The JSF Expression Language

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/JSF-Tutorial/

This somewhat old tutorial covers JSF 1, and is left online for those maintaining existing projects. All new projects should use JSF 2, which is both simpler and more powerful. See http://www.coreservlets.com/JSF-Tutorial/jsf2/.

Customized Java EE Training: http://courses.coreservlets.com/
Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android. Developed and taught by well-known author and developer. At public venues or onsite at your location.

For live training on JSF 1 or 2, please see courses at http://courses.coreservlets.com/.

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.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
  - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services

Contact hall@coreservlets.com for details.
Agenda

• Motivating use of the expression language
  – Comparing to the JSP 2.0 EL
• Accessing bean properties
  – Direct
  – Nested
• Submitting bean properties
  – Expressions in output values
  – Expressions in submission values
  – Expressions for action controllers
• Accessing collection elements
• Using implicit objects and operators

Advantages of the Expression Language (Important)

• Shorthand notation for bean properties.
  – To reference the companyName property (i.e., result of
    the getCompanyName method) of a scoped variable (i.e.
    object stored in request, session, or application scope) or
    managed bean named company, you use
      #{company.companyName}. To reference the firstName
    property of the president property of a scoped variable or
    managed bean named company, you use
      #{company.president.firstName}.
• Simple access to collection elements.
  – To reference 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 for collections.
Advantages of the Expression Language (Less Important)

- **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.

- **Conditional output.**
  - To choose among output options, you do not have to resort to JavaScripting 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.

---

The JSF EL vs. the JSP 2.0 EL

### JSF 1.1 EL
- Can be used only in attributes of JSF tags
- Requires a taglib declaration
- Available in servers supporting JSP 1.2+
  - E.g., WebLogic 8.1, Tomcat 4, Oracle 9i, WebSphere 5
- Uses #{blah}
- Can represent submitted data and output values
- Looks in request, session, application, and managed beans defs

### JSP 2.0 EL
- Can be used anywhere in the JSP page
- Requires no taglib declaration
- Available only in servers supporting JSP 2.0+
  - E.g., WebLogic 9, Tomcat 5 & 6, Oracle 10g, WebSphere 6
- Uses ${blah}
- Represents output values only
- Looks in request, session, and application only
Activating the Expression Language in JSP 2.0

- Available only in servers that support JSP 2.0 or 2.1 (servlets 2.4 or 2.5)
  - E.g., Tomcat 5 or 6, not Tomcat 4
  - See http://theserverside.com/reviews/matrix.tss
- You must use the JSP 2.0 (servlet 2.4) web.xml file
  - The web.xml file in the sample JSF apps uses servlets 2.3 (JSP 1.2)
  - The sample apps at coreservlets.com already use this version, or use any web.xml file distributed with Tomcat 5 or 6.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee web-app_2_4.xsd"
  version="2.4">
  ...
</web-app>
```

Preventing Use of Standard Scripting Elements in JSP 2.

- To enforce EL-only with no scripting, use scripting-invalid in web.xml
  - Still permits both the JSF EL and the JSP 2.0 EL

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee web-app_2_4.xsd"
  version="2.4">
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <scripting-invalid>true</scripting-invalid>
  </jsp-property-group>
</web-app>
```
Downsides to Preventing Use of Scripting Elements

• Harder debugging
  – <% System.out.println("...."); %>

• No redirects
  – <% response.sendRedirect("welcome.faces"); %>

• Some techniques hard to do with MVC
  – <%
    if (outputShouldBeExcel()) {
      response.setContentType("application/vnd.ms-excel");
    }
  %>

• Just because scripting is usually bad does not mean it is always bad
Outputting Bean Properties

• `{varName.propertyName}`
  – Means to search the HttpServletRequest, the HttpSession, the ServletContext (i.e. look for a scoped variable), and managed beans definitions, *in that order*, and output the specified bean property
  – Must be used in attribute of a JSF tag

• Equivalent forms
  – `<h:outputText value="#{customer.firstName}"/>`
    • Works with all JSF versions. Scoped variable or managed bean.
  – `${customer.firstName}`
    • Works only with JSP 2.0 and later. Scoped variable only.
  – `<%@ page import="coreservlets.NameBean" %>
   <% NameBean person = (NameBean)pageContext.findAttribute("customer"); %>
   <%= person.getFirstName() %>`
    • Ugly pre-EL version.

Bean Properties Example: TestBean

```java
package coreservlets;

import java.util.*;

public class TestBean {
    private Date creationTime = new Date();
    private String greeting = "Hello";

    public Date getCreationTime() {
        return(creationTime);
    }

    public String getGreeting() {
        return(greeting);
    }

    public double getRandomNumber() {
        return(Math.random());
    }
}
```
Bean Properties Example: faces-config.xml

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE ...>

<faces-config>
  <managed-bean>
    <managed-bean-name>testBean</managed-bean-name>
    <managed-bean-class>
      coreservlets.TestBean
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
  ...
</faces-config>
```

Bean Properties Example: bean-properties.jsp (.faces)

```jsp
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
  ...
</f:view>

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">Accessing Bean Properties</TH></TR>
</TABLE>
<UL>
  <LI>Creation time:  
    <h:outputText value="#{testBean.creationTime}"/>
  <LI>Greeting:  
    <h:outputText value="#{testBean.greeting}"/>
  <LI>Random number:  
    <h:outputText value="#{testBean.randomNumber}"/>
</UL>
</BODY></HTML>
</f:view>
```
Bean Properties Example:

Result

Accessing Bean Properties

- Creation time: Mon Mar 07 07:21:07 GMT-05:00 2005
- Greeting: Hello
- Random number: 0.6007462465168825
Nested Bean Properties

- \#{varName.prop1.prop2}
  - First searches scoped variables and managed beans definitions for an entry named varName
  - Then accesses prop1 property
    - i.e., calls getProp1 method
  - Then accesses prop2 property of that result
    - i.e., calls getProp2 on the output of getProp1
  - Can be nested arbitrarily

Nested Properties Example: NameBean

```java
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);
    }

    public void setFirstName(String newFirstName) {
        firstName = newFirstName;
    }
    ...
}
```
Nested Properties Example: CompanyBean

```java
package coreservlets;

public class CompanyBean {
    private String companyName;
    private String business;

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

    public String getCompanyName() { return(companyName); }
    public void setCompanyName(String newCompanyName) {
        companyName = newCompanyName;
    }
    ...
}
```

Nested Properties Example: EmployeeBean

```java
package coreservlets;

public class EmployeeBean {
    private NameBean name;
    private CompanyBean company;

    public EmployeeBean(NameBean name, CompanyBean company) {
        setName(name);
        setCompany(company);
    }

    public EmployeeBean() {
        this(new NameBean("Marty", "Hall"),
             new CompanyBean("coreservlets.com",
                              "J2EE Training and Consulting"));
    }

    public NameBean getName() { return(name); }
    public void setName(NameBean newName) {
        name = newName;
    }
    ...
}
```
Nested Properties Example: faces-config.xml

```xml
<faces-config>
  ...
  <managed-bean>
    <managed-bean-name>employee</managed-bean-name>
    <managed-bean-class>
      coreservlets.EmployeeBean
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
  ...
</faces-config>
```

Nested Properties Example: nested-properties.jsp (.faces)

```jsp
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
  ...
  <BODY>
    <TABLE BORDER=5 ALIGN="CENTER">
      <TR><TH CLASS="TITLE">Using Nested Bean Properties</TH></TR>
    </TABLE>
    <UL>
      <LI>Employee's first name:
        <h:outputText value="#{employee.name.firstName}"/>
      <LI>Employee's last name:
        <h:outputText value="#{employee.name.lastName}"/>
      <LI>Name of employee's company:
        <h:outputText value="#{employee.company.companyName}"/>
      <LI>Business area of employee's company:
        <h:outputText value="#{employee.company.business}"/>
    </UL>
  </BODY></HTML>
</f:view>
```
Nested Properties Example: Result

Using Nested Bean Properties

- Employee's first name: Marty
- Employee's last name: Hall
- Name of employee's company: coreservlets.com
- Business area of employee's company: J2EE Training and Consulting

Submitting Bean Properties
Three Meanings of #{...}

- **Designating output value**
  - #{varName.propertyName} means to output the given property of the given scoped variable or managed bean
  - `<h:outputText value="#{employee.address}"/>
    • Anytime accessed, means to output text
  - `<h:inputText value="#{employee.address}"/>
    • When form initially displayed, means to prepopulate field

- **Designating submitted value**
  - `<h:inputText value="#{employee.address}"/>
    • When form submitted, designates where value stored

- **Designating method call after submission**
  - `<h:commandButton value="Button Label"
      action="#{employee.processEmployee}"/>
    • When form submitted, designates action handler

JSP 2.0 and Struts Equivalents

- **Designating output value**
  - `<h:outputText value="#{employee.address}"/>
    • Similar to ${employee.address}, but scoped vars only
    • Similar to `<bean:write name="employee" property="address"/> but scoped vars only
  - `<h:inputText value="#{employee.address}"/>
    • Similar to JSP 2.0
      `<INPUT TYPE="TEXT"...VALUE="${employee.address}"
    • Similar to html:text in Struts

- **Designating submitted value**
  - No JSP 2.0 equivalent
  - Similar to html:text in Struts

- **Designating method call after submission**
  - No JSP 2.0 or Struts equivalent
Submitting Properties Example: EmployeeBean

```java
package coreservlets;

public class EmployeeBean {
    private NameBean name;
    private CompanyBean company;

    ...

    public String processEmployee() {
        if (Math.random() < 0.5) {
            return("accept");
        } else {
            return("reject");
        }
    }
}
```

Nested Properties Example: faces-config.xml

```xml
<faces-config>
    ...
    <managed-bean>
        <managed-bean-name>employee</managed-bean-name>
        <managed-bean-class>
            coreservlets.EmployeeBean
        </managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>
    ...
    <navigation-rule>
        <from-view-id>/submitting-properties.jsp</from-view-id>
        <navigation-case>
            <from-outcome>accept</from-outcome>
            <to-view-id>/WEB-INF/results/accept.jsp</to-view-id>
        </navigation-case>
        <navigation-case>
            <from-outcome>reject</from-outcome>
            <to-view-id>/WEB-INF/results/reject.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>
</faces-config>
```
Submitting Properties Example: submitting-properties.jsp (.faces)

... 
<h:form>
Your first name:  
<h:inputText value="#{employee.name.firstName}"/>

Your last name:  
<h:inputText value="#{employee.name.lastName}"/>

Name of your company:  
<h:inputText value="#{employee.company.companyName}"/>

Business area of your company:  
<h:inputText value="#{employee.company.business}"/>

<h:commandButton value="Process" action="#{employee.processEmployee}"/>

</h:form>

Submitting Properties Example: Input Page Result
Submitting Properties Example: accept.jsp (JSF-Only Version)

```html
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
...

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">Employee Accepted</TH></TR>
</TABLE>
<UL>
  <LI>Employee's first name:
      <h:outputText value="#{employee.name.firstName}"/>
  <LI>Employee's last name:
      <h:outputText value="#{employee.name.lastName}"/>
  <LI>Name of employee's company:
      <h:outputText value="#{employee.company.companyName}"/>
  <LI>Business area of employee's company:
      <h:outputText value="#{employee.company.business}"/>
</UL>
Congratulations.
</BODY></HTML>
</f:view>
```

Submitting Properties Example: accept.jsp (JSP 2.0 Version)

```html
...

<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">Employee Accepted</TH></TR>
</TABLE>
<UL>
  <LI>Employee's first name:
      ${employee.name.firstName}
  <LI>Employee's last name:
      ${employee.name.lastName}
  <LI>Name of employee's company:
      ${employee.company.companyName}
  <LI>Business area of employee's company:
      ${employee.company.business}
</UL>
Congratulations.
</BODY></HTML>
```
Submitting Properties Example: reject.jsp (JSF-Only Version)

```html
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
  ...
  <BODY>
  <TABLE BORDER=5 ALIGN="CENTER">
    <TR><TH CLASS="TITLE">Employee Rejected</TH></TR>
  </TABLE>
  <UL>
    <LI>Employee's first name:
      <h:outputText value="#{employee.name.firstName}"/>
    <LI>Employee's last name:
      <h:outputText value="#{employee.name.lastName}"/>
    <LI>Name of employee's company:
      <h:outputText value="#{employee.company.companyName}"/>
    <LI>Business area of employee's company:
      <h:outputText value="#{employee.company.business}"/>
  </UL>
  Congratulations.
  </BODY></HTML>
</f:view>
```

Submitting Properties Example: reject.jsp (JSP 2.0 Version)

```html
...  
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">Employee Rejected</TH></TR>
</TABLE>
<UL>
  <LI>Employee's first name:
      ${employee.name.firstName}
  <LI>Employee's last name:
      ${employee.name.lastName}
  <LI>Name of employee's company:
      ${employee.company.companyName}
  <LI>Business area of employee's company:
      ${employee.company.business}
</UL>
Congratulations.
</BODY></HTML>
```
Submitting Properties Example: Results

- Employee’s last name: Gates
- Name of employee’s company: microsoft.com
- Business area of employee’s company: Wielding Monopoly

Congratulations.

Submitting Properties Example: Results (Continued)

- Employee’s last name: Ellison
- Name of employee’s company: oracle.com
- Business area of employee’s company: Squashing Competitors

Go away.
Accessing Collections

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]` (getting and setting)
  - List. Equivalent to
    - `theList.get(index)` or `theList.set(index, submitted-val)`
  - Map. Equivalent to
    - `theMap.get(key)` or `theMap.put(key, submitted-val)`
- **Equivalent forms (for HashMap)**
  - `{stateCapitals["maryland"]}`
  - `{stateCapitals.maryland}`
  - But the following is illegal since 2 is not a legal var name
    - `{listVar.2}`

Collections Example: PurchaseBean

```java
public class PurchaseBean {
    private String[] cheapItems = {
        "Gum", "Yo-yo", "Pencil"
    };
    private List<String> mediumItems =
        new ArrayList<String>();
    private Map<String,String> valuableItems =
        new HashMap<String,String>();
    private boolean isEverythingOK = true;

    public PurchaseBean() {
        mediumItems.add("iPod");
        mediumItems.add("GameBoy");
        mediumItems.add("Cell Phone");
        valuableItems.put("low", "Lamborghini");
        valuableItems.put("medium", "Yacht");
        valuableItems.put("high", "Chalet");
    }

    public String[] getCheapItems() {
        return(cheapItems);
    }
    public List<String> getMediumItems() {
        return(mediumItems);
    }
    public Map<String,String> getValuableItems() {
        return(valuableItems);
    }
}
Collections Example:  
PurchaseBean (Continued)

```java
public String purchaseItems() {
    isEverythingOK = Utils.doBusinessLogic(this);
    isEverythingOK = Utils.doDataAccessLogic(this);
    if (isEverythingOK) {
        return("success");
    } else {
        return("failure");
    }
}
```

Collections Example:  
Utils

```java
public class Utils {
    public static boolean doBusinessLogic
                (PurchaseBean bean) {
        // Business logic omitted
        return(Math.random() > 0.1);
    }

    public static boolean doDataAccessLogic
                (PurchaseBean bean) {
        // Database access omitted
        return(Math.random() > 0.1);
    }
}
```
Collections Example: faces-config.xml

```xml
<faces-config>
  ...
  <managed-bean>
    <managed-bean-name>purchases</managed-bean-name>
    <managed-bean-class>coreservlets.PurchaseBean</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
  ...
  <navigation-rule>
    <from-view-id>/using-collections.jsp</from-view-id>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/WEB-INF/results/success.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>failure</from-outcome>
      <to-view-id>/WEB-INF/results/failure.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```

Collections Example: using-collections.jsp (.faces)

```xml
...  
<h:form>
  <UL>
    <LI><B>Cheap Items</B></LI>
    <OL> 
      <LI><h:inputText value="#{purchases.cheapItems[0]}"/>
      <LI><h:inputText value="#{purchases.cheapItems[1]}"/>
      <LI><h:inputText value="#{purchases.cheapItems[2]}"/>
    </OL>
    <LI><B>Medium Items</B></LI>
    <OL>
      <LI><h:inputText value="#{purchases.mediumItems[0]}"/>
      <LI><h:inputText value="#{purchases.mediumItems[1]}"/>
      <LI><h:inputText value="#{purchases.mediumItems[2]}"/>
    </OL>
  </UL>
</h:form>
```
<LI><B>Valuable Items</B>
    <UL>
        <LI>Low:
            <h:inputText value='#{purchases.valuableItems["low"]}'/>
        <LI>Medium:
            <h:inputText value='#{purchases.valuableItems["medium"]}'/>
        <LI>High:
            <h:inputText value='#{purchases.valuableItems["high"]}'/>
    </UL>
</UL>
<h:commandButton value="Purchase"
    action="#{purchases.purchaseItems}"/>
</h:form>
...

• **Important note**
  – Since I am using double quotes around the hash table key,
    I have to use single quotes around the entire JSF expression
Submitting Properties Example: success.jsp (JSF-Only Version)

```jsp
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>

<UL>
<LI><B>Cheap Items</B>
<OL>
<LI><h:outputText value="#{purchases.cheapItems[0]}"/>
<LI><h:outputText value="#{purchases.cheapItems[1]}"/>
<LI><h:outputText value="#{purchases.cheapItems[2]}"/>
</OL>

<LI><B>Medium Items</B>
<OL>
<LI><h:outputText value="#{purchases.mediumItems[0]}"/>
<LI><h:outputText value="#{purchases.mediumItems[1]}"/>
<LI><h:outputText value="#{purchases.mediumItems[2]}"/>
</OL>

...</f:view>
```

Submitting Properties Example: success.jsp (JSP 2.0 Version)

```jsp...
<UL>
<LI><B>Cheap Items</B>
<OL>
<LI>${purchases.cheapItems[0]}
<LI>${purchases.cheapItems[1]}
<LI>${purchases.cheapItems[2]}
</OL>

<LI><B>Medium Items</B>
<OL>
<LI>${purchases.mediumItems[0]}
<LI>${purchases.mediumItems[1]}
<LI>${purchases.mediumItems[2]}
</OL>

...</jsp>```
Submitting Properties Example: Results

Success

- Cheap Items
  1. Paper clip
  2. Straw
  3. Pen
- Medium Items
  1. Walkman
  2. XBox
  3. Cell Phone
- Valuable Items
  - Low Ferrari
  - Medium Yacht
  - High Core Servlets and JavaServer Pages

Failure

- Cheap Items
  1. Rubber band
  2. Notebook
  3. Whistle
- Medium Items
  1. Paperback
  2. Radio
  3. Satellite Phone
- Valuable Items
  - Low: Volkswagen
  - Medium: Winnebago
  - High: Teach Yourself .NET in 24 Hours

Implicit Objects and Operators

Customized Java EE Training: http://courses.coreservlets.com/
Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android. Developed and taught by well-known author and developer. At public venues or onsite at your location.
JSF EL Has Almost the Same Predefined Variables as JSP 2

- **facesContext. The FacesContext object.**
  - E.g. `#{facesContext.externalContext.session.id}`
- **param and paramValues. Request params.**
  - E.g. `#{param.custID}`
- **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}`
- **initParam. Context initialization param.**
- **requestScope, sessionScope, applicationScope.**
  - Instead of searching scopes.

**Problem**

- Using implicit objects usually works poorly with MVC model

---

Example: Implicit Objects

```html
<!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)

Expression Language Operators

- **Arithmetic**
  - + - * / div % mod

- **Relational**
  - == eq != ne < lt > gt <= le >= ge

- **Logical**
  - && and || or ! Not

- **Empty**
  - Empty
  - True for null, empty string, empty array, empty list, empty map. False otherwise.

- **CAUTION**
  - Use extremely sparingly to preserve MVC model
### Example: Operators

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3 + 2 - 1$</td>
<td>4</td>
</tr>
<tr>
<td>$1 &lt; 2$</td>
<td>true</td>
</tr>
<tr>
<td>$1 + 2 * 3 + 3/4$</td>
<td>7.75</td>
</tr>
<tr>
<td>$2/3 &gt;= 3/2$</td>
<td>true</td>
</tr>
<tr>
<td>$3 % 2$</td>
<td>1</td>
</tr>
<tr>
<td>$3/4 == 0.75$</td>
<td>true</td>
</tr>
</tbody>
</table>

### Example: Operators (Result)

![EL Operators](http://localhost:8080/operators.png)
Evaluating Expressions Conditionally

- `{$ test ? expression1 : expression2 }`
  - Evaluates test and outputs either expression1 or expression2

**Problems**
- Relatively weak
  - `c:if` and `c:choose` from JSTL are much better
  - Tempts you to put business/processing logic in JSP page.
  - Should only be used for presentation logic.
    - Even then, consider alternatives

---

Example: Conditional Expressions

```java
public class Conditionals extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        SalesBean apples =
            new SalesBean(150.25, -75.25, 22.25, -33.57);
        SalesBean oranges =
            new SalesBean(-220.25, -49.57, 138.25, 12.25);
        request.setAttribute("apples", apples);
        request.setAttribute("oranges", oranges);
        RequestDispatcher dispatcher =
            request.getRequestDispatcher("/el/conditionals.jsp");
        dispatcher.forward(request, response);
    }
}
```
Example: Conditional Expressions (Continued)

```java
public class SalesBean {
    private double q1, q2, q3, q4;

    public SalesBean(double q1Sales,
                      double q2Sales,
                      double q3Sales,
                      double q4Sales) {
        q1 = q1Sales; q2 = q2Sales;
        q3 = q3Sales; q4 = q4Sales;
    }

    public double getQ1() { return(q1); }
    public double getQ2() { return(q2); }
    public double getQ3() { return(q3); }
    public double getQ4() { return(q4); }
    public double getTotal() {
        return(q1 + q2 + q3 + q4);
    }
}
```

Example: Conditional Expressions (Continued)

```html
...<table border=1 align="center">
  <tr><th>
    Apples
  </th><th class="colored">
    Oranges
  </th>
</tr>
<tr class="colored">
  <td align="right">
    \$\{apples.q1\}
  </td><td align="right">
    \$\{oranges.q1\}
  </td>
</tr>
<tr class="colored">
  <td align="right">
    \$\{apples.q2\}
  </td><td align="right">
    \$\{oranges.q2\}
  </td>
</tr>
<tr class="colored">
  <td align="right">
    bgcolor="{(apples.total < 0) ? "RED" : "WHITE" }">
    \$\{apples.total\}
  </td><td align="right">
    bgcolor="{(oranges.total < 0) ? "RED" : "WHITE" }">
    \$\{oranges.total\}
  </td>
</tr>
</table>...
```
Example: Conditional Expressions (Result)

Conditional Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Apples</th>
<th>Oranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Quarter</td>
<td>150.25</td>
<td>-220.25</td>
</tr>
<tr>
<td>Second Quarter</td>
<td>-75.25</td>
<td>-49.57</td>
</tr>
<tr>
<td>Third Quarter</td>
<td>22.25</td>
<td>138.25</td>
</tr>
<tr>
<td>Fourth Quarter</td>
<td>-33.57</td>
<td>12.25</td>
</tr>
<tr>
<td>Total</td>
<td>63.68</td>
<td>-119.32</td>
</tr>
</tbody>
</table>

Summary

- The JSF EL provides concise, easy-to-read access to
  - Bean properties
  - Collection elements
- Plays Triple Role
  - Output values
  - Submitted values
  - Action handlers
- JSF EL for input values similar to JSP 2 EL
  - Except JSF EL accesses managed beans even if they are not yet scoped variables
  - Submitted values and action handlers: no JSP 2.0 equiv
Questions?

Customized Java EE Training: http://courses.coreservlets.com/
Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android. Developed and taught by well-known author and developer. At public venues or onsite at your location.