Integrating Servlets and JSP: The Model View Controller (MVC) Architecture

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/csajsp2.html

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 1.x & JSF 2.0, Struts Classic & Struts 2, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6. Developed and taught by well-known author and developer. At public venues or onsite at your location.

For live Java EE training, please see training courses at http://courses.coreservlets.com/.
Servlets, JSP, Struts Classic, Struts 2, JSF 1.x, JSF 2.0, Ajax (with jQuery, Dojo, Prototype, Ext, etc.), GWT, Java 5, Java 6, Spring, Hibernate/JPA, and customized combinations of topics.

Taught by the author of Core Servlets and JSP, More Servlets and JSP, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization. Contact hall@coreservlets.com for details.
Agenda

• Understanding the benefits of MVC
• Using RequestDispatcher to implement MVC
• Forwarding requests from servlets to JSP pages
• Handling relative URLs
• Choosing among different display options
• Comparing data-sharing strategies

MVC Motivation

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 1.x & JSF 2.0, Struts Classic & Struts 2, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.
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

Why Combine Servlets & JSP?

- Typical picture: use JSP to make it easier to develop and maintain the HTML content
  - For simple dynamic code, call servlet code from scripting elements
  - For slightly more complex applications, use custom classes called from scripting elements
  - For moderately complex applications, use beans and custom tags
- But, that’s not enough
  - For complex processing, starting with JSP is awkward
  - Despite the ease of separating the real code into separate classes, beans, and custom tags, the assumption behind JSP is that a single page gives a single basic look
Possibilities for Handling a Single Request

• **Servlet only. Works well when:**
  – Output is a binary type. E.g.: an image
  – There is no output. E.g.: you are doing forwarding or redirection as in Search Engine example.
  – Format/layout of page is highly variable. E.g.: portal.

• **JSP only. Works well when:**
  – Output is mostly character data. E.g.: HTML
  – Format/layout mostly fixed.

• **Combination (MVC architecture). Needed when:**
  – A single request will result in multiple substantially different-looking results.
  – You have a large development team with different team members doing the Web development and the business logic.
  – You perform complicated data processing, but have a relatively fixed layout.

MVC Misconceptions

• **An elaborate framework is necessary**
  – Frameworks are sometimes useful
    • Struts
    • JavaServer Faces (JSF)
  – They are not required!
    • Implementing MVC with the built-in RequestDispatcher works very well for most simple and moderately complex applications

• **MVC totally changes your overall system design**
  – You can use MVC for individual requests
  – Think of it as the MVC approach, not the MVC architecture
    • Also called the Model 2 approach
Review: 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
    - In this version of MVC, it is not required to have zero arg constructor if you only instantiate from Java code
  - Should have no public instance variables (fields)
    - I hope you already 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 can use `isXxx` instead of `getXxx`
Bean Properties: Examples

<table>
<thead>
<tr>
<th>Method Names</th>
<th>Property Name</th>
<th>Example JSP Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>getFirstName</td>
<td>firstName</td>
<td>&lt;jsp:getProperty ... property=&quot;firstName&quot;/&gt; &lt;jsp:setProperty ... property=&quot;firstName&quot;/&gt; ${customer.firstName}</td>
</tr>
<tr>
<td>setFirstName</td>
<td></td>
<td>&lt;jsp:getProperty ... property=&quot;firstName&quot;/&gt; &lt;jsp:setProperty ... property=&quot;firstName&quot;/&gt; ${customer.firstName}</td>
</tr>
<tr>
<td>isExecutive</td>
<td>executive</td>
<td>&lt;jsp:getProperty ... property=&quot;executive&quot;/&gt; &lt;jsp:setProperty ... property=&quot;executive&quot;/&gt; ${customer.executive}</td>
</tr>
<tr>
<td>setExecutive(boolean)</td>
<td>executive</td>
<td>&lt;jsp:getProperty ... property=&quot;executive&quot;/&gt; &lt;jsp:setProperty ... property=&quot;executive&quot;/&gt; ${customer.executive}</td>
</tr>
<tr>
<td>getExecutive</td>
<td>executive</td>
<td>&lt;jsp:getProperty ... property=&quot;executive&quot;/&gt; &lt;jsp:setProperty ... property=&quot;executive&quot;/&gt; ${customer.executive}</td>
</tr>
<tr>
<td>getZIP</td>
<td>ZIP</td>
<td>&lt;jsp:getProperty ... property=&quot;ZIP&quot;/&gt; &lt;jsp:setProperty ... property=&quot;ZIP&quot;/&gt; ${address.ZIP}</td>
</tr>
<tr>
<td>setZIP</td>
<td></td>
<td>&lt;jsp:getProperty ... property=&quot;ZIP&quot;/&gt; &lt;jsp:setProperty ... property=&quot;ZIP&quot;/&gt; ${address.ZIP}</td>
</tr>
</tbody>
</table>

Note 1: property name does not exist anywhere in your code. It is just a shortcut for the method name.
Note 2: property name is derived only from method name. Instance variable name is irrelevant.

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
  – Eclipse: src/folderMatchingPackage
  – Deployed: .../WEB-INF/classes/folderMatchingPackage
• Beans (and utility classes) must always be in packages!
Basic MVC Design

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 1.x & JSF 2.0, Struts Classic & Struts 2, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6. Developed and taught by well-known author and developer. At public venues or onsite at your location.

MVC Flow of Control

HTML or JSP

Form

Java Code
(Business Logic)

Servlet

Results (beans)

Form

submit form
(Form ACTION matches url-pattern of servlet)

store beans in request, session, or application scope

request.setAttribute("customer", currentCustomer);

invoke appropriate JSP page

JSP1

JSP2

JSP3

(Extract data from beans and put in output)

${customer.firstName}
Implementing MVC with RequestDispatcher

1. **Define beans to represent result data**
   - Ordinary Java classes with at least one `getBlah` method

2. **Use a servlet to handle requests**
   - Servlet reads request parameters, checks for missing and malformed data, calls business logic, etc.

3. **Obtain bean instances**
   - The servlet invokes business logic (application-specific code) or data-access code to obtain the results.

4. **Store the bean in the request, session, or servlet context**
   - The servlet calls `setAttribute` on the request, session, or servlet context objects to store a reference to the beans that represent the results of the request.

---

Implementing MVC with RequestDispatcher (Continued)

5. **Forward the request to a JSP page.**
   - The servlet determines which JSP page is appropriate to the situation and uses the `forward` method of `RequestDispatcher` to transfer control to that page.

6. **Extract the data from the beans.**
   - **JSP 1.2**: the JSP page accesses beans with `jsp:useBean` and a scope matching the location of step 4. The page then uses `jsp:getProperty` to output the bean properties.
   - **JSP 2.0**: the JSP page uses `${nameFromServlet.property}` to output bean properties.
   - Either way, the JSP page does not create or modify the bean; it merely extracts and displays data that the servlet created.
Request Forwarding Example

```java
public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    ... // Do business logic and get data
    String operation = request.getParameter("operation");
    if (operation == null) {
        operation = "unknown";
    }
    String address;
    if (operation.equals("order")) {
        address = "/WEB-INF/Order.jsp";
    } else if (operation.equals("cancel")) {
        address = "/WEB-INF/Cancel.jsp";
    } else {
        address = "/WEB-INF/UnknownOperation.jsp";
    }
    RequestDispatcher dispatcher =
            request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
}
```

jsp:useBean in MVC vs. in Standalone JSP Pages

- The JSP page should not create the objects
  - The servlet, not the JSP page, should create all the data objects. So, to guarantee that the JSP page will not create objects, you should use
    `<jsp:useBean ... type="package.Class" />`
  instead of
    `<jsp:useBean ... class="package.Class" />`

- The JSP page should not modify the objects
  - So, you should use jsp:getProperty but not jsp:setProperty.
Scopes: request, session, and application (ServletContext)

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP 1.x & JSF 2.0, Struts Classic & Struts 2, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

Reminder: jsp:useBean Scope Alternatives (JSP 1.2 Only!)

- **request**
  - `<jsp:useBean id="..." type="..." scope="request" />`

- **session**
  - `<jsp:useBean id="..." type="..." scope="session" />`

- **application**
  - `<jsp:useBean id="..." type="..." scope="application" />`

- **page**
  - `<jsp:useBean id="..." type="..." scope="page" />`
    - or just
    - `<jsp:useBean id="..." type="..." />`
  - This scope is not used in MVC (Model 2) architecture
Request-Based Data Sharing

- **Servlet**
  ValueObject value = new ValueObject(...);
  request.setAttribute("key", value);
  RequestDispatcher dispatcher =
  request.getRequestDispatcher
  ("/WEB-INF/SomePage.jsp");
  dispatcher.forward(request, response);

- **JSP 1.2**
  <jsp:useBean id="key" type="somePackage.ValueObject"
  scope="request" />
  <jsp:getProperty name="key" property="someProperty" />

- **JSP 2.0**
  ${key.someProperty}

Request-Based Data Sharing: Simplified Example

- **Servlet**
  Assume that the Customer constructor handles missing/malformed data.
  Customer myCustomer =
  Lookup.findCust(request.getParameter("customerID"));
  request.setAttribute("customer", myCustomer);
  RequestDispatcher dispatcher =
  request.getRequestDispatcher
  ("/WEB-INF/SomePage.jsp");
  dispatcher.forward(request, response);

- **JSP 1.2**
  <jsp:useBean id="customer" type="somePackage.Customer"
  scope="request" />
  <jsp:getProperty name="customer" property="firstName" />

- **JSP 2.0**
  ${customer.firstName}
  Note: the Customer class must have a method called "getFirstName".
**Session-Based Data Sharing**

- **Servlet**

  ```java
  ValueObject value = SomeLookup.findResult(...);
  HttpSession session = request.getSession();
  session.setAttribute("key", value);
  RequestDispatcher dispatcher = request.getRequestDispatcher("/WEB-INF/SomePage.jsp");
  dispatcher.forward(request, response);
  ```

- **JSP 1.2**

  ```html
  <jsp:useBean id="key" type="somePackage.ValueObject"
  scope="session" />
  <jsp:getProperty name="key" property="someProperty" />
  ```

- **JSP 2.0**

  ```html
  ${key.someProperty}
  ```

**Session-Based Data Sharing: Variation**

- **Redirect to page instead of forwarding to it**
  - Use response.sendRedirect instead of RequestDispatcher.forward

- **Distinctions: with sendRedirect:**
  - User sees JSP URL (user sees only servlet URL with RequestDispatcher.forward)
  - Two round trips to client (only one with forward)

- **Advantage of sendRedirect**
  - User can visit JSP page separately
    - User can bookmark JSP page

- **Disadvantages of sendRedirect**
  - Two round trips to server is more expensive
  - Since user can visit JSP page without going through servlet first, bean data might not be available
    - So, JSP page needs code to detect this situation
ServletContext-Based Data Sharing (Rare)

- **Servlet**
  
  ```java
  synchronized(this) {
    ValueObject value = SomeLookup.findResult(...);
    getServletContext().setAttribute("key", value);
    RequestDispatcher dispatcher = request.getRequestDispatcher
      ("/WEB-INF/SomePage.jsp");
    dispatcher.forward(request, response);
  }
  ```

- **JSP 1.2**
  
  ```
  <jsp:useBean id="key" type="somePackage.ValueObject"
    scope="application" />
  <jsp:getProperty name="key" property="someProperty" />
  ```

- **JSP 2.0**
  
  ```
  ${key.someProperty}
  ```

Relative URLs in JSP Pages

- **Issue:**
  - Forwarding with a request dispatcher is transparent to the client. *Original* URL is only URL browser knows about.

- **Why does this matter?**
  - What will browser do with tags like the following?

  ```
  <img src="foo.gif" ...>
  <link rel="stylesheet"
    href="my-styles.css"
    type="text/css">
  <a href="bar.jsp">...</a>
  ```

  - Browser treats addresses as relative to *servlet URL*
Applying MVC: Bank Account Balances

- **Bean**
  - BankCustomer

- **Business Logic**
  - BankCustomerLookup

- **Servlet that populates bean and forwards to appropriate JSP page**
  - Reads customer ID, calls BankCustomerLookup’s data-access code to obtain BankCustomer
  - Uses current balance to decide on appropriate result page

- **JSP pages to display results**
  - Negative balance: warning page
  - Regular balance: standard page
  - High balance: page with advertisements added
  - Unknown customer ID: error page
public class ShowBalance extends HttpServlet {
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        BankCustomer currentCustomer =
            BankCustomerLookup.getCustomer(request.getParameter("id"));
        request.setAttribute("customer", currentCustomer);
        String address;
        if (currentCustomer == null) {
            address =
                "/WEB-INF/bank-account/UnknownCustomer.jsp";
        } else if (currentCustomer.getBalance() < 0) {
            address =
                "/WEB-INF/bank-account/NegativeBalance.jsp";
        } ...
        RequestDispatcher dispatcher =
            request.getRequestDispatcher(address);
        dispatcher.forward(request, response);
    }
}

public class BankCustomer {
    private final String id, firstName, lastName;
    private final double balance;

    public BankCustomer(String id,
            String firstName,
            String lastName,
            double balance) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.balance = balance;
    }

    // Getters for four instance variables. No setters.

    public double getBalanceNoSign() {
        return (Math.abs(balance));
    }
}
Bank Account Balances: Business Logic

```java
public class BankCustomerLookup {
    private static Map<String, BankCustomer> customers;

    static {
        // Populate Map with some sample customers
    }

    public static BankCustomer getCustomer(String id) {
        return (customers.get(id));
    }
}
```

Bank Account Balances: Input Form

```html
...<fieldset>
    <legend>Bank Account Balance</legend>
    <form action="/show-balance">
        Customer ID: <input type="text" name="id"><br>
        <input type="submit" value="Show Balance">
    </form>
</fieldset>
...```

For the servlet, use the address http://host/appName/show-balance that is set via url-pattern in web.xml.
Bank Account Balances: 
JSP 1.2 Code (Negative Balance)

... 
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
   <TR><TH CLASS="TITLE">
      We Know Where You Live!</TH></TR>
   <P>
   <IMG SRC="/bank-support/Club.gif" ALIGN="LEFT">
   <jsp:useBean id="customer" type="coreservlets.BankCustomer" scope="request" />

   Watch out, 
   <jsp:getProperty name="customer" property="firstName" />, we know where you live.
   
   Pay us the $<jsp:getProperty name="customer" property="balanceNoSign" />
   you owe us before it is too late!
</BODY></HTML>

Bank Account Balances: 
JSP 2.0 Code (Negative Balance)

... 
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
   <TR><TH CLASS="TITLE">
      We Know Where You Live!</TH></TR>
   <P>
   <IMG SRC="/bank-support/Club.gif" ALIGN="LEFT">
   Watch out, ${customer.firstName}, we know where you live.
   
   Pay us the $$${customer.balanceNoSign}$$ you owe us before it is too late!
</BODY></HTML>
Bank Account Balances: web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" ...>
  <!-- Use the URL http://host/app/show-balance instead of http://host/app/servlet/coreservlets.ShowBalance -->
  <servlet>
    <servlet-name>ShowBalance</servlet-name>
    <servlet-class>coreservlets.ShowBalance</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ShowBalance</servlet-name>
    <url-pattern>/show-balance</url-pattern>
  </servlet-mapping>
  ...
</web-app>
```

Bank Account Balances: Results

We Know Where You Live!

Pay us the $3456 if you own it before it is too late!

Your Balance

- First name: Jane
- Last name: Hacker
- ID: id0902
- Balance: $1234.56

Your Balance

It is an honor to serve you, Jane Hacker!

Since you are one of our most valued customers, we would like to offer you the opportunity to spend a mere fraction of your $987654.32 on a boat worthy of your status. Please visit our boat store for more information.
Comparing Data-Sharing Approaches: Request

• Goal
  – Display a random number to the user

• Type of sharing
  – Each request should result in a new number, so request-based sharing is appropriate.

Request-Based Sharing: Bean

```java
package coreservlets;

public class NumberBean {
    private final double num;

    public NumberBean(double number) {
        this.num = number;
    }

    public double getNumber() {
        return num;
    }
}
```

The property name in JSP will be "number". The property name is derived from the method name, not from the instance variable name. Also note the lack of a corresponding setter.
public class RandomNumberServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                        HttpServletResponse response)
            throws ServletException, IOException {
        NumberBean bean =
            RanUtils.getRandomNum(request.getParameter("range"));
        request.setAttribute("randomNum", bean);
        String address = "/WEB-INF/mvc-sharing/RandomNum.jsp";
        RequestDispatcher dispatcher =
            request.getRequestDispatcher(address);
        dispatcher.forward(request, response);
    }
}

public class RanUtils {
    public static NumberBean getRandomNum(String rangeString) {
        double range;
        try {
            range = Double.parseDouble(rangeString);
        } catch(Exception e) {
            range = 10.0;
        }
        return(new NumberBean(Math.random() * range));
    }
}

Request-Based Sharing: URL Pattern (web.xml)

... 
<servlet>
    <servlet-name>RandomNumberServlet</servlet-name>
    <servlet-class>
        coreservlets.RandomNumberServlet
    </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>RandomNumberServlet</servlet-name>
    <url-pattern>/random-number</url-pattern>
</servlet-mapping>
...

Request-Based Sharing: Input Form

... 
<fieldset>
    <legend>Random Number</legend>
    <form action="%random-number">
        Range: <input type="text" name="range"><br/>
        <input type="submit" value="Show Number">
    </form>
</fieldset>
...
Request-Based Sharing: JSP 1.2

```html
...  
<BODY>
<jsp:useBean id="randomNum"
    type="coreservlets.NumberBean"
    scope="request" />
</BODY>
</HTML>
```

Request-Based Sharing: JSP 2.0

```html
...  
<BODY>
</BODY></HTML>
```
Comparing Data-Sharing Approaches: Session

- **Goal**
  - Display users’ first and last names.
  - If the users fail to tell us their name, we want to use whatever name they gave us previously.
  - If the users do not explicitly specify a name and no previous name is found, a warning should be displayed.

- **Type of sharing**
  - Data is stored for each client, so session-based sharing is appropriate.
Session-Based Sharing: Bean

```java
public class NameBean implements Serializable {
    private String firstName = "Missing first name";
    private String lastName = "Missing last name";

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        if (!isMissing(firstName)) {
            this.firstName = firstName;
        }
    }

    ... // getLastName, setLastName

    private boolean isMissing(String value) {
        return (value == null) || (value.trim().equals(""));
    }
}
```

Session-Based Sharing: Servlet

```java
public class RegistrationServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        HttpSession session = request.getSession();
        synchronized(session) {
            NameBean nameBean = (NameBean)session.getAttribute("name");
            if (nameBean == null) {
                nameBean = new NameBean();
                session.setAttribute("name", nameBean);
            }
            nameBean.setFirstName(request.getParameter("firstName"));
            nameBean.setLastName(request.getParameter("lastName"));
            String address = "/WEB-INF/mvc-sharing/ShowName.jsp";
            RequestDispatcher dispatcher =
                request.getRequestDispatcher(address);
            dispatcher.forward(request, response);
        }
    }
}
```
Session-Based Sharing: JSP 1.2

...  

```html
<BODY>
<H1>Thanks for Registering</H1>
<jsp:useBean id="name"
    type="coreservlets.NameBean"
    scope="session" />

<H2>First Name: 
    <jsp:getProperty name="name"
        property="firstName" /></H2>

<H2>Last Name: 
    <jsp:getProperty name="name"
        property="lastName" /></H2>
</BODY></HTML>
```

Session-Based Sharing: JSP 2.0

...  

```html
<BODY>
<H1>Thanks for Registering</H1>

<H2>First Name: ${name.firstName}</H2>

<H2>Last Name: ${name.lastName}</H2>
</BODY></HTML>
```
Session-Based Sharing: Results

Note: url-pattern in web.xml is "register".

Comparing Data-Sharing Approaches: ServletContext

• Goal
  – Display a prime number of a specified length.
  – If the user fails to tell us the desired length, we want to use whatever prime number we most recently computed for any user.

• Type of sharing
  – Data is shared among multiple clients, so application-based sharing is appropriate.
ServletContext-Based Sharing: Bean

package coreservlets;
import java.math.BigInteger;

public class PrimeBean {
    private BigInteger prime;

    public PrimeBean(String lengthString) {
        int length = 150;
        try {
            length = Integer.parseInt(lengthString);
        } catch (NumberFormatException nfe) {} 
        this.prime = Primes.nextPrime(Primes.random(length));
    }

    public BigInteger getPrime() {
        return prime;
    }
    ...
}

ServletContext-Based Sharing: Servlet

public class PrimeServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException, IOException {
        String length = request.getParameter("primeLength");
        ServletContext context = getServletContext();
        synchronized(this) {
            if (((context.getAttribute("primeBean") == null) ||
            (length != null)) {
                PrimeBean primeBean = new PrimeBean(length);
                context.setAttribute("primeBean", primeBean);
            }
        }
        String address = "/WEB-INF/mvc-sharing/ShowPrime.jsp";
        RequestDispatcher dispatcher =
        request.getRequestDispatcher(address);
        dispatcher.forward(request, response);
    }
}
ServletContext-Based Sharing: JSP 1.2

...<BODY>
<H1>A Prime Number</H1>
<jsp:useBean id="primeBean"
    type="coreservlets.PrimeBean"
    scope="application" />
<jsp:getProperty name="primeBean"
    property="prime" />
</BODY></HTML>

ServletContext-Based Sharing: JSP 2.0

...<BODY>
<H1>A Prime Number</H1>
${primeBean.prime}
</BODY></HTML>
ServletContext-Based Sharing: Results

A Prime Number
2984395329555253146348053326235290576965964450721837455981640997842913

Note: url-pattern in web.xml is "prime".

Forwarding and Including

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 1.x & JSF 2.0, Struts Classic & Struts 2, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.
Forwarding from JSP Pages

```jsp
<% String destination;
    if (Math.random() > 0.5) {
        destination = "/examples/page1.jsp";
    } else {
        destination = "/examples/page2.jsp";
    }
%>
<jsp:forward page="<%= destination %>"/>

• Legal, but bad idea
  – Business and control logic belongs in servlets
  – Keep JSP focused on presentation
```

Including Pages Instead of Forwarding to Them

• **With the forward method of RequestDispatcher:**
  – New page generates all of the output
  – Original page cannot generate any output

• **With the include method of RequestDispatcher:**
  – Output can be generated by multiple pages
  – Original page can generate output before and after the included page
  – Original servlet does not see the output of the included page (for this, see later topic on servlet/JSP filters)

  Applications
  • Portal-like applications (see first example)
  • Setting content type for output (see second example)
Using RequestDispatcher.include:

```
response.setContentType("text/html");
String firstTable, secondTable, thirdTable;
if (someCondition) {
    firstTable = "/WEB-INF/Sports-Scores.jsp";
    secondTable = "/WEB-INF/Stock-Prices.jsp";
    thirdTable = "/WEB-INF/Weather.jsp";
} else if (...) { ... }
RequestDispatcher dispatcher =
    request.getRequestDispatcher("/WEB-INF/Header.jsp");
dispenser.include(request, response);
dispenser =
    request.getRequestDispatcher(firstTable);
dispenser.include(request, response);
dispenser =
    request.getRequestDispatcher(secondTable);
dispenser.include(request, response);
dispenser =
    request.getRequestDispatcher(thirdTable);
dispenser.include(request, response);
dispenser =
    request.getRequestDispatcher("/WEB-INF/Footer.jsp");
dispenser.include(request, response);
```

Using RequestDispatcher.include:
Setting Content-Type of Output

```
// From Ajax example
public void doGet(...) ... {
...
    if ("xml".equals(format)) {
        response.setContentType("text/xml");
        outputPage = "/WEB-INF/results/cities-xml.jsp";
    } else if ("json".equals(format)) {
        response.setContentType("application/json");
        outputPage = "/WEB-INF/results/cities-json.jsp";
    } else {
        response.setContentType("text/plain");
        outputPage = "/WEB-INF/results/cities-string.jsp";
    }
    RequestDispatcher dispatcher =
        request.getRequestDispatcher(outputPage);
    dispatcher.include(request, response);
}
```
cities-xml.jsp

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cities>
  <city>
    <name>${cities[0].name}</name>
    <time>${cities[0].shortTime}</time>
    <population>${cities[0].population}</population>
  </city>
  ...
</cities>
```

**Notes**
- Because I use .jsp (not .jspx) and classic JSP syntax, the default content-type is text/html
- I could use `<%@ page contentType="text/xml" %>` here, but it is more convenient to do it in calling servlet and keep this page simple and focused on presentation

**Summary**

**• Use MVC (Model 2) approach when:**
- One submission will result in more than one basic look
- Several pages have substantial common processing

**• Architecture**
- A servlet answers the original request
- Servlet does the real work & stores results in beans
  - Beans stored in HttpServletRequest, HttpSession, or ServletContext
- Servlet forwards to JSP page via forward method of RequestDispatcher
- JSP page reads data from beans
  - JSP 1.2: jsp:useBean with appropriate scope (request, session, or application) plus jsp:getProperty
  - JSP 2.x: `${beanName.propertyName}`
Questions?

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 1.x & JSF 2.0, Struts Classic & Struts 2, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.