Internet Technologies
10 – Integrating JSP and Servlets

F. Ricci
2010/2011
Content

- Understanding the benefits of MVC
- Using `RequestDispatcher` to implement MVC
- Forwarding requests from servlets to JSP pages
- Handling relative URLs
- Including pages from servlets
- Forwarding pages in JSP
- Expression language
- Accessing scoped variables
- Accessing Bean properties
- Accessing collections
Uses of JSP Constructs

- Simple 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
  - MVC with beans, custom tags, and a framework like Struts or JSF

- Complex Application
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 (lecture 7)
  - Format/layout of page is highly variable

- **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 have a relatively fixed layout, but perform complicated data processing.
Model-View-Controller

- An approach where you break the response into three pieces
  - **The controller:** the part that handles the request, decides what logic to invoke, and decides what JSP page should apply
  - **The model:** the classes that represent the data being displayed
  - **The view:** the JSP pages that represent the output that the client sees

- Examples
  - **MVC using RequestDispatcher** - works very well for most simple and moderately complex applications
  - Struts *(future course)*
  - JavaServer Faces JSF *(future course)*
MVC Flow of Control

HTML or JSP

Form

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

Servlet

(Store beans in request, session, or application scope)

request.setAttribute("customer", currentCustomer);

Java Code
(Business Logic)

Results (beans)

return final result

JSP

JSP1
JSP2
JSP3

(Extract data from beans and put in output)

te:getProperty
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, calls business logic, 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 `ServletContext` objects to store a reference to the beans that represent the results of the request.
Implementing MVC with RequestDispatcher

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` (from the `ServletRequest`) 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.
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);
}
First get the appropriate RequestDispatcher:
```
request.getRequestDispatcher(address);
```

Then the RequestDispatcher can:
- `forward(ServletRequest request,
  ServletResponse response)` - forwards a request to another resource on the same server
- That resource can be a Servlet, JSP page or a simple HTML page
- `include(ServletRequest request,
  ServletResponse response)` - works like a server-side include (SSI) and includes the response from the given resource (Servlet, JSP page, HTML page) within the caller 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
  - 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`. 
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 (Why?)
Request-Based Data Sharing

- **Servlet**

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

- **JSP**

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

  Name chosen by the servlet.

  Name of accessor method, minus the word "get", with next letter changed to lower case.
Session-Based Data Sharing

- **Servlet**
  
  ```java
  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**
  
  ```jsp
  <jsp:useBean id="key" type="somePackage.ValueObject"
  scope="session" />
  <jsp:getProperty name="key" property="someProperty" />
  ```
Session-Based Data Sharing: Variation

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

- **Distinctions: with** `sendRedirect`:
  - With **redirect** 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

Servlet

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

JSP

<jsp:useBean id="key" type="somePackage.ValueObject"
    scope="application" />

<jsp:getProperty name="key"
    property="someProperty" />
Relative URLs in JSP Pages

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

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

```html
<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

- **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
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);
    }
}

The bean contains a small data base of customers in a static variable

BankCustomer.java
ShowBalance.java

call1     call2     call3
We Know Where You Live!

Watch out, {firstName}, we know where you live.

Pay us the \$ {balanceNoSign} you owe us before it is too late!
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>
Bank Account Balances: Results

We Know Where You Live!

Watch out, John, we know where you live.
Pay us the $3456.78 you owe us before it is too late!

Your Balance

- First name: Jane
- Last name: Hacker
- ID: id002
- Balance: $1234.56

Unknown Customer

Unrecognized customer ID.

Your Balance

It is an honor to serve you, Juan 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.
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

- 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

- Applications
  - Portal-like applications (see first example)
  - Including alternative content types for output (see second example)
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);
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("text/javascript");
        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);
}
Expression Language

- When using MVC in JSP 2.x-compliant server change:

  `<jsp:useBean id="someName" type="somePackage.someClass" scope="request, session, or application"/>
  <jsp:getProperty name="someName" property="someProperty"/>

- To:

  `${someName.someProperty}`
Advantages of the Expression Language

- **Concise access to stored objects**
  - To output a "scoped variable" (object stored with `setAttribute` in the PageContext, HttpServletRequest, HttpSession, or ServletContext) named `saleItem`, you use `${saleItem}`

- **Shorthand notation for bean properties**
  - To output the `companyName` property (i.e., result of the `getCompanyName` method) of a scoped variable named `company`, you use `${company.companyName}`
  - To access the `firstName` property of the president property of a scoped variable named `company`, you use `${company.president.firstName}`.
Advantages of the Expression Language

- **Simple access to collection elements**
  - To access an element of an array, List, or Map, you use `${variable[indexOrKey]}`
  - Provided that the index or key is in a form that is legal for Java variable names, the dot notation for beans is interchangeable with the bracket notation

- **Succinct access to request parameters, cookies, and other request data**
  - To access the standard types of request data, you can use one of several predefined implicit objects

- **A small but useful set of simple operators**
  - To manipulate objects within EL expressions, you can use any of several arithmetic, relational, logical, or empty-testing operators.
Advantages of the Expression Language

- **Conditional output**
  - To choose among output options, you do not have to resort to Java scripting elements. Instead, you can use `${test ? option1 : option2}`

- **Automatic type conversion**
  - The expression language removes the need for most typecasts and for much of the code that parses strings as numbers

- **Empty values instead of error messages**
  - In most cases, missing values or NullPointerExceptions result in empty strings, not thrown exceptions.
Accessing Scoped Variables

- **${varName}**
  - Searches the PageContext, the HttpServletRequest, the HttpSession, and the ServletContext, *in that order*, and output the object with that attribute name - PageContext does not apply with MVC

- **Equivalent (alternative) forms**
  1. `${name}`
  2. `<%= pageContext.findAttribute("name") %>`
  3. `<jsp:useBean id="name" type="somePackage.SomeClass" scope="..."><%= name %>`
public class ScopedVars extends HttpServlet {
    public void doGet(HttpServletRequest request,
                        HttpServletResponse response)
            throws ServletException, IOException {
        request.setAttribute("attribute1", "First Value");
        HttpSession session = request.getSession();
        session.setAttribute("attribute2", "Second Value");
        ServletContext application = getServletContext();
        application.setAttribute("attribute3",
                             new java.util.Date());
        request.setAttribute("repeated", "Request");
        session.setAttribute("repeated", "Session");
        application.setAttribute("repeated",
                              "ServletContext");
        RequestDispatcher dispatcher =
            request.getRequestDispatcher("scoped-vars.jsp");
        dispatcher.forward(request, response);
    }
}
Example: Accessing Scoped Variables (cont.)

<!DOCTYPE ...>
...
<TABLE BORDER=5 ALIGN="CENTER">
   <TR><TH CLASS="TITLE">
      Accessing Scoped Variables
   </TABLE>
</TABLE>
<P>
<UL>
   <LI><B>attribute1:</B> ${attribute1}
   <LI><B>attribute2:</B> ${attribute2}
   <LI><B>attribute3:</B> ${attribute3}
   <LI><B>Source of "repeated" attribute:</B> ${repeated}
</UL>
</P>
</BODY></HTML>
Example: Accessing Scoped Variables (Result)

- attribute1: First Value
- attribute2: Second Value
- attribute3: Fri Apr 17 10:46:59 CEST 2009
- Source of "repeated" attribute: Request
Accessing Bean Properties

- \${varName.propertyName}
  - Means to find scoped variable of given name and output the specified bean property

- Equivalent forms:
  1. \${customer.firstName}
  2. < %@ page import="coreservlets.NameBean" %>
     <% NameBean person =
       (NameBean)pageContext.findAttribute
       ("customer");
     %>
     <%= person.getFirstName() %>
Accessing Bean Properties

- **Equivalent forms:**
  - `${customer.firstName}`
  - `<jsp:useBean id="customer" type="coreservlets.NameBean" scope="request, session, or application" />
    <jsp:getProperty name="customer" property="firstName" />

- This is better than script on previous slide
  - But, **requires you to know the scope**
  - **And fails for subproperties:**
  - No non-Java equivalent to:
    - `${customer.address.zipCode}`
public class BeanProperties extends HttpServlet {
    public void doGet(HttpServletRequest request, 
        HttpServletResponse response) 
        throws ServletException, IOException {
        Name name = new Name("Marty", "Hall");
        Company company =
            new Company("coreservlets.com",
                "Java EE Training and Consulting");
        Employee employee =
            new Employee(name, company);
        request.setAttribute("employee", employee); 
        RequestDispatcher dispatcher =
            request.getRequestDispatcher 
                ("/WEB-INF/results/bean-properties.jsp");
        dispatcher.forward(request, response);
    }
}
Example: Accessing Bean Properties

```java
public class Employee {
    private Name name;
    private Company company;

    public Employee(Name name, Company company) {
        setName(name);
        setCompany(company);
    }

    public Name getName() { return(name); }

    public void setName(Name name) {
        this.name = name;
    }

    public CompanyBean getCompany() { return(company); }

    public void setCompany(Company company) {
        this.company = company;
    }
}
```
Example: Accessing Bean Properties

```java
public class Name {
    private String firstName;
    private String lastName;

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

    public String getFirstName() {
        return (firstName);
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return (lastName);
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```
public class Company {
    private String companyName;
    private String business;

    public Company(String companyName, String business) {
        setCompanyName(companyName);
        setBusiness(business);
    }

    public String getCompanyName() { return (companyName); }

    public void setCompanyName(String companyName) {
        this.companyName = companyName;
    }

    public String getBusiness() { return (business); }

    public void setBusiness(String business) {
        this.business = business;
    }
}
Example: Accessing Bean Properties (Cont.)

<!DOCTYPE ...>
...
<UL>
  <LI><B>First Name:</B>  
      ${employee.name.firstName}
  <LI><B>Last Name:</B>  
      ${employee.name.lastName}
  <LI><B>Company Name:</B>  
      ${employee.company.companyName}
  <LI><B>Company Business:</B>  
      ${employee.company.business}
</UL>
</BODY></HTML>
Example: Accessing Bean Properties (Result)

- **First Name:** Marty
- **Last Name:** Hall
- **Company Name:** coreservlets.com
- **Company Business:** J2EE Training and Consulting

call
Equivalence of Dot and Array Notations

- Equivalent forms
  - `${name.property}`
  - `${name["property"]}`

- Reasons for using array notation
  - To access arrays, lists, and other collections
    - See upcoming slides
  - To calculate the property name at request time.
    - `{name1[name2]}` (no quotes around name2)
  - To use names that are illegal as Java variable names
    - `{foo["bar-baz"]}`
    - `{foo["bar.baz"]}`
Accessing Collections

- `${attributeName[entryName]}

Works for

- Array. Equivalent to
  - `theArray[index]`

- List. Equivalent to
  - `theList.get(index)`

- Map. Equivalent to
  - `theMap.get(keyName)`

Equivalent forms (for HashMap)

- `${stateCapitals["maryland"]}`
- `${stateCapitals.maryland}`

But the following is illegal since 2 is not a legal var name: `${listVar.2}`
public class Collections extends HttpServlet {
   public void doGet(HttpServletRequest request,
       HttpServletResponse response)
       throws ServletException, IOException {
   String[] firstNames = { "Bill", "Scott", "Larry" };
   ArrayList lastNames = new ArrayList();
   lastNames.add("Ellison");
   lastNames.add("Gates");
   lastNames.add("McNealy");
   HashMap companyNames = new HashMap();
   companyNames.put("Ellison", "Sun");
   companyNames.put("Gates", "Oracle");
   companyNames.put("McNealy", "Microsoft");
   request.setAttribute("first", firstNames);
   request.setAttribute("last", lastNames);
   request.setAttribute("company", companyNames);
   RequestDispatcher dispatcher = 
       request.getRequestDispatcher("collections.jsp");
   dispatcher.forward(request, response);
   }
}
Example: Accessing Collections (Continued)

<!DOCTYPE ...>
...
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
    Accessing Collections
  </TABLE>
  <P>
  <UL>
    <LI>${first[0]} ${last[0]} (${company["Ellison"]})
    <LI>${first[1]} ${last[1]} (${company["Gates"]})
    <LI>${first[2]} ${last[2]} (${company["McNealy"]})
  </UL>
</BODY></HTML>
Example: Accessing Collections (Result)

- Bill Ellison (Sun)
- Scott Gales (Oracle)
- Larry McNealy (Microsoft)
Referencing Implicit Objects (I)

- **pageContext**: The PageContext object
  - E.g. `${pageContext.session.id}`

- **Using the pageContext object** you can obtain:
  - Request: `pageContext.request`
  - Response: `pageContext.response`
  - Session: `pageContext.session`
  - Out: `pageContext.out`
  - ServletContext: `pageContext.out`

- **param** and **paramValues**: Request params
  - E.g. `${param.custID}`

*The value(s) of custID parameter*
Referencing Implicit Objects (II)

- **header** and **headerValues**: Request headers
  - E.g. `${header.Accept} or ${header["Accept"]}`
  - `${header["Accept-Encoding"]}`

- **cookie**: Cookie object (not cookie value)
  - E.g. `${cookie.userCookie.value} or ${cookie["userCookie"].value}`

- **pageScope, requestScope, sessionScope, applicationScope**
  - Instead of searching scopes
  - `${requestScope.name}` look only in the HttpServletRequest object.
Example: Implicit Objects

<!DOCTYPE ...>
...

<P>
<UL>
  <LI><B>test Request Parameter:</B>  
      ${param.test}
  <LI><B>User-Agent Header:</B>  
      ${header["User-Agent"]}
  <LI><B>JSESSIONID Cookie Value:</B>  
      ${cookie.JSESSIONID.value}
  <LI><B>Server:</B>  
      ${pageContext.servletContext.serverInfo}
</UL>
</BODY></HTML>
Example: Implicit Objects (Result)

- test Request: Parameter:
- User-Agent Header: Mozilla/5.0 (Windows, U, Windows NT 5.1, en-GB, rv:1.9.0.8) Gecko/20000101 Firefox/3.0.8
- JSESSIONID Cookie Value: 4B91ACB0561268EF3C988079BF14C3
- Server: Apache Tomcat/6.0.16

call