Chapter 7

How to work with sessions and cookies
Objectives

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.
Why session tracking is difficult with HTTP

Client

Browser

Browser

Browser

Server

Web server

Web server

Web server

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

The id is carried in a cookie, in the most commonly used mechanism.
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.
ch07cart: The Index page

<table>
<thead>
<tr>
<th>Description</th>
<th>Price</th>
<th>Add To Cart</th>
</tr>
</thead>
<tbody>
<tr>
<td>86 (the band) - True Life Songs and Pictures</td>
<td>$14.95</td>
<td></td>
</tr>
<tr>
<td>Paddlefoot - The first CD</td>
<td>$12.95</td>
<td></td>
</tr>
<tr>
<td>Paddlefoot - The second CD</td>
<td>$14.95</td>
<td></td>
</tr>
<tr>
<td>Joe Rut - Genuine Wood Grained Finish</td>
<td>$14.95</td>
<td></td>
</tr>
</tbody>
</table>
The Cart page

To change the quantity, enter the new quantity and click on the Update button.

Continue Shopping
Checkout
Page flow with parameters shown coming into servlet (●)

- productCode = 8601
- quantity = 0
- action = shop
- action = checkout

Checkout.jsp
How are those parameters generated by the JSPs?

The forms in the JSPs have hidden and non-hidden fields, for example the Update button on cart.jsp:

```html
<form action="" method="post">
  <input type="hidden" name="productCode" value="${item.product.code}">
  <input type="text" name="quantity" value="${item.quantity}" id="quantity">
  <input type="submit" value="Update">
</form>
```

So if the $item (a LineItem object) has productCode pf01 and quantity 2 this generates HTML like this:

```html
<form action="" method="post">
  <input type="hidden" name="productCode" value="pf01">
  <input type="text" name="quantity" value="2" id="quantity">
  <input type="submit" value="Update">
</form>
```

Which yields parameters productCode=pf01&quantity=2 in the POST body.
public class CartServlet extends HttpServlet {

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

        String url = "../index.jsp";
        ServletContext sc = getServletContext();

        // get current action
        String action = request.getParameter("action");
        if (action == null) {
            action = "cart";  // default action
        }
    }
CartServlet.java, continued

    // perform action and set URL to appropriate page
    if (action.equals("shop")) {
        url = "/index.jsp"; // the "index" page
    }
    else if (action.equals("cart")) {
        String productCode =
            request.getParameter("productCode");
        String quantityString =
            request.getParameter("quantity");

        HttpSession session = request.getSession();
        Cart cart = (Cart) session.getAttribute("cart");
        if (cart == null) {
            cart = new Cart();
        }
        ...
        session.setAttribute("cart", cart);
        url = "/cart.jsp";
### A method of the request object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSession()</td>
<td>Returns the HttpSession object.</td>
</tr>
</tbody>
</table>

### Three methods of the session object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setAttribute(String name, session Object o)</code></td>
<td>Stores any object in the session as an attribute.</td>
</tr>
<tr>
<td><code>getAttribute(String name)</code></td>
<td>Returns the value of the specified attribute as an Object type.</td>
</tr>
<tr>
<td><code>removeAttribute(String name)</code></td>
<td>Removes the specified attribute from this session.</td>
</tr>
</tbody>
</table>
How to get a session object
HttpPost 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.
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.
The syntax for link URLs with parameters

url?paramName1=paramValue1&paramName2=paramValue2&...  

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

localhost:8080/ch07cart/cart?productCode=8601
More examples of URLs with parameters

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 using URLs with parameters

- 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 URLs with parameters

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

A better way: use forms that POST parameter data…
A form with a hidden text field and a button

```html
<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

![URL example]
A form tag that uses EL to set hidden field values

```html
<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.
The Index page

List of albums

- 86 (the band) - True Life Songs and Pictures
- Paddlefoot - The First CD
- Paddlefoot - The Second CD
- Joe Rut - Genuine Wood Grained Finish
The Register page

![The Register page](image)

**Download registration**

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

- **Email:** jsmith@gmail.com
- **First Name:** John
- **Last Name:** Smith

**Register**
The Downloads page

Downloads

86 (the band) - True Life Songs and Pictures

<table>
<thead>
<tr>
<th>Song title</th>
<th>Audio Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>You Are a Star</td>
<td>MP3</td>
</tr>
<tr>
<td>Don't Make No Difference</td>
<td>MP3</td>
</tr>
</tbody>
</table>
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
<?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

```html
<!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>
</p>

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

The index.jsp file (continued)

```html
<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>
```
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)

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

</body>

</html>
```
The 8601_download.jsp file

```html
<!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)

```html
<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>
```

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

```java
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);
@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)

```java
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";
        }
    }

    return url;
}
```
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;
    
}