Simplifying Access to Java Code: JSP 2.0 Expression Language

Uses of JSP Constructs

$$\begin{tikzcd}\text{Simple Application} \arrow[d] \\ \text{Complex Application}\end{tikzcd}$$

- 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, and a framework like Struts or JSF

Why Combine Servlets & JSP?

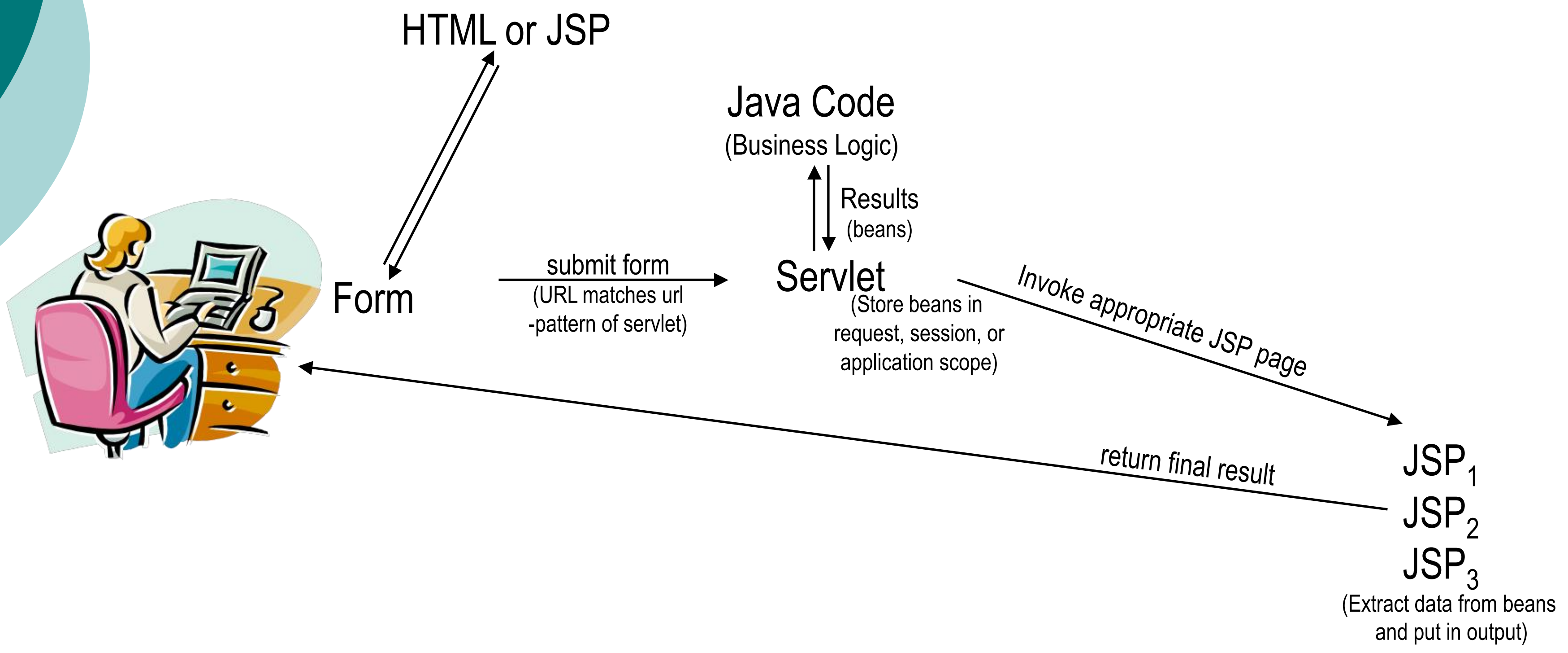
- Typical picture: use JSP to make it easier to develop and maintain the HTML content:
  - For simple dynamic code, call servlet code from scripting elements.
  - For slightly more complex applications, use custom classes called from scripting elements.
  - For moderately complex applications, use beans.

- 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.

Servlets and JSP: 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
- 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 Flow of Control



Implementing MVC with RequestDispatcher

1. Define beans to represent the data
2. Use a servlet to handle requests
3. Populate the beans
4. Store the bean in the request, session, or servlet context
9. Forward the request to a JSP page.
11. Extract the data from the beans.

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);
}
```

Note: When you use the forward method of RequestDispatcher, the client sees the URL of the original servlet, not the URL of the final JSP page.

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}
- 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 for collections.

Advantages of the Expression Language …

- 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 Java scripting elements. Instead, you can use
    - ${test ? option1 : option2}.

Advantages of the Expression Language …

- 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.

Invoking the Expression Language

- Basic form: ${expression}
  - These EL elements can appear in ordinary text or in JSP tag attributes, provided that those attributes permit regular JSP expressions. For example:
    - <UL> <LI>Name: ${expression1}
    -   <LI>Address: ${expression2}
    - </UL>
    - <jsp:include page="${expression3}" />

- The EL in tag attributes
  - You can use multiple expressions (possibly intermixed with static text) and the results are coerced to strings and concatenated. For example:
    - <jsp:include page="${expr1}blah ${expr2}" />

Escaping Special Characters

- To get ${ in the page output
  - Use \${ in the JSP page.
- To get a single quote within an EL expression
  - Use \'
- To get a double quote within an EL expression
  - Use \"

Accessing Scoped Variables

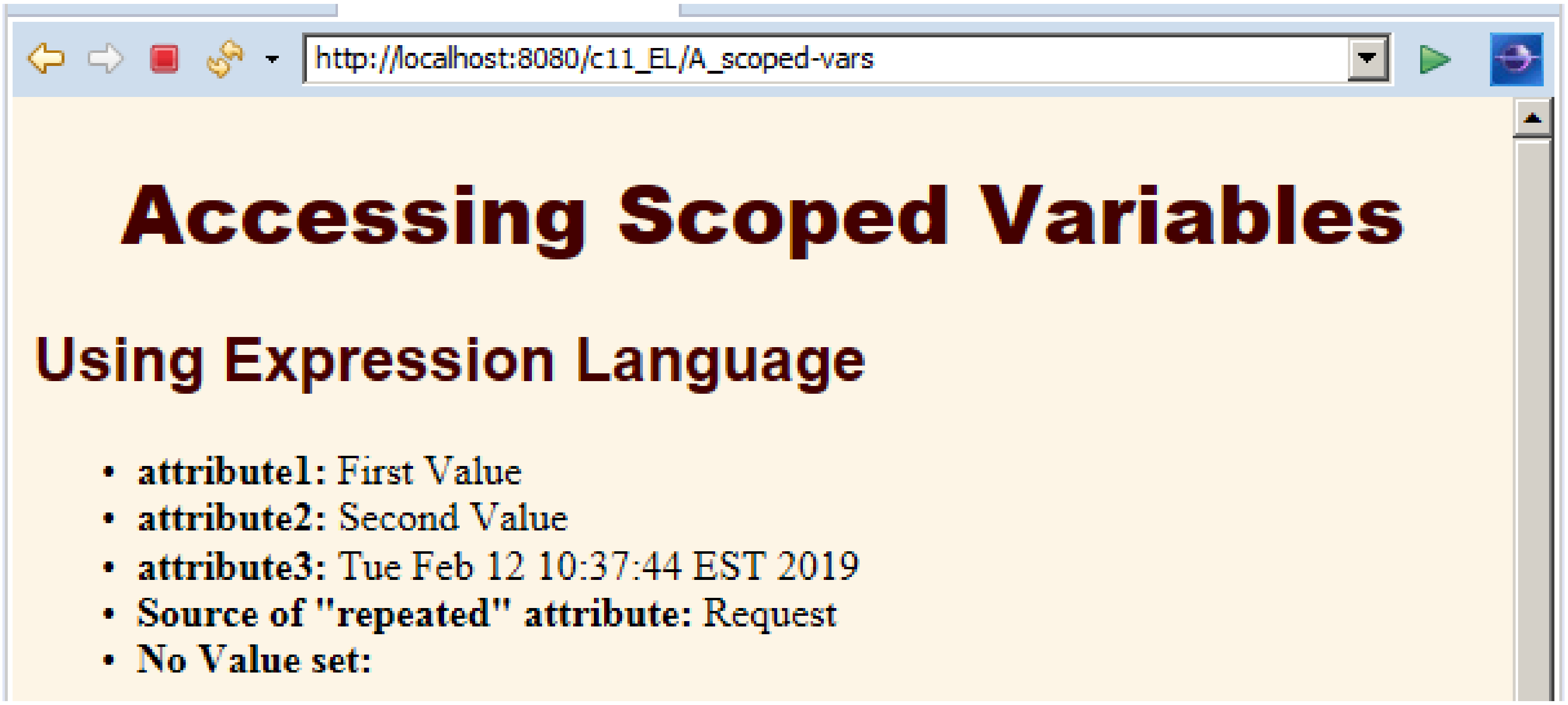
- ${varName}
  - Means to **search** the PageContext, the HttpServletRequest, the HttpSession, and the ServletContext, *in that order*, and output the object with that attribute name.
  - PageContext does not apply with MVC.
- Equivalent forms
  - ${name}
  - <%= pageContext.findAttribute("name") %>
  - <jsp:useBean id="name"
                        type="somePackage.SomeClass"
                        scope="...">
    <%= name %>

JSP/EL Naming Access Scope

```java
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);
```

```html
<!DOCTYPE html>
<html>
<head><title>Accessing Scoped Variables</title>
</head>
<body>
<table border=5 align="center”><tr><th class="title">
  Accessing Scoped Variables
</table>
<p>
<ul>
  <li>attribute1: ${attribute1}
  <li>attribute2: ${attribute2}
  <li>attribute3: ${attribute3}
  <li>Source of "repeated" attribute: ${repeated}
  <li>< strong >No value set: ${nothing}</li>
</ul>
</body></html>
```

Example: Accessing Scoped Variables (Result)



http://localhost:8080/c11_EL/A_scoped-vars

Accessing Scoped Variables

Using Expression Language

- **attribute1:** First Value
- **attribute2:** Second Value
- **attribute3:** Tue Feb 12 10:37:44 EST 2019
- **Source of "repeated" attribute:** Request
- **No Value set:**

Accessing Bean Properties

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

  -
    ```
    <%@ 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}

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"]}

Example: Accessing Bean Properties - Name

```
package beansRobjects;

public class NameBean {
  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; }
  }

  public String getLastName() {return(lastName); }

  public void setLastName(String lastName) {
    if (!isMissing(lastName)) { this.lastName = lastName; }
  }

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

Example: Accessing Bean Properties - Company

```
package beansRobjects;

public class CompanyBean {

  private String companyName;
  private String business;

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

  public String getBusiness() { return(business); }
  public void setBusiness(String newBusiness) {
    this.business = newBusiness;  }

}
```

Example: Accessing Bean Properties - Employee

```
package beansRobjects;

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

  public NameBean getName() { return(name); }
  public void setName(NameBean newName) {
    name = newName;}

  public CompanyBean getCompany(){return(company);}
  public void setCompany(CompanyBean newCompany) {
    company = newCompany;  }}
```

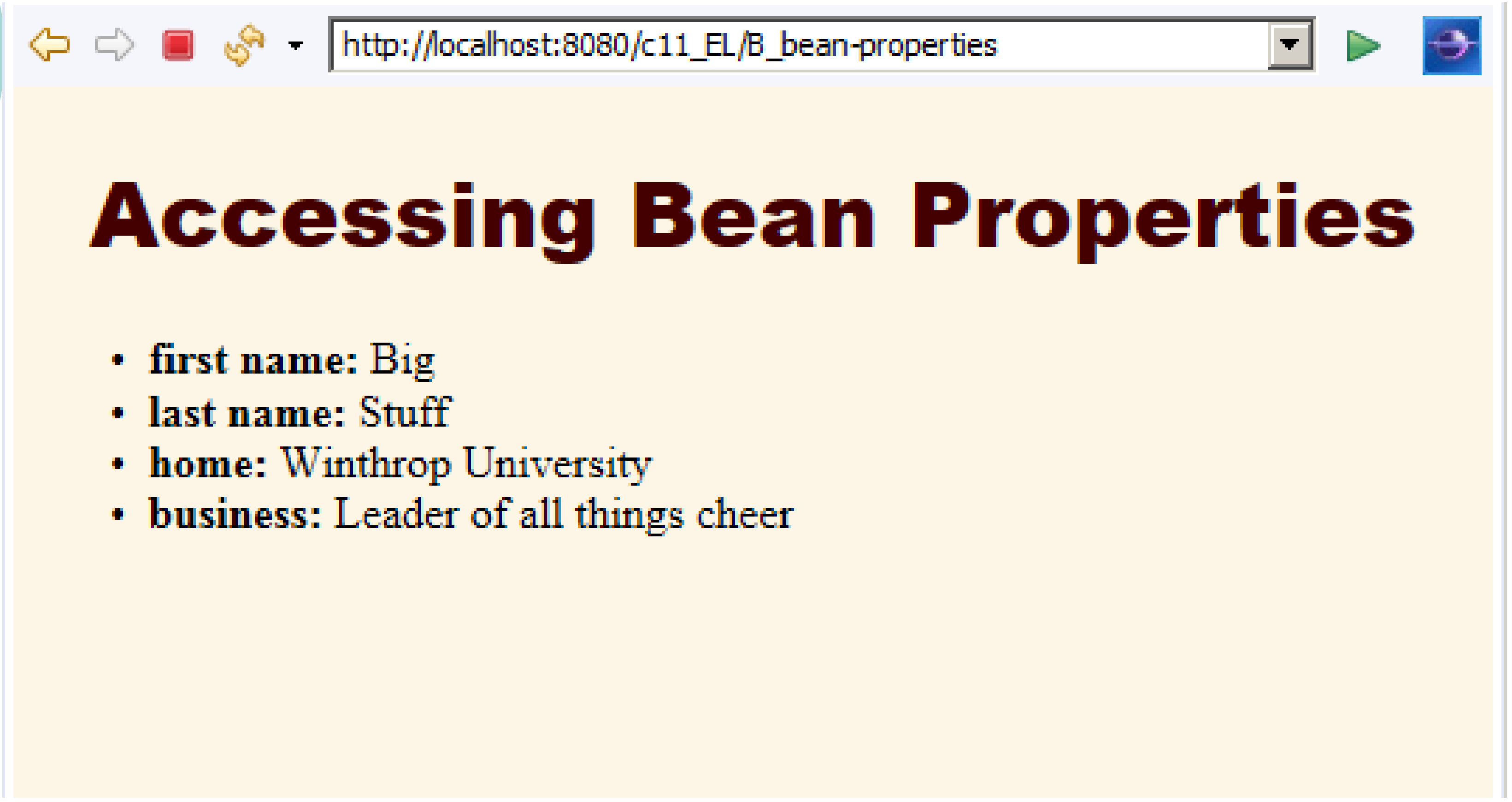
Example: Accessing Bean Properties

```
@WebServlet("/B_bean-properties")
public class B_BeanProperties extends HttpServlet {
     private static final long serialVersionUID = 1L;

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

    Name name = new beansRobjects.Name();
    name.setFirstName("Big");
   name.setLastName("Stuff");

    Company company = new Company();
    company.setCompanyName("Winthrop University");
    company.setBusiness("Leader of all things cheer");

    Employee employee = new Employee();
    employee.setName(name);
    employee.setCompany(company);

    request.setAttribute("employee", employee);// set an object

    RequestDispatcher dispatcher = request.getRequestDispatcher("/B_bean-properties.jsp");
    dispatcher.forward(request, response);

  }
}
```

Example: Accessing Bean Properties …

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Accessing Bean Properties</title>
    <link rel="stylesheet" href="./css/JSP-Styles.css"
type="text/css">
</head>
<body>
    <h1>Accessing Bean Properties</h1>
    <ul>
      <li><b>first name:</b> ${employee.name.firstName}
</li>
      <li><b>last name:</b> ${employee.name.lastName} </li>
      <li><b>home:</b> ${employee.company.companyName}</li>
      <li><b>business:</b> ${employee.company.business}</li>
    </ul>
</body>
</html>
```

Example: Accessing Bean Properties (Result)



http://localhost:8080/c11_EL/B_bean-properties

Accessing Bean Properties

- **first name:** Big
- **last name:** Stuff
- **home:** Winthrop University
- **business:** Leader of all things cheer

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}**

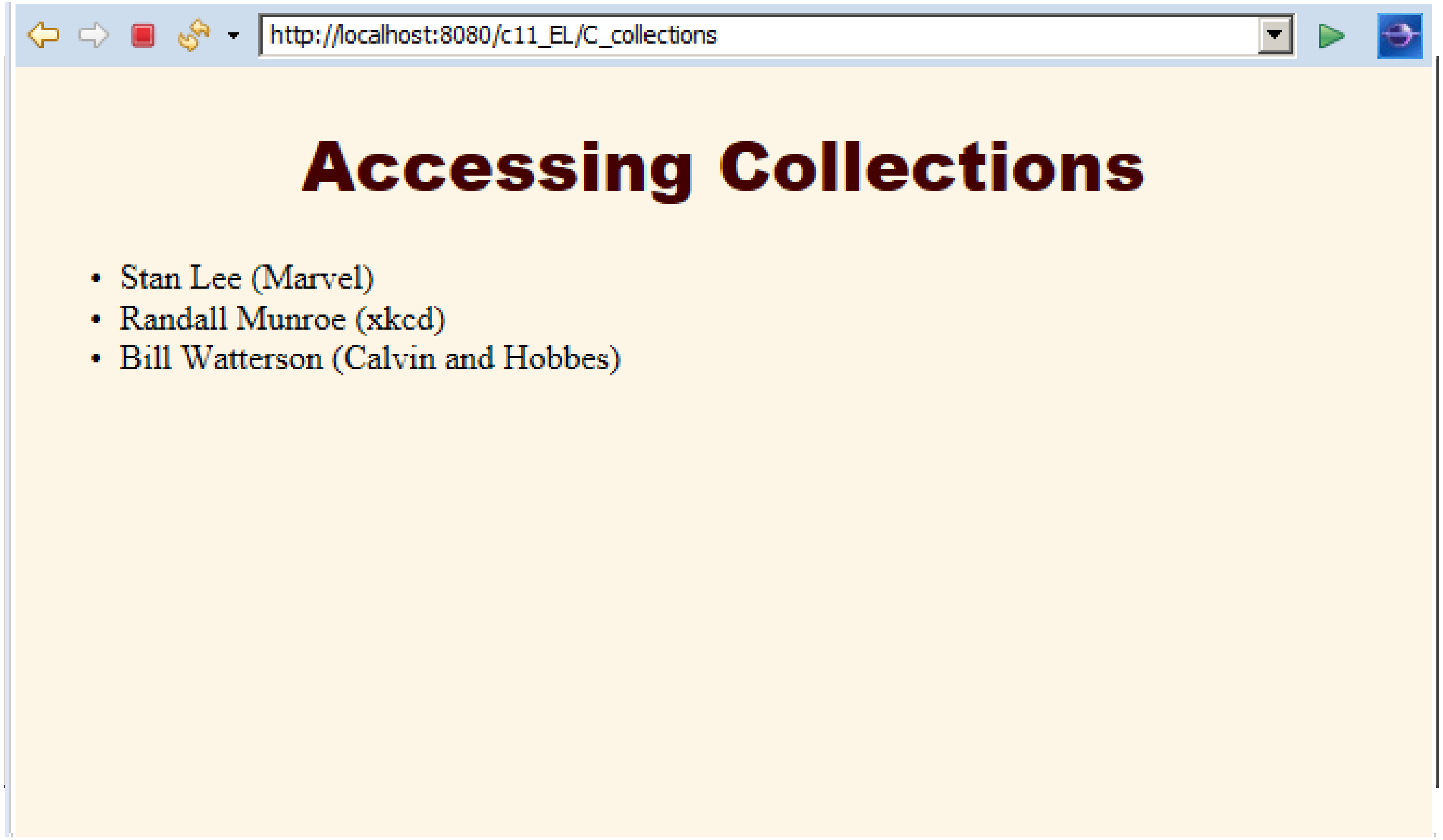
Example: Accessing Collections

```
public class Collections extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

        String[] firstNames = { "Stan", "Randall", "Bill" };     // array
        request.setAttribute("first", firstNames);

        ArrayListList<String> lastNames = new ArrayList<String>();        // List
                lastNames.add("Lee");
                lastNames.add("Munroe");
                lastNames.add("Watterson");
        request.setAttribute("last", lastNames);

    Map<String,String> companyNames =  new HashMap<String,String>();
                companyNames.put("Lee", "Marvel");
                companyNames.put("Munroe", "xkcd");
                companyNames.put("Watterson", "Calvin and Hobbes");
        request.setAttribute("company", companyNames);

    RequestDispatcher dispatcher = request.getRequestDispatcher("/C_collections.jsp");
    dispatcher.forward(request, response);
  }
}
```

Example: Accessing Collections …

```
<!DOCTYPE HTML>
<html>
<head>
     <title>Accessing Collections</title>
</head>
<body>
     <h1>Accessing Collections</h1>
     <ul>
       <li>${first[0]} ${last[0]} (${company["Lee"]})</li>
       <li>${first[1]} ${last[1]} (${company["Munroe"]})</li>
       <li>${first[2]} ${last[2]} (${company["Watterson"]})</li>
     </ul>
</body>
```

Example: Accessing Collections (Result)



Accessing Collections

- Stan Lee (Marvel)
- Randall Munroe (xkcd)
- Bill Watterson (Calvin and Hobbes)

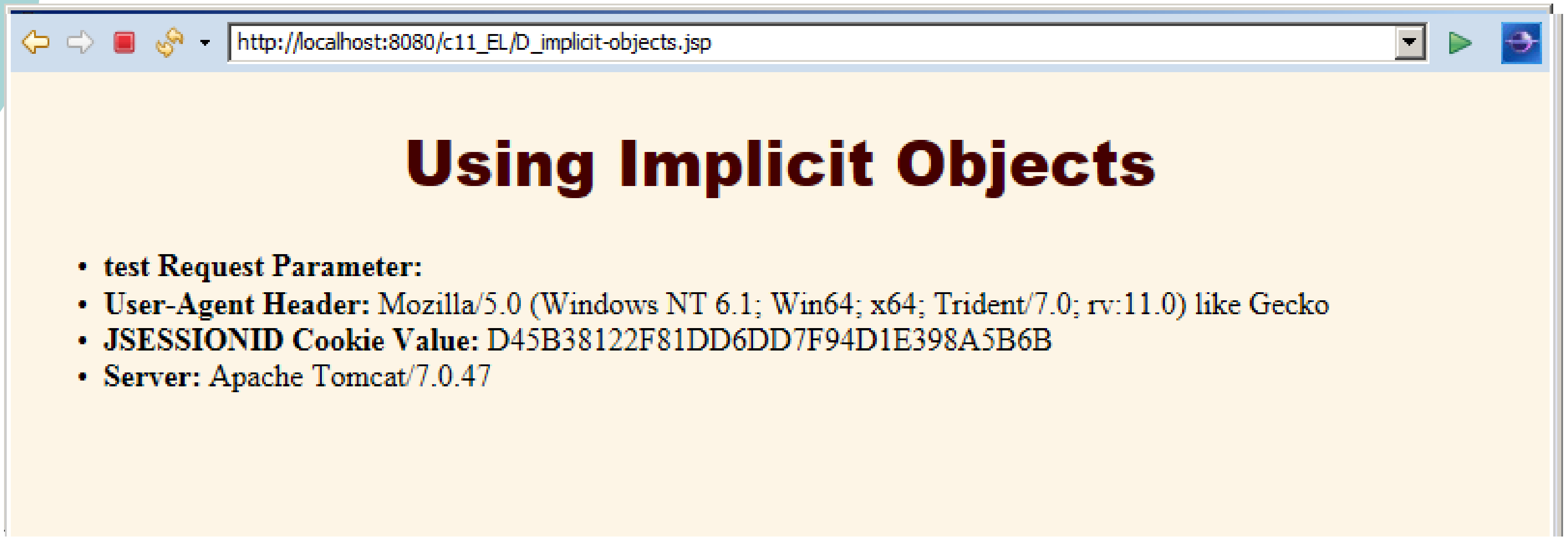
Referencing Implicit Objects (Predefined Variable Names)

- pageContext - The PageContext object.
  - E.g. ${pageContext.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}

Example: Implicit Objects

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Accessing Collections</title>
  <link rel="stylesheet" href="./css/JSP-Styles.css" type="text/css">
</head>
<body>
  <h1>Using Implicit Objects</h1>
  <ul>
    <li><b>test Request Parameter:</b> ${param.test}</li>
    <li><b>User-Agent Header:</b> ${header["User-Agent"]}</li>
    <li><b>JSESSIONID Cookie Value:</b> ${cookie.JSESSIONID.value}</li>
    <li><b>Server:</b> ${pageContext.servletContext.serverInfo}</li>
  </ul>
</body>
</html>
```

Example: Implicit Objects (Result)


http://localhost:8080/c11_EL/D_implicit-objects.jsp

Using Implicit Objects

- **test Request Parameter:**
- **User-Agent Header:** Mozilla/5.0 (Windows NT 6.1; Win64; x64; Trident/7.0; rv:11.0) like Gecko
- **JSESSIONID Cookie Value:** D45B38122F81DD6DD7F94D1E398A5B6B
- **Server:** Apache Tomcat/7.0.47

Expression Language Operators

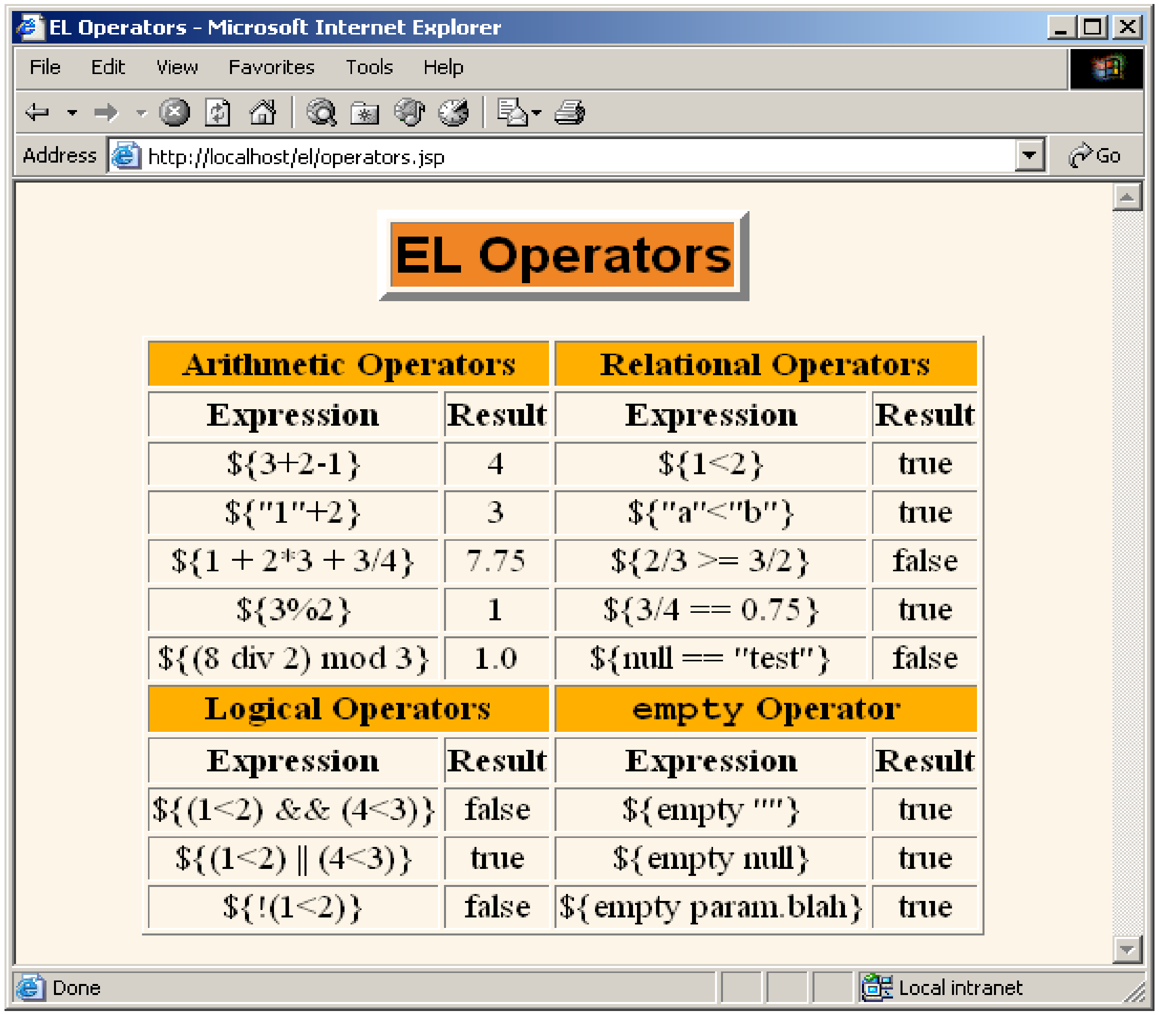
- Arithmetic
  - + - * / div % mod
- Relational
  - == eq != ne < lt > gt <= le >= ge
- Logical
  - && and || or ! Not
- Empty
  - True for null, empty string, empty array, empty list, empty map. False otherwise.
- CAUTION
  - Use extremely sparingly to preserve MVC model

Example: Operators

```
<tr>
    <td>&#36;&#123;3+2-1&#125;</td>
    <td>${3+2-1}</td>  <%-- addition/subtraction --%>
    <td>&#36;&#123;1&lt;2&#125;</td>
    <td>${1<2}</td>      <%-- numerical comparison --%>
</tr>
<tr>
    <td>&#36;&#123;"1"+2&#125;</td>
    <td>${"1"+2}</td>     <%-- string conversion --%>
    <td>&#36;&#123;"a"&lt;"b"&#125;</td>
    <td>${"a"<"b"}</td> <%-- lexical comparison --%>
</tr>
<tr>
    <td>&#36;&#123;1 + 2*3 + 3/4&#125;</td>
    <td>${1 + 2*3 + 3/4}</td>  <%-- mult/div --%>
    <td>&#36;&#123;2/3 &gt;= 3/2&#125;</td>
    <td>${2/3 >= 3/2}</td>        <%-- >= --%>
</tr>
<tr>
    <td>&#36;&#123;3%2&#125;</td>
    <td>${3%2}  </td>                <%-- modulo --%>
    <td>&#36;&#123;3/4 == 0.75&#125;</td>
    <td>${3/4 == 0.75}</td> <%-- numeric = --%>
</tr>

<tr>
<%-- div and mod are alternatives to / and % --%>
    <td>&#36;&#123;(8 div 2) mod 3&#125;</td>
    <td>${(8 div 2) mod 3}</td>
<%-- compares with "equals" but returns false for null --%>
    <td>&#36;&#123;null &#61;&#61; &quot;test&quot;&#125;</td>
    <td>${null == "test"}</td>
</tr>
```

```
<tr>
    <th class="colored" colspan="2">logical operators </th>
    <th class="colored" colspan="2">empty operator</th>
</tr>
<tr>
    <th>expression</th>
    <th>result</th>
    <th>expression</th>
    <th>result</th>
</tr>
<tr>
    <td>&#36;&#123;(1&lt;2) &amp;&amp; (4&lt;3)&#125;</td>
    <td>${(1<2) && (4<3)}</td> <%--and--%>
    <td>&#36;&#123;empty &quot;&quot;&#125;</td>
    <td>${empty ""}</td> <%-- empty string --%>
</tr>
<tr>
    <td>&#36;&#123;(1&lt;2) || (4&lt;3)&#125;</td>
    <td>${(1<2) || (4<3)}</td> <%--or--%>
    <td>&#36;&#123;empty null&#125;</td>
    <td>${empty null}</td> <%-- null --%>
</tr>
<tr>
    <td>&#36;&#123;!(1&lt;2)&#125;</td>
    <td>${!(1<2)}</td>  <%-- not -%>
    <%-- handles null or empty string in request param --%>
    <td>&#36;&#123;empty param.blah&#125;</td>
    <td>${empty param.blah}</td>
</tr>
```

Example: Operators (Result)


EL Operators - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/el/operators.jsp
Go
Done
Local intranet

EL Operators

Arithmetic Operators		Relational Operators	
Expression	Result	Expression	Result
${3+2-1}	4	${1<2}	true
${"1"+2}	3	${"a"<"b"}	true
${1 + 2*3 + 3/4}	7.75	${2/3 >= 3/2}	false
${3%2}	1	${3/4 == 0.75}	true
${(8 div 2) mod 3}	1.0	${null == "test"}	false
Logical Operators		**empty Operator**	
Expression	Result	Expression	Result
${(1<2) && (4<3)}	false	${empty ""}	true
${(1<2) || (4<3)}	true	${empty null}	true
${!(1<2)}	false	${empty param.blah}	true

Summary

- The JSP 2.0 EL provides concise, easy-to-read access to
  - Bean properties
  - Collection elements
  - Standard HTTP elements such as request parameters, request headers, and cookies
- The JSP 2.0 EL works best with MVC
  - Use only to output values created by separate Java code
- Resist use of EL for business logic