Implementing **session tracking** from scratch
The session-tracking API
Storing immutable objects vs. storing mutable objects
Examples of usage of the session-tracking API
Understanding the benefits of **beans**
Creating beans
Installing bean classes on your server
Accessing bean properties
Explicitly setting bean properties
Automatically setting bean properties from request parameters
Sharing beans among multiple servlets and JSP pages

**Most of the slides were made available by www.coreservlets.com**
Rolling Your Own Session Tracking: Cookies

- Idea: associate cookie with data on server
  
  ```java
  String sessionID = makeUniqueString();
  HashMap sessionInfo = new HashMap();
  HashMap globalTable = findTableStoringSessions();
  globalTable.put(sessionID, sessionInfo);
  Cookie sessionCookie =
      new Cookie("JSESSIONID", sessionID);
  sessionCookie.setPath("/");
  response.addCookie(sessionCookie);
  ```

- Still to be done:
  - Extracting cookie that stores session identifier
  - Setting appropriate expiration time for cookie
  - Associating the hash tables with each request
  - Generating the unique session identifiers
Your Own Session Tracking: **URL-Rewriting**

- **Idea**
  - Client appends some extra data on the end of each URL that identifies the session
  - Server associates that identifier with data it has stored about that session
  - E.g., http://host/path/file.html; jsessionid=1234

- **Advantage**
  - Works even if cookies are disabled or unsupported

- **Disadvantages**
  - Must encode all URLs that refer to your own site
  - All pages must be dynamically generated
  - Fails for bookmarks and links from other sites.
Your Own Session Tracking: **Hidden Fields**

- **Idea:**
  
  `<INPUT TYPE="HIDDEN" NAME="session" VALUE="...">`

- **Advantage**
  
  - Works even if cookies are disabled or unsupported

- **Disadvantages**
  
  - Lots of tedious processing
  
  - All pages must be the result of form submissions.
Session Tracking in Java

- **Session objects live on the server**
- Sessions automatically **associated** with client via cookies or URL-rewriting
- Use `request.getSession()` to get session
  - Behind the scenes, the system looks at cookie or URL extra info and **sees if it matches the key** to some previously stored session object
  - If so, it returns that stored session object
  - If not, **it creates a new one**, assigns a cookie or URL info as its key, and returns that new session object
- **Hashtable-like mechanism lets you store arbitrary objects inside the HttpSession object**
  - `setAttribute(name, value)` stores values
  - `getAttribute(name)` retrieves values
Session Tracking Basics

- Access the session object
  - Call `request.getSession` to get `HttpSession` object
  - This is a hashtable associated with the user

- Store information in a session
  - Use `setAttribute` with a key (String) and a value (object)

- Look up information associated with a session
  - Call `getAttribute` on the `HttpSession` object, cast the return value to the appropriate type, and check whether the result is null

- Discard session data
  - Call `removeAttribute` discards a specific value
  - Call `invalidate` to discard an entire session.
HttpSession session = request.getSession();
SomeClass value =
    (SomeClass)session.getAttribute("credentials");
if (value == null) {
    value = new SomeClass(...);
    session.setAttribute("credentials", value);
}
doSomethingWith(value);

- Do not need to call setAttribute again (after modifying value) if the modified value is the same object
- But, if value is immutable (e.g. a String object), modified value will be a new object reference, and you must call setAttribute again.
What Changes if Server Uses URL Rewriting?

- **Session tracking code: No change**
- Code that generates hypertext links back to same site:
  - Pass URL through `response.encodeURL()`
    - If server is using cookies, this returns URL unchanged
    - If server is using URL rewriting, this appends the session info to the URL
    - E.g.:
      ```java
      String url = "order-page.html";
      url = response.encodeURL(url);
      ```
- Code that does `sendRedirect` to own site:
  - Pass URL through `response.encodeRedirectURL()`
HttpSession Methods

- **getAttribute**
  - Extracts a previously stored value from a session object - returns null if no value is associated with given name

- **setAttribute**
  - Associates a value with a name
  - If you want to monitor changes: value must implement HttpSessionBindingListener interface (method valueBound)

- **removeAttribute**
  - Removes values associated with name

- **getAttributeNames**
  - Returns names of all attributes in the session

- **getId**
  - Returns the unique identifier.
 HttpSession Methods (Continued)

- **isNew**
  - Determines if session is new to *client* (e.g. the client has not sent back the cookie)

- **getCreationTime**
  - Returns time at which session was first created

- **getLastAccessedTime**
  - Returns time at which session was last sent from client

- **getMaxInactiveInterval**, **setMaxInactiveInterval**
  - Gets or sets the amount of time session should go without access before being invalidated

- **invalidate**
  - Invalidates current session.
public class ShowSession extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        HttpSession session = request.getSession();
        String heading;
        Integer accessCount =
            (Integer)session.getAttribute("accessCount");
        if (accessCount == null) {
            accessCount = new Integer(0);
            heading = "Welcome, Newcomer";
        } else {
            heading = "Welcome Back";
            accessCount =
                new Integer(accessCount.intValue() + 1);
        }
        session.setAttribute("accessCount", accessCount);
    }
}
A Servlet that Shows Per-Client Access Counts

PrintWriter out = response.getWriter();
...
out.println((docType +
    "<HTML>
    "<HEAD><TITLE>" + title + "</TITLE></HEAD>
    "<BODY BGCOLOR="#FDF5E6">
    "<CENTER>" +
    "<H1>" + heading + "</H1>
    "<H2>Information on Your Session:</H2>
    "<TABLE BORDER=1>
    "<TR BGCOLOR="#FFAD00">
    "<TD>Info Type</TD><TD>Value</TD>
    "<TD>Number of Previous Accesses</TD><TD>" + accessCount + "</TD>
    "</TABLE>
    "</CENTER>
    "</BODY>
    "</HTML>"));
Shows Per-Client Access Counts: Result 1

Welcome, Newcomer

Information on Your Session:

<table>
<thead>
<tr>
<th>Info Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>009B20AD9661F17AC433193A62DC13CC</td>
</tr>
<tr>
<td>Creation Time</td>
<td>Mon Oct 16 10:02:36 GMT-05:00 2006</td>
</tr>
<tr>
<td>Time of Last Access</td>
<td>Mon Oct 16 10:02:36 GMT-05:00 2006</td>
</tr>
<tr>
<td>Number of Previous Accesses</td>
<td>0</td>
</tr>
</tbody>
</table>

new Date (session.getCreat ionTime())

new Date (session.getLastAcc essedTime())

session.getId()
Shows Per-Client Access Counts: Result 12

Welcome Back

Information on Your Session:

<table>
<thead>
<tr>
<th>Info Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>009B20AD9661F17AC433193A62DC13CC</td>
</tr>
<tr>
<td>Creation Time</td>
<td>Mon Oct 16 10:02:36 GMT-05:00 2006</td>
</tr>
<tr>
<td>Time of Last Access</td>
<td>Mon Oct 16 10:04:36 GMT-05:00 2006</td>
</tr>
<tr>
<td>Number of Previous Accesses</td>
<td>11</td>
</tr>
</tbody>
</table>
Accumulating a List of User Data: Front End

Order Form

New Item to Order: Yacht

Order and Show All Purchases
Accumulating a List of User Data: Result

Items Purchased

- Yacht
- Chalet
- Lamborghini
- *Core Servlets and JavaServer Pages*
public class ShowItems extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        HttpSession session = request.getSession();
        ArrayList previousItems = (ArrayList) session.getAttribute("previousItems");
        if (previousItems == null) {
            previousItems = new ArrayList();
            session.setAttribute("previousItems", previousItems);
        }
        String newItem = request.getParameter("newItem");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Items Purchased";
        String docType = "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 " + " Transitional//EN">\n";
Accumulating a List of User Data

```java
out.println(docType +
  "<HTML>\n" +
  "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
  "<BODY BGCOLOR="#FDF5E6">\n" +
  "<H1>" + title + "</H1>";

synchronized(previousItems) {
  if (newItem != null) {
    previousItems.add(newItem);
  }
  if (previousItems.size() == 0) {
    out.println("<I>No items</I>");
  } else {
    out.println("<UL>");
    for(int i=0; i<previousItems.size(); i++) {
      out.println("<LI>" + (String)previousItems.get(i));
    }
    out.println("</UL>");
  }
}
out.println("</BODY></HTML>");
```
An On-Line Bookstore

All-Time Best Children's Fantasy Books

*The Chronicles of Narnia* by C.S. Lewis ($19.95)

The classic children's adventure pitting Aslan the Great Lion and his followers against the White Witch and the forces of evil. Dragons, magicians, quests, and talking animals wound around a deep spiritual theme. Series includes *The Magician's Nephew, The Lion, the Witch and the Wardrobe, The Horse and His Boy, Prince Caspian, The Voyage of the Dawn Treader, The Silver Chair, The Horse and His Boy, and *The Last Battle.*

*The Prydain Series* by Lloyd Alexander ($19.95)

Humble pig-keeper Taran joins mighty Lord Gwydion in his battle against the evil Eilonwy, wannabe bard Flinelwudur Fflam, and friends along the way. Series includes *The Book of Three, The Black Cauldron, The High King, the Palace of Silver,* and *The High King.*

*The Harry Potter Series* by J.K. Rowling ($19.95)

The first five of the popular stories about wizard-in-training Harry Potter and his friends, including *Harry Potter and the Sorcerer's Stone, Harry Potter and the Chamber of Secrets, Harry Potter and the Prisoner of Azkaban,* and *Harry Potter and the Goblet of Fire,* and *Harry Potter and the Order of the Phoenix.*

All-Time Best Computer Books

*Core Servlets and JavaServer Pages 2nd Edition (Volume 1)* by Marty Hall and Larry Brown ($39.95)

The definitive reference on servlets and JSP from Prentice Hall and Sun Microsystems Press. Nominated for the Nobel Prize in Literature.

*Core Web Programming, 2nd Edition* by Marty Hall and Larry Brown ($49.99)

One stop shopping for the Web programmer. Topics include

- Thorough coverage of Java 2; including threads, networking, Swing, Java 2D, RMI, JDBC, and Collections
- A fast introduction to HTML 4.01, including frames, style sheets, and layers.
- A fast introduction to HTTP 1.1, servlets, and JavaServer Pages.
- A quick overview of JavaScript 1.2

Add to Shopping Cart
An On-Line Bookstore

This servlet displays three forms

- Two ("update order" submit) call again the same servlet and update the number of items in the cart
- The third call the checkout page (not displayed here)
An On-Line Bookstore

- Session tracking code stays the same as in simple examples
- Shopping cart is an attribute of the session object
- Shopping cart class is relatively complex
  - Identifies items by a unique catalog ID
  - Does not repeat items in the cart
    - Instead, each entry has a count associated with it
      - If count reaches zero, item is deleted from cart
- Pages built automatically from objects that have descriptions of books.
Distributed and Persistent Sessions

- Some servers support **distributed Web applications**
  - **Load balancing** used to send different requests to different machines - sessions should still work even if different hosts are hit

- Some servers support persistent sessions
  - Session data written to disk and reloaded when server is restarted (as long as browser stays open)
    - Tomcat 5 and 6 support this

- To support both, session data should implement the java.io.Serializable interface
  - There are no methods in this interface; it is just a flag:
    ```java
    public class MySessionData implements Serializable
    {
        ...
    }
    ```
  - Built-in classes like String and ArrayList are already Serializable
Uses of JSP Constructs

- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Servlet/JSP combo (MVC)
- MVC with JSP expression language
- Custom tags
- MVC with beans, custom tags, and a framework like Struts or JSF
What Are Beans?

- **Java classes** that follow certain **conventions:**
  - Must have a **zero-argument** (empty) **constructor**
    - You can satisfy this requirement either by explicitly defining such a constructor or by omitting all constructors
    - *Not required if no JSP creates instances*
  - Should have **no public instance variables** (fields)
    - You should follow this practice and use **accessor** methods instead of allowing direct access to fields
  - Persistent values should be accessed through methods called **getXxx** and **setXxx**
    - If class has method `getTitle` that returns a **String**, class is said to have a **String property** named `title`
    - **Boolean** properties use `isXxx` instead of `getXxx`. 
Using Beans: Basic Tasks

- **jsp:useBean** - in the simplest case, this element builds a new bean

  ```xml
  <jsp:useBean id="beanName" class="package.Class" />
  ```

- **jsp:setProperty** - this element modifies a bean property (i.e., calls a method of the form setXxx)

  ```xml
  <jsp:setProperty name="beanName" property="propertyName" value="propertyValue" />
  ```

- **jsp:getProperty** - this element reads and outputs the value of a bean property

  ```xml
  <jsp:getProperty name="beanName" property="propertyName" />
  ```
Using Beans in Standalone Pages

- User submits form that refers to a JSP page
  
  `<FORM ACTION="SomePage.jsp"/>

- JSP page **instantiates** a bean
  
  `<jsp:useBean id="myBean" class="..."/>

- The JSP page **passes** some request **data** to the bean
  
  `<jsp:setProperty name="myBean" property="customerID" value="..."/>

- The JSP page outputs some value(s) derived from the request data and contained in the bean
  
  `<jsp:getProperty name="myBean" property="bankAccountBalance"/>`
Example: StringBean

```java
package coreservlets;

public class StringBean {
    private String message = "No message specified";

    public String getMessage() {
        return(message);
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

- Beans installed in normal Java directory
  - Let the IDE to decide where to create
  - Deployment: WEB-INF/classes/
    folderMatchingPackage

- Beans must **always** be in packages!
<jsp:useBean id="stringBean"
   class="coreservlets.StringBean" />

<OL>
  <LI>Initial value (from jsp:getProperty):
      <I><jsp:getProperty name="stringBean"
        property="message" /></I>
  <LI>Initial value (from JSP expression):
      <I><%= stringBean.getMessage() %></I>
  <LI><jsp:setProperty name="stringBean"
        property="message"
        value="Best string bean: Fortex" />
      Value after setting property with jsp:setProperty:
      <I><jsp:getProperty name="stringBean"
        property="message" /></I>
  <LI><%= stringBean.setMessage
        ("My favorite: Kentucky Wonder"); %>
      Value after setting property with scriptlet:
      <I><%= stringBean.getMessage() %></I>
</OL>
Using JavaBeans with JSP

1. Initial value (from jsp:getProperty): No message specified
2. Initial value (from JSP expression): No message specified
3. Value after setting property with jsp:setProperty: Best string bean: Portex
4. Value after setting property with scriptlet: My favorite: Kentucky Wonder
Why Using Accessors, Not Public Fields

- To be a bean, you cannot have public fields
- So, you should replace
  ```java
  public double speed;
  ```
  with
  ```java
  private double speed;
  ```
  ```java
  public double getSpeed() {
    return(speed);
  }
  ```
  ```java
  public void setSpeed(double newSpeed) {
    speed = newSpeed;
  }
  ```
- You should do this in all your Java code anyhow. Why?
Why You Should Use Accessors

- You can put constraints on values:
  ```java
  public void setSpeed(double newSpeed) {
      if (newSpeed < 0) {
          sendErrorMessage(...);
          newSpeed = Math.abs(newSpeed);
      }
      speed = newSpeed;
  }
  ```

- You can change your internal representation without changing interface:
  ```java
  // Now using metric units (kph, not mph)
  public void setSpeed(double newSpeed) {
      speedInKPH = convert(newSpeed);
  }
  public void setSpeedInKPH(double newSpeed) {
      speedInKPH = newSpeed;
  }
  ```

New field replaces `speed` but you must convert input in `mph`
Why You Should Use Accessors

- You can perform arbitrary side effects

  public double setSpeed(double newSpeed) {
    speed = newSpeed;
    updateSpeedometerDisplay();
  }

- If users of your class **accessed the fields directly**, then they would each be responsible for executing side effects

- Too much work and runs huge risk of having display inconsistent from actual values.
Setting Bean Properties

```html
<!DOCTYPE ...>
...
<jsp:useBean id="entry"
    class="coreservlets.SaleEntry" />
<%-- setItemID expects a String --%>
<jsp:setProperty
    name="entry"
    property="itemID"
    value='<%= request.getParameter("itemID") %>' />
```

- Here we have used a JSP expression for the `value` attribute.
- JSP attribute values must be fixed strings but for the `value` attribute that is possible.
- Single quote is used because double quote is used in the expression.
Setting Bean Properties: Explicit Conversion & Assignment

```java
int numItemsOrdered = 1;
try {
    numItemsOrdered =
        Integer.parseInt(request.getParameter("numItems"));
} catch(NumberFormatException nfe) {} %>

<%-- setNumItems expects an int --%>

<jsp:setProperty
    name="entry"
    property="numItems"
    value="<%= numItemsOrdered %>>" />
```
Setting Bean Properties: Explicit Conversion & Assignment

```jsp
<%double discountCode = 1.0;
try {
    String discountString = request.getParameter("discountCode");
    discountCode =
        Double.parseDouble(discountString);
} catch(NumberFormatException nfe) {} %>

<%-- setDiscountCode expects a double --%>

<jsp:setProperty
    name="entry"
    property="discountCode"
    value="<%= discountCode %>" />
```
Setting Bean Properties: Explicit Conversion & Assignment

Reading parameters and creating a bean with property values equal to the parameters. Displaying the bean property values.

call

SaleEntry1.jsp

SaleEntry1-Form.jsp

Bean SaleEntry.java
Associating Properties with Input Parameters

- Use the `param` attribute of `jsp:setProperty` to indicate that
  - **Value should come** from specified request parameter
  - Simple automatic type conversion from the `String` type of the parameter should be performed for properties that expect values of standard types
    - `boolean`, `Boolean`, `byte`, `Byte`, `char`, `Character`, `double`, `Double`, `int`, `Integer`, `float`, `Float`, `long`, or `Long`.
Associating Properties with Input Parameters

```jsp
<jsp:useBean id="entry"
    class="coreservlets.SaleEntry" />

<jsp:setProperty
    name="entry"
    property="itemID"
    param="itemID" />

<jsp:setProperty
    name="entry"
    property="numItems"
    param="numItems" />

<jsp:setProperty
    name="entry"
    property="discountCode"
    param="discountCode" />
```

The parameter named "itemID" received by the JSP is passed to the bean "entry" to set the property "itemID".
Associating All Properties with Input Parameters

- Use "*" for the value of the property attribute of \textit{jsp:setProperty} to indicate that
  - \textit{Value should come from request parameter whose name \textbf{matches} property name}
  - Simple automatic type conversion should be performed

\begin{verbatim}
<jsp:useBean id="entry"
    class="coreservlets.SaleEntry" />
<jsp:setProperty name="entry" property="*" />
\end{verbatim}

- This is extremely convenient for making "form beans": objects whose properties are filled in from a form submission.

SaleEntry3.jsp
Warning

- **No action is taken when an input parameter is missing** – the system does not supply a null value so the bean must have default values for properties.

- **Automatic type conversion does not guard against illegal values** as you could do with manual type conversion.

- **Bean property names and request parameters are case sensitive** – the property name and the request parameter name must match exactly.
Sharing Beans

- Up to now the beans we created are bound to local variables in _jspService
- You can use the scope attribute to specify additional places where bean is stored

```jsp
<jsp:useBean id="..." class="..." scope="..." />
```

- Lets multiple servlets or JSP pages share data
- Also permits conditional bean creation
  - Creates new object only if it can't find existing one in the location specified by scope
  - Only if it has not been created by another JSP or servlet.

Values:
- page,
- request,
- session,
- application.
Sharing Beans: Example

- **page1.jsp**
  
  ```jsp
  <jsp:useBean id="foo" class="..." scope="application"/>
  <jsp:setProperty name="foo" property="message" value="Hello"/>
  <jsp:getProperty name="foo" property="message">
  ```

- **page2.jsp**
  
  ```jsp
  <jsp:useBean id="foo" class="..." scope="application"/>
  <jsp:getProperty name="foo" property="message">
  ```

- Possible scenario 1
  - Joe goes to page 2: output is "Default Message"
  - Jane goes to page 1: output is "Hello"

- Possible scenario 2
  - Joe goes to page 1: output is "Hello"
  - Jane goes to page 2: output is "Hello"
Values of the scope Attribute

- **Page:** `<jsp:useBean ... scope="page"/>` or `<jsp:useBean...>`
  - **Default value:** bean object is placed in the `PageContext` object, accessible through the `pageContext` variable, for the duration of the current request (see JSP specifications)
  - Methods in the **generated servlet** can access the bean

- **Application:** `<jsp:useBean ... scope="application"/>`
  - Bean will be stored in ServletContext - available through the `application` variable or by call to `getServletContext()` (method of `HttpServlet`)  
  - ServletContext is **shared by all servlets** in the same Web application.
Values of the scope Attribute

- **Session**: `<jsp:useBean ... scope="session"/>`
  - Bean will be stored in the HttpSession object associated with the current request (session variable), where it can be accessed from regular servlet code with `getAttribute` and `setAttribute`, as with normal session objects.

- **Request**: `<jsp:useBean ... scope="request"/>`
  - Bean object should be placed in the HttpServletRequest object for the duration of the current request, where it is available by means of `getAttribute`.
  - *Do not miss* `getAttribute with getParameter`, *used to access the parameters sent by the client.*
Sharing Beans Four Ways: Bean Code

package coreservlets;

public class BakedBean {
    private String level = "half-baked"; //default value
    private String goesWith = "hot dogs"; //default value

    public String getLevel() {
        return(level);
    }
    public void setLevel(String newLevel) {
        level = newLevel;
    }
    public String getGoesWith() {
        return(goesWith);
    }
    public void setGoesWith(String dish) {
        goesWith = dish;
    }
}

BakedBean.java
Page-Scoped - Unshared

- Create the bean
  - Use `jsp:useBean with scope="page"` (or no scope at all, since page is the default)

- Modify the bean
  - Use `jsp:setProperty with property="*"`
  - This will supply request parameters that match the bean property names (level and goesWith)

- Access the bean
  - Use `jsp:getProperty`
Baked Bean Values: page-based Sharing

<jsp:useBean id="pageBean" class="coreservlets.BakedBean" />
<jsp:setProperty name="pageBean" property="*" />

Bean level:

<jsp:getProperty name="pageBean" property="level" />

Dish bean goes with:

<jsp:getProperty name="pageBean" property="goesWith" />

No scope attribute.
page scope is default.
Result by passing a parameter

- A parameter is passed to the JSP: `goesWith=fish`
Result without passing parameters

- A second call without passing a parameter
- The value of the bean property `goesWith`, set in the previous call is forgotten, the default ("hot dogs") is returned.
Request-Based Sharing

- Create the bean
  - Use `jsp:useBean with scope="request"`
- Modify the bean
  - Use `jsp:setProperty with property="*"`
  - When calling the JSP supply request parameters that match the bean property names
- Access the bean in the 1st (main) page
  - Use `jsp:getProperty`
  - Use `jsp:include (action not the directive) to invoke the second page`
- Access the bean in the 2nd (included) page
  - Use `jsp:useBean with the same id as in the first page, again with scope="request"`
  - Then, use `jsp:getProperty`
<BODY>

<H1>Baked Bean Values: request-based Sharing</H1>

<jsp:useBean id="requestBean"
class="coreservlets.BakedBean"
scope="request" />

<jsp:setProperty name="requestBean"
property="*" />

<H2>Bean level:</H2>

<jsp:getProperty name="requestBean"
property="level" /></H2>

<H2>Dish bean goes with:</H2>

<jsp:getProperty name="requestBean"
property="goesWith" /></H2>

<jsp:include page="BakedBeanDisplay-snippet.jsp" />

</BODY></HTML>
<H1>Repeated Baked Bean Values: request-based Sharing</H1>
<jsp:useBean id="requestBean" class="coreservlets.BakedBean"
    scope="request" />

<H2>Bean level: </H2>
<jsp:getProperty name="requestBean" property="level" />

</H2>

<H2>Dish bean goes with: </H2>
<jsp:getProperty name="requestBean" property="goesWith" />

</H2>
Request-Based Sharing

- The property `goesWith`, set to `fish` in the main JSP is also viewed in the second page (snippet included).
The jsp:param Element: Augmenting Request Parameters

- **Code**
  ```jsp
  <jsp:include page="/fragments/StandardHeading.jsp">
    <jsp:param name="bgColor" value="YELLOW" />
  </jsp:include>
  ```

- **Requested URL**
  ```
  http://host/path/MainPage.jsp?fgColor=RED
  ```

- **Main page**
  - fgColor: RED
  - bgColor: null
  - Regardless of whether you check before or after inclusion

- **Included page**
  - fgColor: RED
  - bgColor: YELLOW
Session-Based Sharing

- Create the bean
  - Use `jsp:useBean with scope="session"`

- Modify the bean
  - Use `jsp:setProperty with property="*"`
  - Then, supply request parameters that match the bean property names

- Access the bean in the initial request
  - Use `jsp:getProperty in the request in which jsp:setProperty is invoked`

- Access the bean later
  - Use `jsp:getProperty in a request that does not include request parameters and thus does not invoke jsp:setProperty`
  - If this request is from the same client the previously modified value is seen.
Session-Based Sharing: Code

...<BODY>
<H1>Baked Bean Values: session-based Sharing</H1>
<jsp:useBean id="sessionBean" class="coreservlets.BakedBean" scope="session" />
<jsp:setProperty name="sessionBean" property="*" />

<H2>Bean level:</H2>
<jsp:getProperty name="sessionBean" property="level" />

</H2>

<H2>Dish bean goes with:</H2>
<jsp:getProperty name="sessionBean" property="goesWith" />

</H2></BODY></HTML>

BakedBeanDisplay-session.jsp
Initial Request

- In the initial request we pass a parameter (goesWith=fish)
Later Request - Same Client

- In a second request we do not pass the parameter but the bean still store the property value set in the previous call

Bean level: half-baked

Dish bean goes with: fish
Application-Based Sharing

- Create the bean
  - Use `jsp:useBean with scope="application"`
- Modify the bean
  - Use `jsp:setProperty with property="*"`
  - Then, supply request parameters that match the bean property names
- Access the bean in the initial request
  - Use `jsp:getProperty in the request in which jsp:setProperty is invoked`
- Access the bean later
  - Use `jsp:getProperty in a request that does not include request parameters and thus does not invoke jsp:setProperty`
  - Whether this request is from the same client or a different client, the previously modified value is seen.
Application-Based Sharing: Code

```html
<BODY>
<H1>Baked Bean Values:
application-based Sharing</H1>
<jsp:useBean id="applicationBean"
class="coreservlets.BakedBean"
scope="application" />
<jsp:setProperty name="applicationBean"
property="*" />

<H2>Bean level:
<jsp:getProperty name="applicationBean"
property="level" />
</H2>

<H2>Dish bean goes with:
<jsp:getProperty name="applicationBean"
property="goesWith"/>
</H2>
</BODY></HTML>
```
Bean property shared between clients

Baked Bean Values: application-based Sharing

Bean level: half-baked
Dish bean goes with: fish
Conditional Bean Operations

- **Bean conditionally created**
  - `jsp:useBean` results in new bean being instantiated **only if no bean with same id and scope can be found**
  - If a bean with same id and scope is found, the preexisting bean is simply bound to variable referenced by id

- **Bean properties conditionally set**
  - `<jsp:useBean ... />` replaced by
    - `<jsp:useBean ...>statements</jsp:useBean>`
    - The statements (`jsp:setProperty` elements) are executed **only** if a new bean is created, not if an existing bean is found.