern)`	By default, the browser only returns a cookie to the host that sent the cookie. To return a cookie to other hosts within the same domain, set a domain pattern like .ads.com. Then, the browser returns the cookie to any subdomain of www.ads.com like www.travel.ads.com or www.camera.ads.com.

Four methods of the Cookie class (continued)

Method	Description
`setSecure(boolean flag)`	By default, the browser sends a cookie over a regular, or encrypted, connection. To force a secure connection, supply a true value for this method.
`setVersion(int version)`	By default, Java creates cookies that use version 0 of the cookie protocol. However, you can specify an int value of 1 for this method to use the new version of the cookie protocol, which is version 1.

A utility class that gets the value of a cookie

```
package murach.util;

import javax.servlet.http.*;

public class CookieUtil {

    public static String getCookieValue(
            Cookie[] cookies, String cookieName) {

        String cookieValue = "";
        if (cookies != null) {
            for (Cookie cookie: cookies) {
                if (cookieName.equals(cookie.getName())) {
                    cookieValue = cookie.getValue();
                }
            }
        }
        return cookieValue;
    }
}
```

Code that uses the CookieUtil class to get the value of a cookie

```
Cookie[] cookies = request.getCookies();
String emailAddress = CookieUtil.getCookieValue(cookies,
     "emailCookie");
```

The syntax for URL rewriting

```
url?paramName1=paramValue1&amp;paramName2=paramValue2&amp;...
```

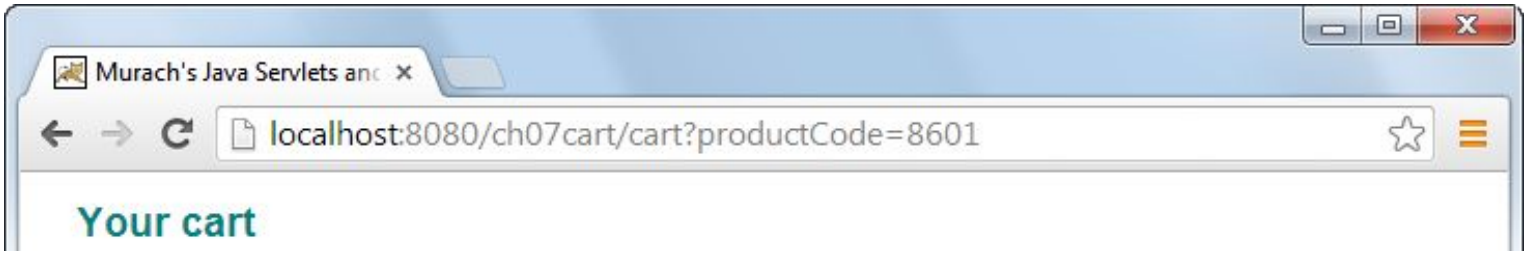
A link that adds a product code to a URL

```
<a href="cart?productCode=8601">Add To Cart</a>
```

The link displayed in a browser

86 (the band) - True Life Songs and Pictures	$14.95	Add To Cart

The URL when you click the link



More examples of URL rewriting

A form tag

```
<form action="cart?productCode=jr01" method="post">
```

A link that uses EL

```
<a href="cart?productCode=${productCode}">Add To Cart</a>
```

A link that includes two parameters

```
<a href="download?action=checkUser&amp;productCode=8601">
Download</a>
```

Two limitations of URL rewriting

- Most browsers limit the number of characters that can be passed by a URL to 2,000 characters.
- It's difficult to include spaces and special characters such as the ? and & characters in parameter values.

Two security risks with URL rewriting

- Parameter values can leak to third-party sites such as Google Analytics or Facebook.
- Parameter values are stored in the browser history.

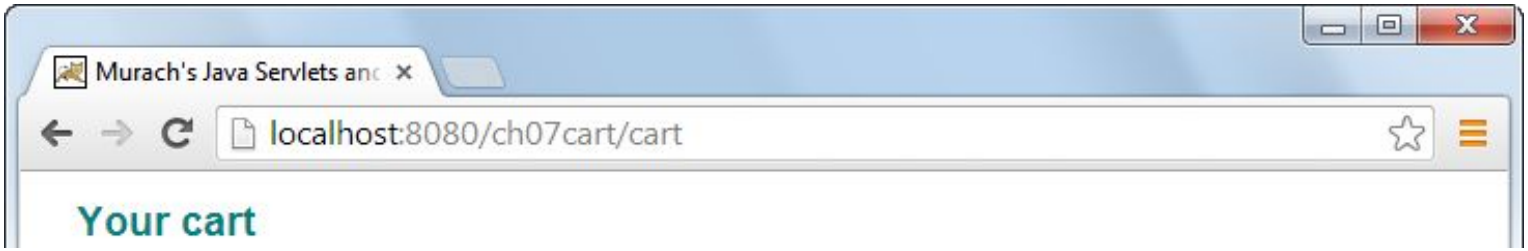
A form with a hidden text field and a button

```
<form action="cart" method="post">
    <input type="submit" value="Add To Cart">
    <input type="hidden" name="productCode" value="8601">
</form>
```

The form displayed in a browser

86 (the band) - True Life Songs and Pictures	$14.95	Add To Cart

The URL when you click the button



A form tag that uses EL to set hidden field values

```
<form action="cart" method="post">
    <input type="hidden" name="productCode"
           value="${product.code}">
    <input type=text size=2 name="quantity"
           value="${lineItem.quantity}">
    <input type="submit" name="updateButton"
           value="Update">
</form>
```

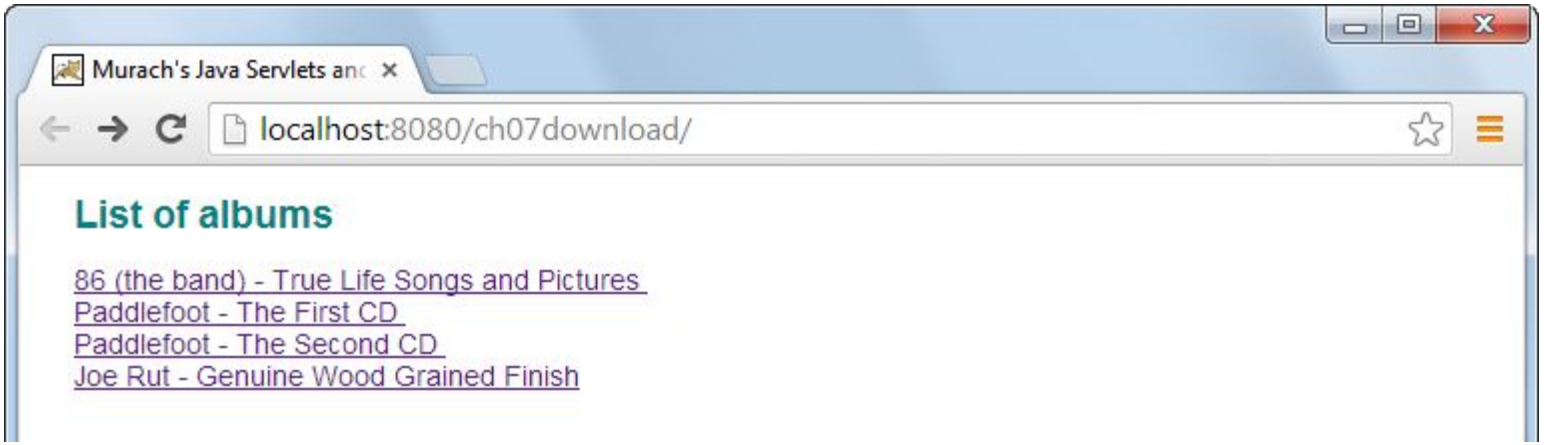
One limitation of hidden fields

- Because hidden fields are displayed in the source code for the page returned to the browser, anyone can view the parameters. As a result, hidden fields aren't appropriate for sensitive data like passwords.

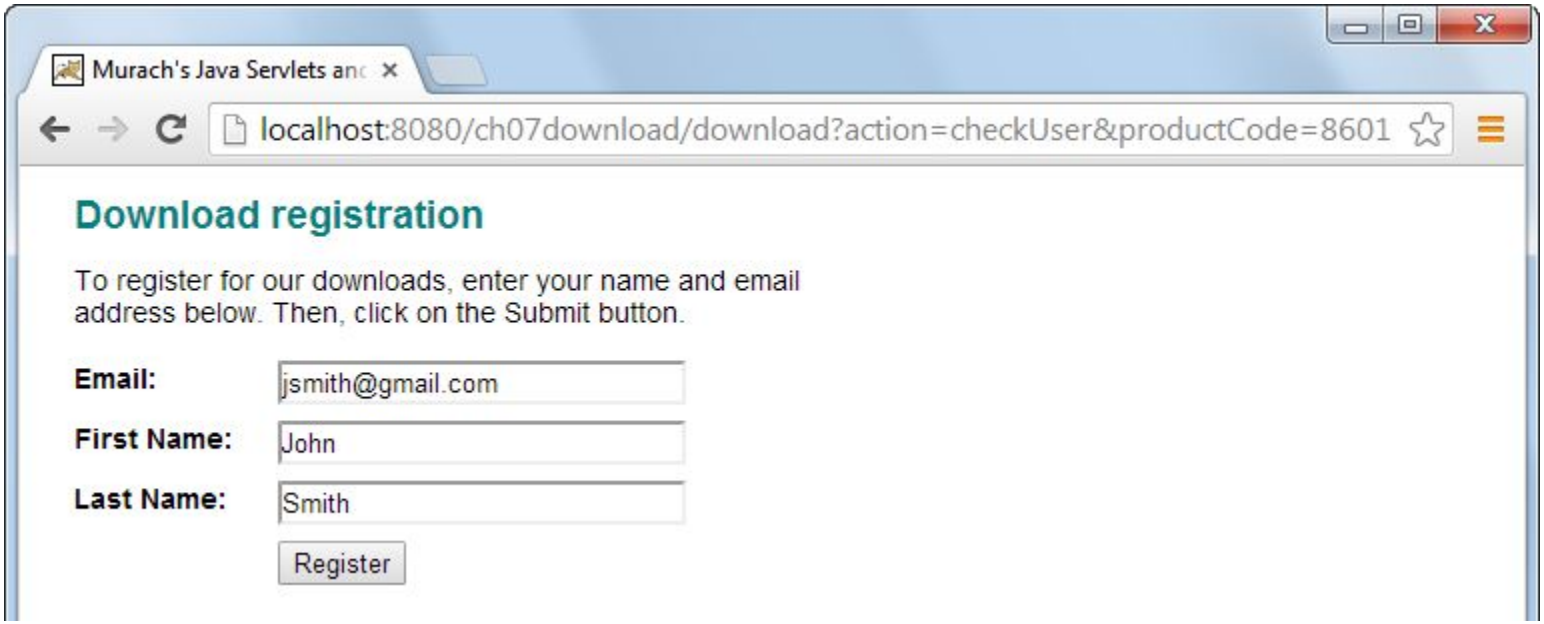
URL rewriting and hidden fields

- You can use *URL rewriting* to pass parameters to a servlet or JSP. To do that, you add the parameters to the end of the URL.
- You can use *hidden fields* to pass parameters to a servlet or JSP. To do that, you code hidden fields within a form tag.

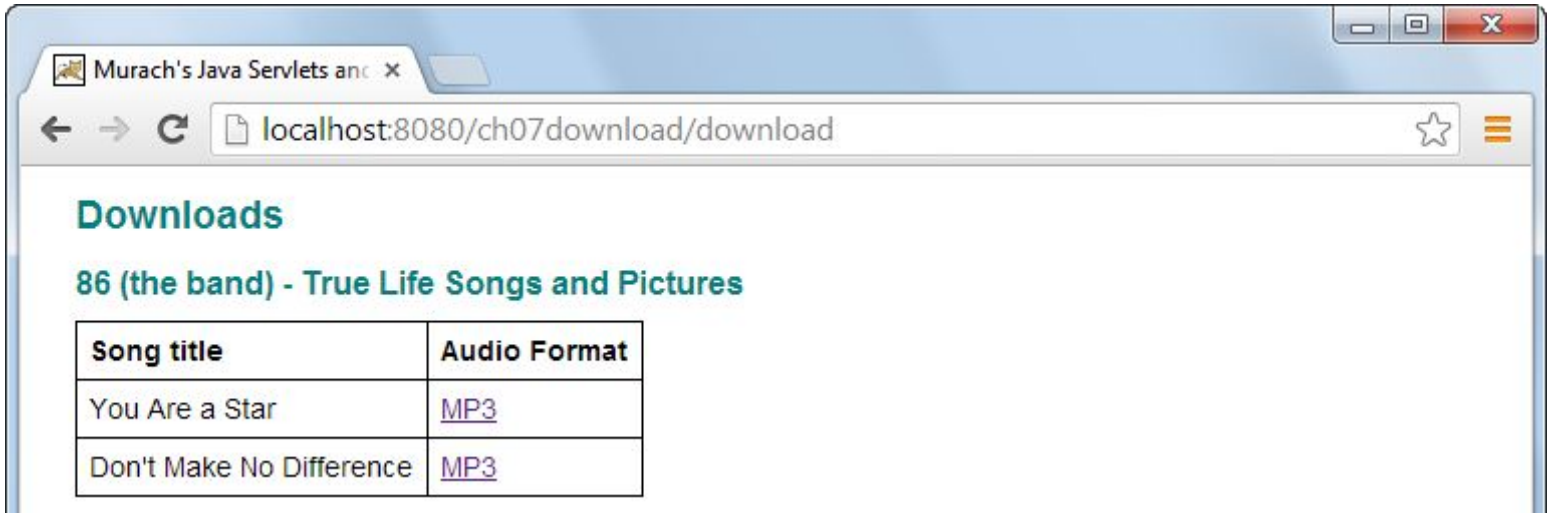
The Index page



The Register page



The Downloads page



The names of the jsp files

```
index.jsp
register.jsp
8601_download.jsp
pf01_download.jsp
pf02_download.jsp
jr01_download.jsp
```

The name of the controller servlet

```
murach.download.DownloadServlet
```

The file structure for the mp3 files

```
musicStore/sound/8601/*.mp3
musicStore/sound/pf01/*.mp3
musicStore/sound/pf02/*.mp3
musicStore/sound/jr01/*.mp3
```

The web.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <servlet>
        <servlet-name>DownloadServlet</servlet-name>
        <servlet-class>murach.download.DownloadServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>DownloadServlet</servlet-name>
        <url-pattern>/download</url-pattern>
    </servlet-mapping>

    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

The index.jsp file

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Murach's Java Servlets and JSP</title>
    <link rel="stylesheet" href="styles/main.css" type="text/css"/>
</head>
<body>

<h1>List of albums</h1>

<p>
<a href="download?action=checkUser&amp;productCode=8601">
    86 (the band) - True Life Songs and Pictures
</a><br>

<a href="download?action=checkUser&amp;productCode=pf01">
    Paddlefoot - The First CD
</a><br>
```

The index.jsp file (continued)

```
<a href="download?action=checkUser&amp;productCode=pf02">
    Paddlefoot - The Second CD
</a><br>

<a href="download?action=checkUser&amp;productCode=jr01">
    Joe Rut - Genuine Wood Grained Finish
</a>
</p>

</body>
</html>
```

The register.jsp file

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Murach's Java Servlets and JSP</title>
    <link rel="stylesheet" href="styles/main.css" type="text/css"/>
</head>
<body>

<h1>Download registration</h1>

<p>To register for our downloads, enter your name and email
   address below. Then, click on the Submit button.</p>
```

The register.jsp file (continued)

```
<form action="download" method="post">
    <input type="hidden" name="action" value="registerUser">
    <label class="pad_top">Email:</label>
    <input type="email" name="email" value="${user.email}"><br>
    <label class="pad_top">First Name:</label>
    <input type="text" name="firstName" value="${user.firstName}"><br>
    <label class="pad_top">Last Name:</label>
    <input type="text" name="lastName" value="${user.lastName}"><br>
    <label> </label>
    <input type="submit" value="Register" class="margin_left">
</form>

</body>

</html>
```

The 8601_download.jsp file

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Murach's Java Servlets and JSP</title>
    <link rel="stylesheet" href="styles/main.css" type="text/css"/>
</head>
<body>

<h1>Downloads</h1>

<h2>86 (the band) - True Life Songs and Pictures</h2>
```

The 8601_download.jsp file (continued)

```
<table>
<tr>
    <th>Song title</th>
    <th>Audio Format</th>
</tr>
<tr>
    <td>You Are a Star</td>
    <td>
        <a href="/musicStore/sound/${productCode}/star.mp3">MP3</a>
    </td>
</tr>
<tr>
    <td>Don't Make No Difference</td>
    <td>
        <a href="/musicStore/sound/${productCode}/no_difference.mp3">
        MP3</a>
    </td>
</tr>
</table>

</body>
</html>
```

The 8601_download.jsp file (continued)

- This is one of the four JSPs for downloading songs. The others are similar.
- When a browser receives the URL for a sound file, it downloads and plays it. That's one of the capabilities of a modern browser.
- This JSP gets the product code from the session object and uses it in the URLs for the sound files. But the URLs could also be hard-coded.
- Another way to handle the downloads is to write one JSP that works for all of the albums. To implement that, store the data for the downloadable songs in one file for each album. Then, the download JSP can get the product code from the session object, read the related file, and load its data into the table.

The DownloadServlet class

```
package murach.download;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

import murach.business.User;
import murach.data.UserIO;
import murach.util.CookieUtil;

public class DownloadServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws IOException, ServletException {

        // get current action
        String action = request.getParameter("action");
        if (action == null) {
            action = "viewAlbums";  // default action
        }
```

The DownloadServlet class (continued)

```
        // perform action and set URL to appropriate page
        String url = "/index.jsp";
        if (action.equals("viewAlbums")) {
            url = "/index.jsp";
        } else if (action.equals("checkUser")) {
            url = checkUser(request, response);
        }

        // forward to the view
        getServletContext()
                .getRequestDispatcher(url)
                .forward(request, response);
    }
```

The DownloadServlet class (continued)

```
@Override
public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {

    String action = request.getParameter("action");

    // perform action and set URL to appropriate page
    String url = "/index.jsp";
    if (action.equals("registerUser")) {
        url = registerUser(request, response);
    }

    // forward to the view
    getServletContext()
            .getRequestDispatcher(url)
            .forward(request, response);
}
```

The DownloadServlet class (continued)

```
private String checkUser(HttpServletRequest request,
        HttpServletResponse response) {

    String productCode = request.getParameter("productCode");
    HttpSession session = request.getSession();
    session.setAttribute("productCode", productCode);
    User user = (User) session.getAttribute("user");

    String url;
    // if User object doesn't exist, check email cookie
    if (user == null) {
        Cookie[] cookies = request.getCookies();
        String emailAddress =
            CookieUtil.getCookieValue(cookies, "emailCookie");

        // if cookie doesn't exist, go to Registration page
        if (emailAddress == null || emailAddress.equals("")) {
            url = "/register.jsp";
        }
```

The DownloadServlet class (continued)

```
            // if cookie exists,
            // create User object and go to Downloads page
            else {
                ServletContext sc = getServletContext();
                String path = sc.getRealPath("/WEB-INF/EmailList.txt");
                user = UserIO.getUser(emailAddress, path);
                session.setAttribute("user", user);
                url = "/" + productCode + "_download.jsp";
            }
        }
        // if User object exists, go to Downloads page
        else {
            url = "/" + productCode + "_download.jsp";
        }
        return url;
    }
```

The DownloadServlet class (continued)

```
private String registerUser(HttpServletRequest request,
        HttpServletResponse response) {

    // get the user data
    String email = request.getParameter("email");
    String firstName = request.getParameter("firstName");
    String lastName = request.getParameter("lastName");

    // store the data in a User object
    User user = new User();
    user.setEmail(email);
    user.setFirstName(firstName);
    user.setLastName(lastName);

    // write the User object to a file
    ServletContext sc = getServletContext();
    String path = sc.getRealPath("/WEB-INF/EmailList.txt");
    UserIO.add(user, path);
```

The DownloadServlet class (continued)

```
        // store the User object as a session attribute
        HttpSession session = request.getSession();
        session.setAttribute("user", user);

        // add a cookie that stores the user's email to browser
        Cookie c = new Cookie("emailCookie", email);
        c.setMaxAge(60 * 60 * 24 * 365 * 2); // set age to 2 years
        c.setPath("/");                      // allow entire app to access it
        response.addCookie(c);

        // create an return a URL for the appropriate Download page
        String productCode = (String)
            session.getAttribute("productCode");
        String url = "/" + productCode + "_download.jsp";
        return url;
    }
}
```