



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

JSP, Servlet, Struts, JSF & Java Training: <http://courses.coreservlets.com>  
J2EE Books from Sun Press: <http://www.coreservlets.com>

2

## 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

3

# Uses of JSP Constructs

Simple Application

Complex Application



- 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

4

J2EE training: <http://courses.coreservlets.com>

# 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

5

J2EE training: <http://courses.coreservlets.com>

# 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.

6

J2EE training: <http://courses.coreservlets.com>

# MVC Misconceptions

- **An elaborate framework is necessary**
  - Frameworks are sometimes useful
    - Struts
    - JavaServer Faces (JSF)
  - They are *not* required!
    - Implementing MVC with the builtin 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

7

J2EE training: <http://courses.coreservlets.com>

## 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
  - 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`

8

J2EE training: <http://courses.coreservlets.com>

## Example: StringBean

```
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**
  - .../WEB-INF/classes/*directoryMatchingPackageName*
- **Beans (and utility classes) must *always* be in packages!**

9

J2EE training: <http://courses.coreservlets.com>

## Implementing MVC with RequestDispatcher

- 1. Define beans to represent the data**
- 2. Use a servlet to handle requests**
  - Servlet reads request parameters, checks for missing and malformed data, etc.
- 3. Populate the beans**
  - The servlet invokes business logic (application-specific code) or data-access code to obtain the results. Results are placed in the beans that were defined in step 1.
- 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.

10

J2EE training: <http://courses.coreservlets.com>

## 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.**
  - 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.
  - The JSP page does not create or modify the bean; it merely extracts and displays data that the servlet created.

11

J2EE training: <http://courses.coreservlets.com>

# Request Forwarding Example

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    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);
}
```

12

J2EE training: <http://courses.coreservlets.com>

## 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`.

13

J2EE training: <http://courses.coreservlets.com>

## Reminder: jsp:useBean Scope Alternatives

- **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

14

J2EE training: <http://courses.coreservlets.com>

## 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}
```

15

J2EE training: <http://courses.coreservlets.com>

# Session-Based Data Sharing

- **Servlet**

```
ValueObject value = new ValueObject(...);
HttpSession session = request.getSession();
session.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="session" />
<jsp:getProperty name="key" property="someProperty" />
```

- **JSP 2.0**

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

16

J2EE training: <http://courses.coreservlets.com>

# Session-Based Data Sharing: Variation

- **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
- **Disadvantage of sendRedirect**
  - Since user can visit JSP page without going through servlet first, JSP data might not be available
    - So, JSP page needs code to detect this situation

17

J2EE training: <http://courses.coreservlets.com>



# ServletContext-Based Data Sharing

- **Servlet**

```
synchronized(this) {  
    ValueObject value = new ValueObject(...);  
    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}`
```

18

J2EE training: <http://courses.coreservlets.com>

# 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="JSP-Styles.css"  
      TYPE="text/css">  
<A HREF="bar.jsp">...</A>
```

- Answer: browser treats them as relative to *servlet URL*

- **Simplest solution:**

- Use URLs that begin with a slash

19

J2EE training: <http://courses.coreservlets.com>

# Applying MVC: Bank Account Balances

- **Bean**
  - BankCustomer
- **Servlet that populates bean and forwards to appropriate JSP page**
  - Reads customer ID, calls data-access code to populate 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

20

J2EE training: <http://courses.coreservlets.com>

# Bank Account Balances: Servlet Code

```
public class ShowBalance extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        BankCustomer customer =
            BankCustomer.getCustomer
                (request.getParameter("id"));

        String address;
        if (customer == null) {
            address =
                "/WEB-INF/bank-account/UnknownCustomer.jsp";
        } else if (customer.getBalance() < 0) {
            address =
                "/WEB-INF/bank-account/NegativeBalance.jsp";
            request.setAttribute("badCustomer", customer);
        }
        ...
        RequestDispatcher dispatcher =
            request.getRequestDispatcher(address);
        dispatcher.forward(request, response);
    }
}
```

21

J2EE training: <http://courses.coreservlets.com>

## Bank Account Balances: JSP 1.2 Code (Negative Balance)

```
...
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
    We Know Where You Live!</TABLE>
<P>
<IMG SRC="/bank-support/Club.gif" ALIGN="LEFT">
<jsp:useBean id="badCustomer"
              type="coreservlets.BankCustomer"
              scope="request" />
Watch out,
<jsp:getProperty name="badCustomer"
                  property="firstName" />,
we know where you live.
<P>
Pay us the $<jsp:getProperty name="badCustomer"
                             property="balanceNoSign" />
you owe us before it is too late!
</BODY></HTML>
```

22

J2EE training: <http://courses.coreservlets.com>

## Bank Account Balances: JSP 2.0 Code (Negative Balance)

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

23

J2EE training: <http://courses.coreservlets.com>

# Bank Account Balances: Results

24

J2EE training: <http://courses.coreservlets.com>

# 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

```
package coreservlets;

public class NumberBean {
    private double num = 0;

    public NumberBean(double number) {
        setNumber(number);
    }

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

    public void setNumber(double number) {
        num = number;
    }
}
```

26

J2EE training: <http://courses.coreservlets.com>

## Request-Based Sharing: Servlet

```
public class RandomNumberServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        NumberBean bean =
            new NumberBean(Math.random());
        request.setAttribute("randomNum", bean);
        String address =
            "/WEB-INF/mvc-sharing/RandomNum.jsp";
        RequestDispatcher dispatcher =
            request.getRequestDispatcher(address);
        dispatcher.forward(request, response);
    }
}
```

27

J2EE training: <http://courses.coreservlets.com>

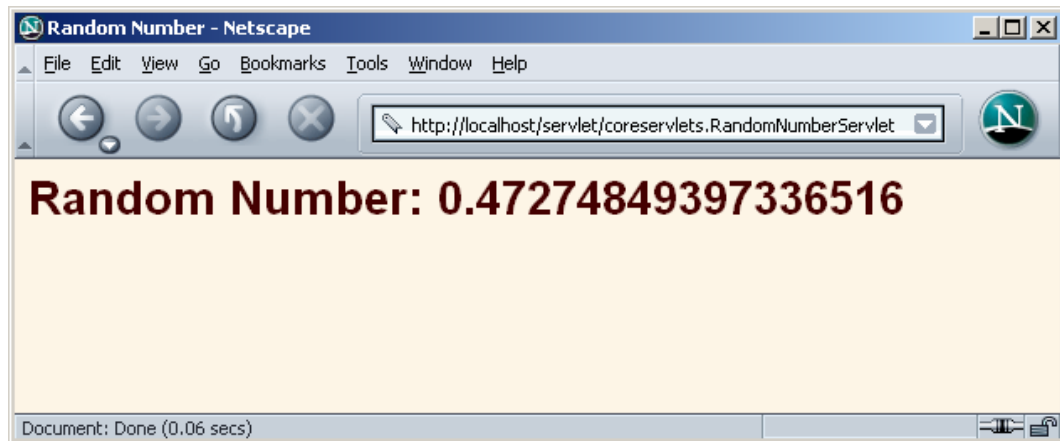
## Request-Based Sharing: JSP 1.2

```
...  
<BODY>  
  <jsp:useBean id="randomNum"  
              type="coreservlets.NumberBean"  
              scope="request" />  
  <H2>Random Number:  
  <jsp:getProperty name="randomNum"  
                  property="number" />  
  
</H2>  
</BODY></HTML>
```

## Request-Based Sharing: JSP 2.0

```
...  
<BODY>  
  <H2>Random Number:  
  ${randomNum.number}  
  </H2>  
</BODY></HTML>
```

# Request-Based Sharing: Results



30

J2EE training: <http://courses.coreservlets.com>

# 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.

31

J2EE training: <http://courses.coreservlets.com>

## Session-Based Sharing: Bean

```
package coreservlets;

public class NameBean {
    private String firstName = "Missing first name";
    private String lastName = "Missing last name";

    public NameBean() {}

    public NameBean(String firstName, String lastName) {
        setFirstName(firstName);
        setLastName(lastName);
    }

    public String getFirstName() {
        return(firstName);
    }

    ...
}
```

32

J2EE training: <http://courses.coreservlets.com>

## Session-Based Sharing: Servlet

```
public class RegistrationServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        HttpSession session = request.getSession();
        NameBean nameBean =
            (NameBean)session.getAttribute("nameBean");
        if (nameBean == null) {
            nameBean = new NameBean();
            session.setAttribute("nameBean", nameBean);
        }
    }
}
```

33

J2EE training: <http://courses.coreservlets.com>



## Session-Based Sharing: Servlet (Continued)

```
String firstName =
    request.getParameter("firstName");
if ((firstName != null) &&
    (!firstName.trim().equals(""))) {
    nameBean.setFirstName(firstName);
}
String lastName =
    request.getParameter("lastName");
if ((lastName != null) &&
    (!lastName.trim().equals(""))) {
    nameBean.setLastName(lastName);
}
String address =
    "/WEB-INF/mvc-sharing/ShowName.jsp";
RequestDispatcher dispatcher =
    request.getRequestDispatcher(address);
dispatcher.forward(request, response);
}
```

34

J2EE training: <http://courses.coreservlets.com>

## Session-Based Sharing: JSP 1.2

```
...
<BODY>
<H1>Thanks for Registering</H1>
<jsp:useBean id="nameBean"
             type="coreservlets.NameBean"
             scope="session" />
<H2>First Name:
<jsp:getProperty name="nameBean"
                 property="firstName" /></H2>
<H2>Last Name:
<jsp:getProperty name="nameBean"
                 property="lastName" /></H2>
</BODY></HTML>
```

35

J2EE training: <http://courses.coreservlets.com>

# Session-Based Sharing: JSP 2.0

```
...  
<BODY>  
<H1>Thanks for Registering</H1>  
<H2>First Name:  
${nameBean.firstName}</H2>  
<H2>Last Name:  
${nameBean.lastName}</H2>  
</BODY></HTML>
```

36

J2EE training: <http://courses.coreservlets.com>

# Session-Based Sharing: Results



37

J2EE training: <http://courses.coreservlets.com>

# 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.

38

J2EE training: <http://courses.coreservlets.com>

# 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) {}
        setPrime(Primes.nextPrime(Primes.random(length)));
    }

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

39

J2EE training: <http://courses.coreservlets.com>

# 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);
        }
    }
}
```

40

J2EE training: <http://courses.coreservlets.com>

# 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>
```

41

J2EE training: <http://courses.coreservlets.com>

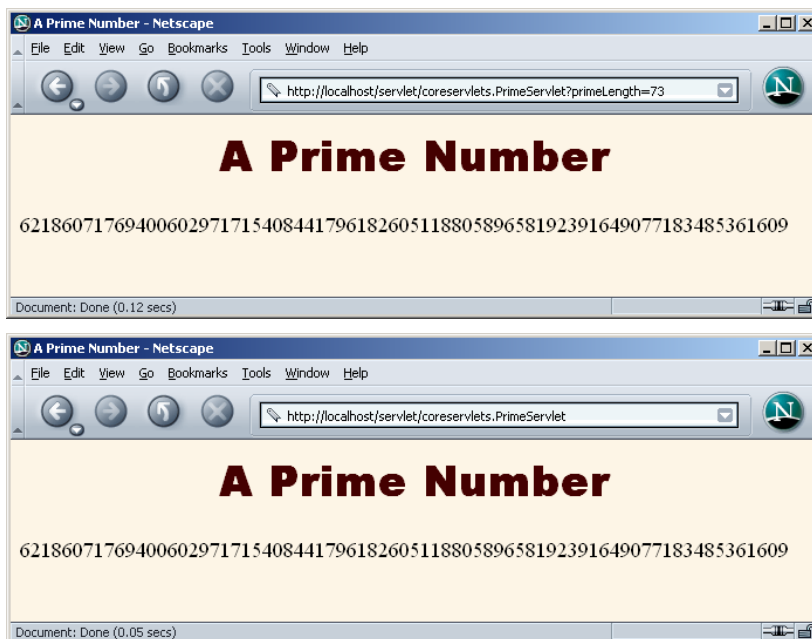
# ServletContext-Based Sharing: JSP 2.0

```
...  
<BODY>  
<H1>A Prime Number</H1>  
${primeBean.prime}  
</BODY></HTML>
```

42

J2EE training: <http://courses.coreservlets.com>

# ServletContext-Based Sharing: Results



43

J2EE training: <http://courses.coreservlets.com>

## Forwarding from JSP Pages

```
<% 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`:**
  - Control is *permanently* transferred to new page
  - Original page *cannot* generate any output
- **With the `include` method of `RequestDispatcher`:**
  - Control is *temporarily* transferred to new page
  - 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)
  - Useful for portals: JSP presents pieces, but pieces arranged in different orders for different users

# Including Pages Instead of Forwarding to Them

```
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");
dispatcher.include(request, response);
dispatcher =
    request.getRequestDispatcher(firstTable);
dispatcher.include(request, response);
dispatcher =
    request.getRequestDispatcher(secondTable);
dispatcher.include(request, response);
dispatcher =
    request.getRequestDispatcher(thirdTable);
dispatcher.include(request, response);
dispatcher =
    request.getRequestDispatcher("/WEB-INF/Footer.jsp");
dispatcher.include(request, response);
```

46

J2EE training: <http://courses.coreservlets.com>

## 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 processing & 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 by means of jsp:useBean with appropriate scope (request, session, or application)

47

J2EE training: <http://courses.coreservlets.com>



# Questions?

**JSP, Servlet, Struts, JSF & Java Training: <http://courses.coreservlets.com>  
J2EE Books from Sun Press: <http://www.coreservlets.com>**