Session Handling

Naveen Kumar K S

adith.naveen@gmail.com

http://naveenks.com

Session tracking

- HTTP is a stateless protocol→ implies that connection is maintained only for one request response cycle.

- Some applications require us to maintain information particular to a user that spans more than one request-response cycle.

- For instance, in an on-line shop, a customer will want add items into the shopping cart while browsing through several pages of catalogue. Here we require a mechanism to maintain the cart information as the customer moves from one page to another.

- Maintaining user information between multiple request-response cycle is termed as **session tracking**. The information that is maintained is called state.

Maintaining State

- State needs to maintained across browser sessions:

  - Sometime we also need some information like login, password, some personal details to be maintained across several browser sessions so that next time we visit the same page, we don't have to retype the information.

  - In such case also user state must be maintained.

  - Also in the shopping cart example, if user accidently closes the window, then he must be able to recover the cart within certain period of time.

  - This type of state can be maintained by using what is called cookies.

Scenario: Without Session tracking

Buy Books

1. ☑ Head First Servlet and JSP
2. ☐ Head First EJB
3. ☐ Head First Java
4. ☐ LINUX in a nutshell

add to cart

1. User select "Head First EJB" book from the page displayed.
2. Clicks on "Add to Cart" (submit button)
3. `doPost()` of `AddToCartServlet` is called

```
protected void doPost(HttpServletRequest request,
   HttpServletResponse response) throws ServletException,
   IOException {
   PrintWriter out = response.getWriter();
   out.println("<html><body>");
   String books[]= request.getParameterValues("book");
   for(int i=0;i<books.length;i++)
    out.println((i+1)+". "+ books[i]+"<br>");
    out.println("<a href='book.html'>Add
      more</a></body></html>");  }
```

1. Head First Servlet and JSP
Add more

1. Head First Servlet and JSP
Add more

Buy Books

1. ☐ Head First Servlet and JSP
2. ☐ Head First EJB
3. ☐ Head First Java
4. ☐ LINUX in a nutshell

add to cart

1. LINUX in a nutshell
Add more

4. The "Head First EJB" which user added to the request is lost!
5. Only the last books that user added to the list remains.

- Since HTTP is a connectionless protocol, connection gets closed after one-request response cycle.
- There for each request a new `HttpServletRequest` and `HttpServeltResponse` object are created.
- Hence the old request object is lost, old values stored with the request is also lost.

Ways to achieve session tracking

- Without explicit support from servlet API – programming session handling

  - URL rewriting

  - Hidden form fields

  - SSL Sessions

- Using Servlet API support

  - `javax.servlet.http.Cookies`

  - `javax.servlet.http.HttpSession`

URL Rewriting

- One of the ways to maintain the request information across multiple requests of the same user is to keep sending back the information by appending the URL with the query string containing the old request parameters.

- But this will require that every page to be dynamically generated since query string has to be computed and appended with every URL. Hence it cannot be enforced for a static html page.

- Also here state can be a maintained only for the same browser session. If accidently user closes the browser window session is lost.

- Example in next slides demonstrates how URL rewriting can be done programmatically.

- We will find that a it requires a lot of coding effort in every dynamic page to achieve session maintenance this way.

Example scenario: URL Rewriting

Buy Books

1. ☑ Head First Servlet and JSP
2. ☑ Head First EJB
3. ☐ Head First Java
4. ☐ LINUX in a nutshell

[add to cart]

http://localhost:8090/WithoutSession/AddToCartServlet?

Books in the cart

Head First Servlet and JSP
Head First EJB
Add more

http://localhost:8090/WithoutSession/
AddToCartServlet?pbook=Head First
Servlet and JSP,Head First EJB

Books in the cart

Head First Servlet and JSP
Head First EJB
Head First Java
LINUX in a nutshell
Add more

Buy Books

1. ☐ Head First Servlet and JSP
2. ☐ Head First EJB
3. ☑ Head First Java
4. ☑ LINUX in a nutshell

[add to cart]

http://localhost:8090/WithoutSession/AddToCartServlet?pbook=Head%20Fir
st%20Servlet%20and%20JSP,Head%20First%20EJB

Displays the form and appends the query string with the form URL if not null

```java
@WebServlet("/AddToCartServlet")
public class AddToCartServlet extends HttpServlet {
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
    String book=request.getQueryString();
    String s= "<html><head> <title>Books</title> </head> "+
            "<body><h1>Buy Books</h1>  "+
            "<form method='post'
    action='AddToCartServlet?"+((book==null)?"":book) +
    "'><ol><li><input type='checkbox' name='book'
    value='Head First Servlet and JSP'>Head First Servlet
    and JSP  "+
    "<li><input type='checkbox' name='book' value='Head
    First EJB'>Head First EJB  "+
    "<li><input type='checkbox' name='book' value='Head
    First Java'>Head First Java  "+
    "<li><input type='checkbox' name='book' value='LINUX in
    a nutshell'>LINUX in a nutshell</ol>  "+
    "<input type='submit' value='add to
    cart'></form></body></html> ";
PrintWriter out = response.getWriter();
out.println(s);}
```

Displays the all the books that have been selected across multiple pages

```java
protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
PrintWriter out = response.getWriter();
out.println("<html><body><h2>Books in the cart</h2>");
String books[]= request.getParameterValues("book");
String book= request.getParameter("pbook");
int j=0;
if(book!=null){
String pbooks[]=book.split(",");
for(j=0;j<pbooks.length;j++){
out.println(pbooks[j]+"<br>");
}}
if(books!=null){
for(int i=0;i<books.length;i++){
out.println( books[i]+"<br>");
if(book==null) {book="" ; }
else book=book+",";
book=book+books[i];
}}
out.println("<a href='AddToCartServlet?pbook="+book+"'>Add
more</a></body></html>");}}
```

Get the query string
parameters to display

Construction of query string
with new form parameters

Hidden Form Fields

- This approach is very similar to that of URL rewriting except that instead of appending a query string, the values of the request is sent in the hidden field of a form.

  - `<INPUT TYPE="hidden" NAME="uid" VALUE="123">`

- This approach also has the same disadvantage as URL rewriting that every page has to be dynamically generated so that the hidden field value can be computed in code. Also the session can be maintained only for one browser window.

- Also this approach requires that every page has a form embedded with a hidden field.

What is a Cookie?

- HTTP Cookie/ Web Cookie/ Browser Cookie is

  - a small piece of textual information

  - sent by the server to the client

  - stored on the client's machine,

  - and returned by the client's machine with each request made to the server.

- Web server sends cookies by sending the Set-Cookie header with HTTP Response.

- Browser sends back the cookie to only to the domain and path that is set with it.

- Cookies maintains information between more than one browser session.

HTTP cookie attributes

- Every cookie has
  - A name-value pair
    - This value that is saved for session maintenance
    - This is the only mandatory value that a cookie must have.
  - Domain, Path
    - server domain and path where browser should sends the cookie
  - Max-Age, Expires
    - Max-Age is tells the browser how long should the cookie be alive given in terms of seconds.
    - how long should the cookie be alive specified in the form of the form of "Wdy, DD Mon YYYY HH:MM:SS GMT
  - secure and HttpOnly
    - These do not have any value associated with them.
    - secure implies that cookie be sent through encrypted transmission like SSL
    - HttpOnly attribute tells browsers to use cookies via the HTTP protocol only
  - Comment

Cookie example and getting cookie

- Response header cookie format

  ```
  Set-Cookie:NAME=VALUE;Comment=COMMENT;
  Domain=DOMAINNAME;Max-Age=SECONDS;  Path=PATH;secure
  ```

- Response header cookie example

  ```
  example:Set-Cookie:userid=anita;Max-
  Age=60;Domain=".myser.com";Path="/"
  ```

- To get all cookies the client sent with this request, method of
  HttpServletRequest
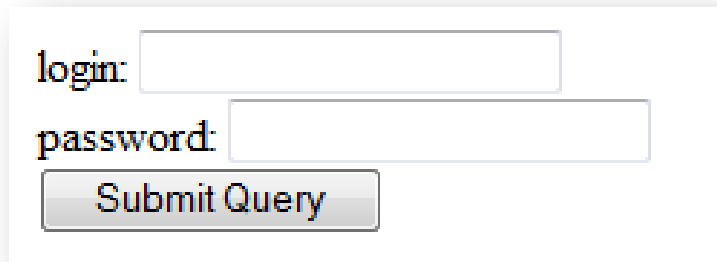
  - **request.getCookies();**

javax.servlet.http.Cookie

- **`Cookie(String name, String value)`**
- **`String getName()`**
- **`void setValue(String newValue),String getValue()`**
- **`void setDomain(String pattern),String getDomain()`**
- **`void setPath(String uri),String getPath()`**
  - By default, the domain and path will be from where cookie originated.
  - If a path like **`/MyWebApp`** is specified then the cookie will be avaialble to all the directories under **`/MyWebApp`**
- **`void setSecure(boolean flag), boolean getSecure()`**
- **`void setHttpOnly(boolean isHttpOnly), boolean isHttpOnly()`**
- **`void setMaxAge(int expiry), int getMaxAge()`**
  - +ve value → cookie will expire after that many seconds have passed and –ve value indicates that it will be deleted when the Web browser exits.
  - By default, -1 indicating the cookie will persist until browser shutdown.
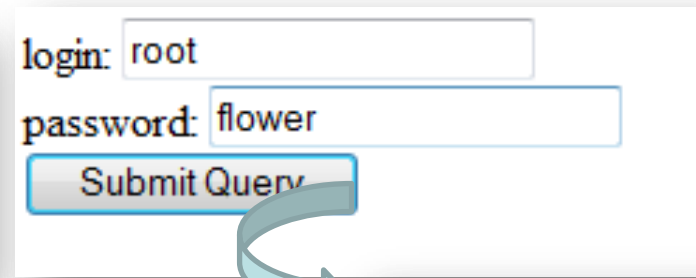
Example: Cookies

- A servlet displays login and password with the values of login and password set if the cookies are set. If cookies are not set, then the login and password entered by the user are set as cookies so that when user requests for this page next time the login and password are automatically set.
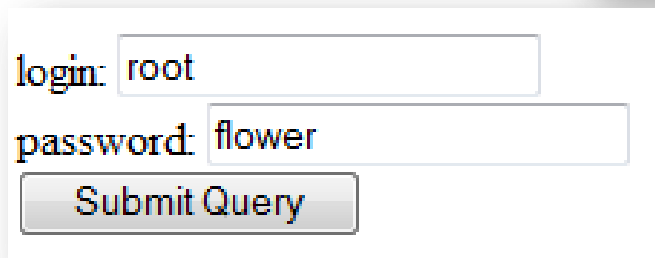
First request:

Cookie login and password are set

Subsequent requests
If the requests are made in within a day, then the login and password are pre-populated

```java
WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
     out.println("<html><head><title>Login </title></head>");
        String login="";String password="";
        Cookie[] cookies=request.getCookies();
        if(cookies!=null){
        for(int i=0; i<cookies.length; i++){
         String test=cookies[i].getName();
        if(test.equals("login")) login=cookies[i].getValue();
        if(test.equals("password"))password=cookies[i].getValue();
}}
out.println("<body><form action='LoginServlet' method='post'>");
out.println("login: <input type='text' name='login' value='"+
login+"'><br>");
out.println("password: <input type='text' name='password'
value='"+ password+"'><br>");
out.println("<input type='submit'><br>");
out.println("</form></body></html>");}
```
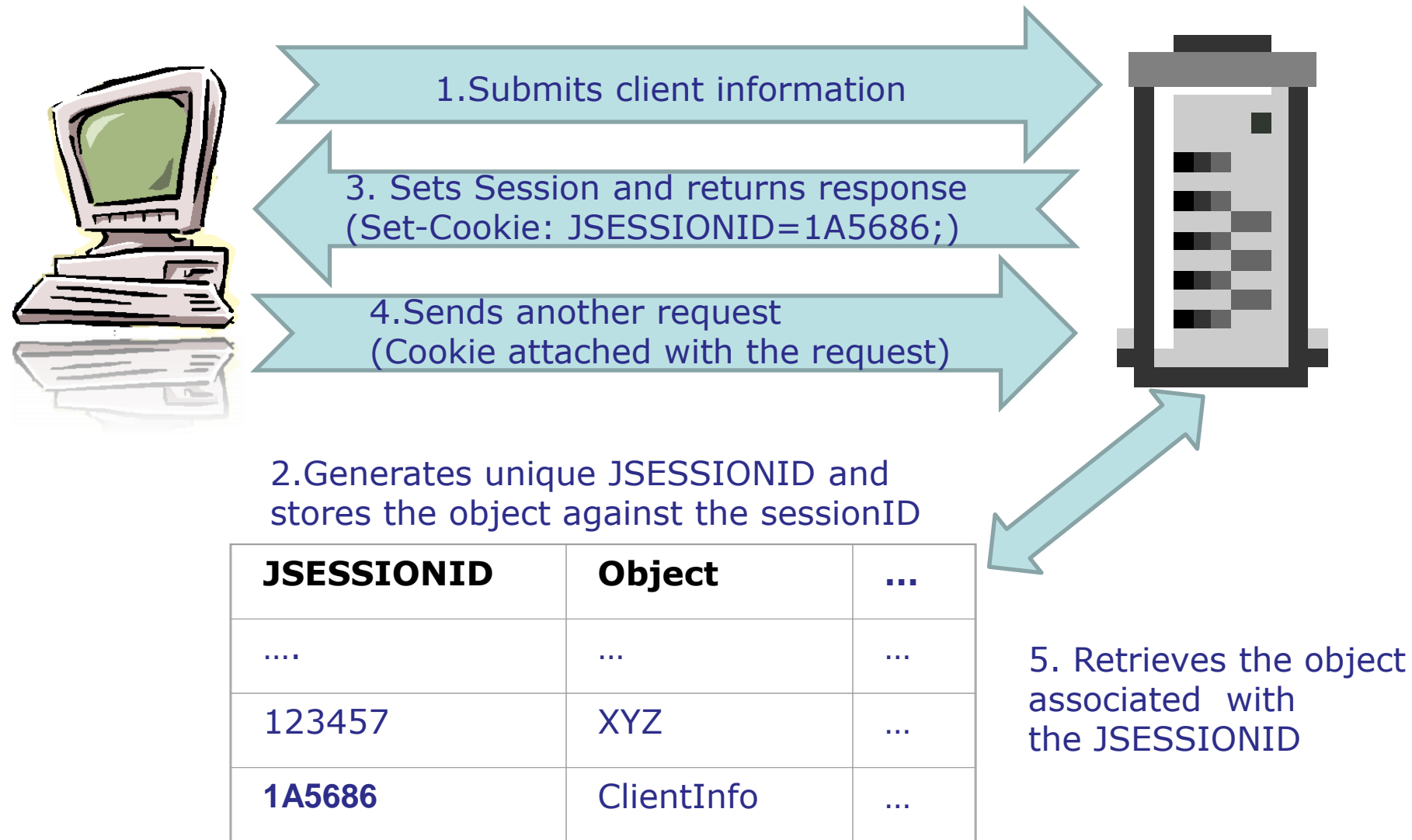
Getting cookies

Pros and cons of cookies

- Cookies are not mixed up with the HTML content, request or response bodies. The server can transparently set the cookie in the response header and extract from the request header.

- Browser use can disable cookies

HttpSession

- A very way to handle session is by using **HttpSession**.

- Java servlet API has an interface called **HttpSession** which helps us in session tracking.

- The web container uses either cookies or URL rewriting for establishing session.

- Like objects is associated with request object, an objects can be objects can be stored in session as well by associating it with a name in the session thus forming a name-value pair.

- If a servlet uses the **RequestDispatcher** to call a servlet in another Web application, any sessions created for and visible to the servlet being called is different from those visible to the calling servlet.

Cookies in session

1. Submits client information

3. Sets Session and returns response
(Set-Cookie: JSESSIONID=1A5686;)

4. Sends another request
(Cookie attached with the request)

2. Generates unique JSESSIONID and
stores the object against the sessionID

JSESSIONID	Object	...
....	...	...
123457	XYZ	...
1A5686	ClientInfo	...

5. Retrieves the object associated with the JSESSIONID

Methods to create HttpSession

- **`HttpServletRequest`** has following methods to create **`HttpSession`**:

  - **`HttpSession getSession()`**

    - If a session already exists then that is returned otherwise a new session is returned.

    - **`HttpSession getSession(boolean create)`**

      - If create is true it functions in the same way as the function **`getSession()`**

      - If create is false then if there is no session associated with the request, this method returns null else returns the session associated with this request.

- **`HttpSession`** interface method

  - **`boolean isNew`**() can be used to check if the session is a new

  - **`String getId()`** returns unique identifier assigned to this session.

Setting, getting and removing session attributes

- Values can be stored in the session object using

  `void setAttribute(String name, Object attribute)`

- Example: `session.setAttribute("uname", "scott")`

- Attributes bound into a session are available to any other servlet that belongs to the same `ServletContext` and handles a request identified as being part of the same session.

- Values can be retrieved from the session object using

   `Object getAttribute(String name)`

- Example: `String o=(String)session.getAttribute("uname");`

- To remove attributes from the session object

  - `void removeAttribute(String name)`

Destroying session

- Session gets destroyed in one of the following ways:
  - On calling invalidate()
    - `void invalidate()`
    - This method invalidates the current session then unbinds any objects bound to it.
    - `IllegalStateException` occurs if this method is called on an already invalidated session
  - When client does not respond with-in the time-out period
  - When application crashes
  - When application is no longer available
- After session has become invalid, accessing session attributes causes `IllegalStateException.`

Timeout

- There are three ways to specify timeout:
  - For every web project, timeout can be set through
    1. DD (web.xml)

       `<session-config>`

         `<session-timeout>30<session-timeout>`

       `</session-config>`

       Timeout interval given in minutes. A session whose timeout period has been set to -1 will never expire.
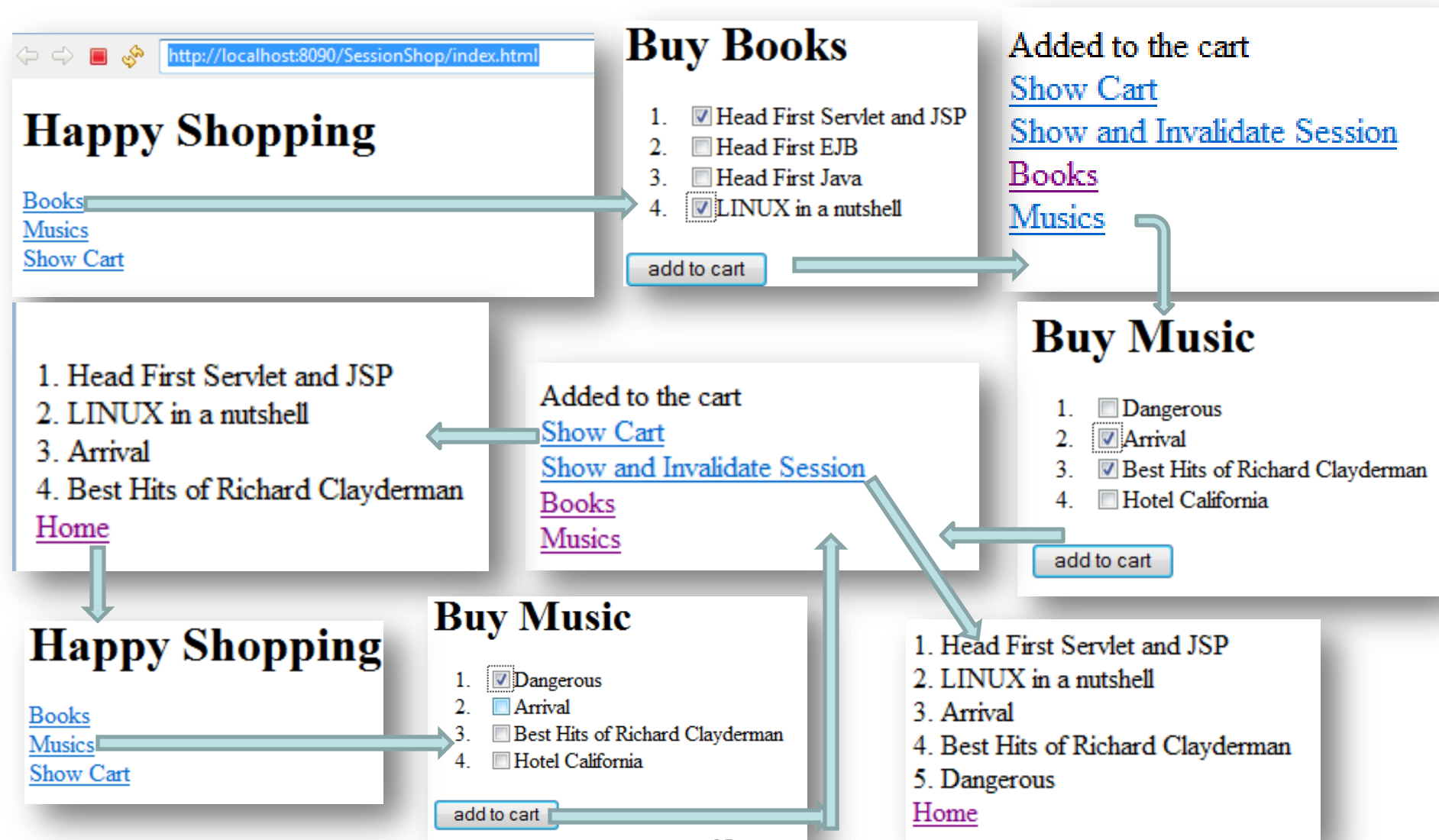   2. **`Object setMaxInactiveInterval(int interval)`**
       - The interval is given in terms of seconds. It is the time between client requests before the servlet container will invalidate this session.
       - A negative time indicates the session should never timeout.
       - This overrides any default in the servlet container or set through the web.xml.
    3. Servlet container provides a way to set the timeout in general for all web application. In cases where the application does not specify the timeout, this is used . For Tomcat, it's 30 minutes by default.

Example: Shopping Cart

- This example demonstrates implementation of a shopping cart.
- User selects items from the Books page and the CD page and adds to cart.
- User can view the cart at any point of time or invalidate the session.

```
<html><head>                              index.html
<title>Books</title>
</head><body>
<h1>Happy Shopping </h1>
<a href="book.html">Books</a><br>
<a href="music.html">Musics</a><br>
<a href=ShowServlet?flag=n>Show Cart</a>
</body></html>
```

```
<html><head><title>Books</title></head>         book.html
<body><h1>Buy Books</h1>
<form method="post" action="AddToCartServlet">
<ol><li><input type="checkbox" name="book" value="Head First
Servlet and JSP">Head First Servlet and JSP
<li><input type="checkbox" name="book" value="Head First
EJB">Head First EJB
<li><input type="checkbox" name="book" value="Head First
Java">Head First Java
<li><input type="checkbox" name="book" value="LINUX in a
nutshell">LINUX in a nutshell</ol>
<input type="submit" value="add to cart"></form>
</body></html>
```

26

```
<html><head><title>Books</title></head>          music.html
<body><h1>Buy Music</h1>
<form method="post" action="AddToCartServlet">
<ol><li><input type="checkbox" name="music"
value="Dangerous">Dangerous
<li><input type="checkbox" name="music"
value="Arrival">Arrival
<li><input type="checkbox" name="music" value="Best Hits of
Richard Clayderman">Best Hits of Richard Clayderman
<li><input type="checkbox" name="music" value="Hotel
California">Hotel California</ol>
<input type=submit value="add to cart">
</form></body></html>
```

```java
@WebServlet("/AddToCartServlet")
public class AddToCartServlet extends HttpServlet {
private static final long serialVersionUID = 1L;
 public void doPost(HttpServletRequest
request,HttpServletResponse response) throws ServletException,
IOException {
PrintWriter out = response.getWriter();
try{
String[] music= request.getParameterValues("music");
String[] book= request.getParameterValues("book");

HttpSession session= request.getSession();

ArrayList<Object> cart=
(ArrayList<Object>)session.getAttribute("cart");
if(cart==null)
cart= new ArrayList<Object>();

if(music!=null)
for(int i=0;i<music.length;i++)cart.add(music[i]);
if(book!=null)
for(int i=0;i<book.length;i++) cart.add(book[i]);
```

Get the if it exists else get new session

Create a new cart object if it does not exist

Add music and CDs to the cart object

```
session.setAttribute("cart",cart);
out.println("Added to the cart<br>");

out.println("<a href='ShowServlet?flag=n'>
Show Cart</a><br>");
out.println("<a href='ShowServlet?flag=y'>
Show and Invalidate Session</a><br>");


out.println("<a href=book.html>Books</a><br>");

out.println("<a href=music.html>Musics</a><br></body></html>");

}catch(Exception e){
out.println("<html><body><font color=red>Some invalid operation
caused an exception to be raised</font>");
out.println("<p> Exception generated
:"+e+"</p></body></html>");}
finally{out.close();}
}
}
```

Add cart to the session

Flag is sent as query string which will determine whether to invalidate session or not

```java
@WebServlet("/ShowServlet")
public class ShowServlet extends HttpServlet {
private static final long serialVersionUID = 1L;
 public void doGet(HttpServletRequest
request,HttpServletResponse response) throws ServletException,
IOException {
PrintWriter out = response.getWriter();
try{

        out.println("<html><body><br>");
        HttpSession session= request.getSession(false);

        ArrayList<Object>
        cart=(ArrayList<Object>)session.getAttribute("cart");
        for(int i=1;i<=cart.size();i++){
        out.println(i+". "+ cart.get(i-1)+"<br>");        }

String flag=request.getParameter("flag");
if(flag.equals("y"))
session.invalidate();
```

Display cart

invalidate session

30

```java
out.println("<a href=index.html>Home</a>");

out.println("</body></html>");
}catch(Exception e)
{
out.println("<html><body><font color=red>Some invalid
operation caused an exception to be raised</font>");
out.println("<p> Exception generated
:"+e.toString()+"</p></body></html>");}
finally{
out.close();
}
}}
```

Cookies disabled: `encodeURL()`

- Some browsers may have cookies disabled. If cookie is disabled, the session code that we have written so far will not work! To make it work we need to do an extra bit.

- On doing this extra bit, application server employs the URL rewriting mechanism when cookies do not work.

- `String encodeURL(String url)` of `HttpResponse` class must be used on all the URLs so that the URL rewriting mechanism works!

- This in turn also means that all the pages must be dynamically generated. And all the links and forms must call encodeURL() on their URLs.

- `encodeURL()` adds appends the session id to the URLs

```
out.println("<a href="+
response.encodeURL("show")+">Show Cart</a><br>");
```

`show.do;jsessionid=00WV14552`

encodeRedirectURL()

- **`String encodeRedirectURL(java.lang.String url)`**

- This method is used to encode the specified URL for use in the **`sendRedirect`** method. Like **`encodeURL()`** method if encoding is not needed URL is returned unchanged.

- The implementation of this method includes the logic to determine whether the session ID needs to be encoded in the URL depending on the URL. If the URL in **`sendRedirect`** is for a different web application when appending session ID is not needed.

- Because of this, the method is separated from the **`encodeURL`** method.

- It is recommended that all URLs sent to the **`HttpServletResponse.sendRedirect`** method should be run through this method so that URL encoding can happen if browsers do not support cookies.

- **`encodeURL()`** and **`encodeRedirectURL()`** methods result in no action if cookies are enabled.

Summary of types of attributes in the application that we have seen so far

- Attributes stored with `ServletContext`

  - Available to the entire web application

- Attributes stored with `ServletConfig`

  - Available only to the particular servlet

- Attributes stored with `HttpSession`

  - Available with respect to the user

- Attributes stored with `HttpServletRequest`

  - Available with respect to particular request