ITMD 415/515
Advanced Software Development

Week 3 – Web Applications, Servlet and JSP

Scott Spyrison

Administrative Stuff

- Remember – Check Canvas for Reading specifics
- GitHub Discussions

Servlet

- Let - suffix. Diminutive suffix. Smaller.
- Little Server, Part of a Server, Server-like
- Servlets: Smaller - yes. Lesser - no!

Figure 7–2 Java Web Application Technologies

HttpServlet

- Read explicit data sent by client (such as form data, request parameters)
- Read implicit data sent by client (such as request headers)
- Invoke other services and generate results
- Send explicit data back to client (HTML, etc)
- Send the implicit data to client (such as status codes and response headers)

A Typical HTTP Request

```
GET /search-servlet?keywords=servlets+jsp HTTP/1.1
Accept: image/gif, image/jpg, */*
Accept-Encoding: gzip
Connection: Keep-Alive
Cookie: userID=id456578
Host: www.somebookstore.com
Referer: http://www.somebookstore.com/findbooks.html
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0;
   Windows NT 5.0)
```

HTTP Request/Response

Request

```
GET /servlet/SomeName
   HTTP/1.1
Host: ...
Header2: ...
...
HeaderN:
   (Blank Line)
```

Response

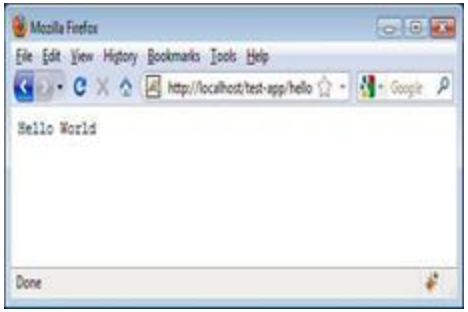
```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
   (Blank Line)
<!DOCTYPE ...>
<HTML>
<HEAD>...</HEAD>
<BODY>
...
</BODY></HTML>
```

A Servlet That Generates Plain Text

```
package testPackage; // Always use packages.

import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebServlet("/hello")
public class HelloWorld extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("Hello World");
  }
}
```



Interpreting HelloWorld Servlet

- @WebServlet("/address")
  - This is the URL relative to your context path
- doGet
  - Code for an HTTP GET request. doPost also common.
- HttpServletRequest
  - Contains anything that comes from the browser
- HttpServletResponse
  - Used to send stuff to the browser. Most common is getWriter for a PrintWriter that points at browser.
- @Override
  - General best practice when overriding inherited methods

IDE Generated Servlet and JSP

- NetBeans **Scaffolding**
- processRequest() method
- Time-saver!
  - Scaffolding: Trust, but Verify
  - Note - try-with-resources

POJO (JavaBean) Conventions and EL

- Java Classes
  - Non-public fields:
    - **private String myVariable**
  - Non-private accessors:
    - **public String getMyVariable()**
  - Non-private mutators:
    - **public void setMyVariable(String myVar)**
- EL (given an instance of MyClass named myClass)
  - **${myClass.myVariable}**

POJO (JavaBean) Conventions and EL

- **What matters is the method name, not the variable name**
- Here is the usual rule to turn a method into property:
  - Drop the word “get” or “set” and change the next letter to lowercase
    - Method name: getFirstName
    - Property name: firstName
    - Example: #{customer.firstName}
  - Exception 1: boolean properties
    - If getter returns boolean or Boolean
    - Method name: getPrime or isPrime
    - Property name: prime
    - Example: #{myNumber.prime}
  - Exception 2: consecutive uppercase letters
    - If two uppercase letters in a row after “get” or “set”
    - Method name: getURL
    - Property name: URL (not uRL)
    - Example: #{webSite.URL}

POJO (JavaBean) Conventions and EL

Method Names	Property Name	Example EL Usage
getFirstName setFirstName	firstName	${customer.firstName}
isExecutive setExecutive (boolean property)	executive	${customer.executive}
getExecutive setExecutive (boolean property)	executive	${customer.executive}
getZIP setZIP	ZIP	${address.ZIP}

Servlet as Controller - Redirect

- Stops processing of the request and sends HTTP status code for redirect.
- Browser URL becomes that which you redirect to
- **Can be any URL**
- Client (browser) initiates a **new request**

Servlet as Controller - Forward

- Passes the control of the request to another servlet or JSP
- **Includes** the request and response objects
- Client browser is unaware
- URL does not change in client
- URL is relative to web application context
- Forward vs Include
  - http://docs.oracle.com/javaee/7/tutorial/servlets007.htm#BNAGI

Summary – Servlet Basics

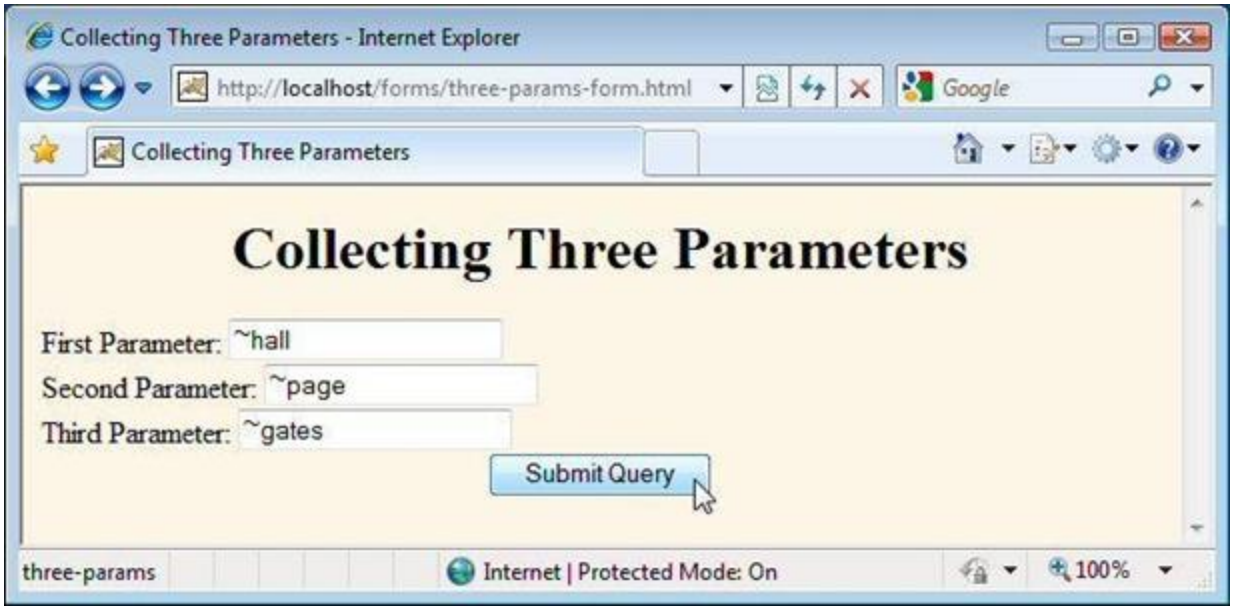
- Main servlet code goes in doGet or doPost:
  - The HttpServletRequest contains the incoming information
  - The HttpServletResponse lets you set outgoing information
- Give address with @WebServlet or web.xml
  - @WebServlet("/some-address")
  - public class SomeServlet extends HttpServlet { … }

Reading Form Data in Servlets

- request.getParameter("name")
  - Returns URL-decoded value of first occurrence of name in query string
  - Works identically for GET and POST requests
  - Returns null if no such parameter is in query data
- request.getParameterValues("name")
  - Returns an array of the URL-decoded values of all occurrences of name in query string (or null)
- request.getParameterNames() or request.getParameterMap()

Example HTML – Form Params

```
<FORM ACTION="three-params">
  First Parameter:  <INPUT TYPE="TEXT" NAME="param1"><BR>
  Second Parameter: <INPUT TYPE="TEXT" NAME="param2"><BR>
  Third Parameter:  <INPUT TYPE="TEXT" NAME="param3"><BR>
  <CENTER><INPUT TYPE="SUBMIT"></CENTER>
</FORM>
```



Example Servlet – Form Params

```
@WebServlet("/three-params")
public class ThreeParams extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>"+title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                "<UL>\n" +
                "  <LI><B>param1</B>: "
                + request.getParameter("param1") + "\n" +
                "  <LI><B>param2</B>: "
                + request.getParameter("param2") + "\n" +
                "  <LI><B>param3</B>: "
                + request.getParameter("param3") + "\n" +
                "</UL>\n" +
                "</BODY></HTML>");
  }
}
```

Server-Side Validation

- Check for missing
  - If Field missing in form, getParameter returns null
  - If Field blank when form is submitted, getParameter returns an empty string or possibly a String with whitespace depending on browser
  - Must check for null before empty string!
- Check for malformed
  - Value is present but in the wrong format

Web Validation

- HTML5 Validation
  - https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms_in_HTML
- Client-side vs Server-side vs Both
- What do we do in the ”real world?”

Validation Best Practices

- Users do crazy things. Some are malicious.
- Client-side vs Server-side vs Both
- Assume user input will be bad
- Avoid showing stack traces to users. Stack traces belong in logs
  - Use default values or suggestions on forms
  - Re-display original form with submitted values and error messages. Previously entered values shouldn't be lost
  - Show standard error pages for users, not system internals
- Without Validation standards, it may be necessary to check for null values (for example):

```
if ((param == null) || (param.trim().equals(""))) {
  doSomethingForMissingValues(...);
} else {
  doSomethingWithParameter(param);
}
```

Good? Bad? Ugly?

Verify your Apple ID. - sspyrisc × | Apache Tomcat/8.5.11 - Error ×

Apple Inc. [US] https://id.apple.com/cgi-bin/verify.cgi?language=US-EN&key=[redacted]

Apps | Bookmarks | gremlins.js | RMS | NUIT | HR | FW | SAIT | SSL | SQL | JS | TODO | MAC

HTTP Status 500 - Prohibited operation Process.start()

type Exception report

message Prohibited operation Process.start()

description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
java.lang.SecurityException: Prohibited operation Process.start()
        com.apple.security.DefaultNoProcess.onBeforeStart(DefaultNoProcess.java:27)
        com.apple.security.transformers.ProcessBuilderVisitor.onBeforeStart(ProcessBuilderVisitor.java:32)
        java.lang.ProcessBuilder.start(ProcessBuilder.java:1029)
        java.lang.Runtime.exec(Runtime.java:620)
        org.apache.catalina.servlets.CGIServlet$CGIRunner.run(CGIServlet.java:1504)
        org.apache.catalina.servlets.CGIServlet.doGet(CGIServlet.java:519)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:622)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
        org.tuckey.web.filters.urlrewrite.RuleChain.handleRewrite(RuleChain.java:176)
        org.tuckey.web.filters.urlrewrite.RuleChain.doRules(RuleChain.java:145)
        org.tuckey.web.filters.urlrewrite.UrlRewriter.processRequest(UrlRewriter.java:92)
        org.tuckey.web.filters.urlrewrite.UrlRewriteFilter.doFilter(UrlRewriteFilter.java:394)
```

note The full stack trace of the root cause is available in the Apache Tomcat/8.5.11 logs.

Apache Tomcat/8.5.11

Good? Bad? Ugly?

https://accessforindividuals.hc

https://accessforindividuals.hcsc.net/benefits/j_security_check

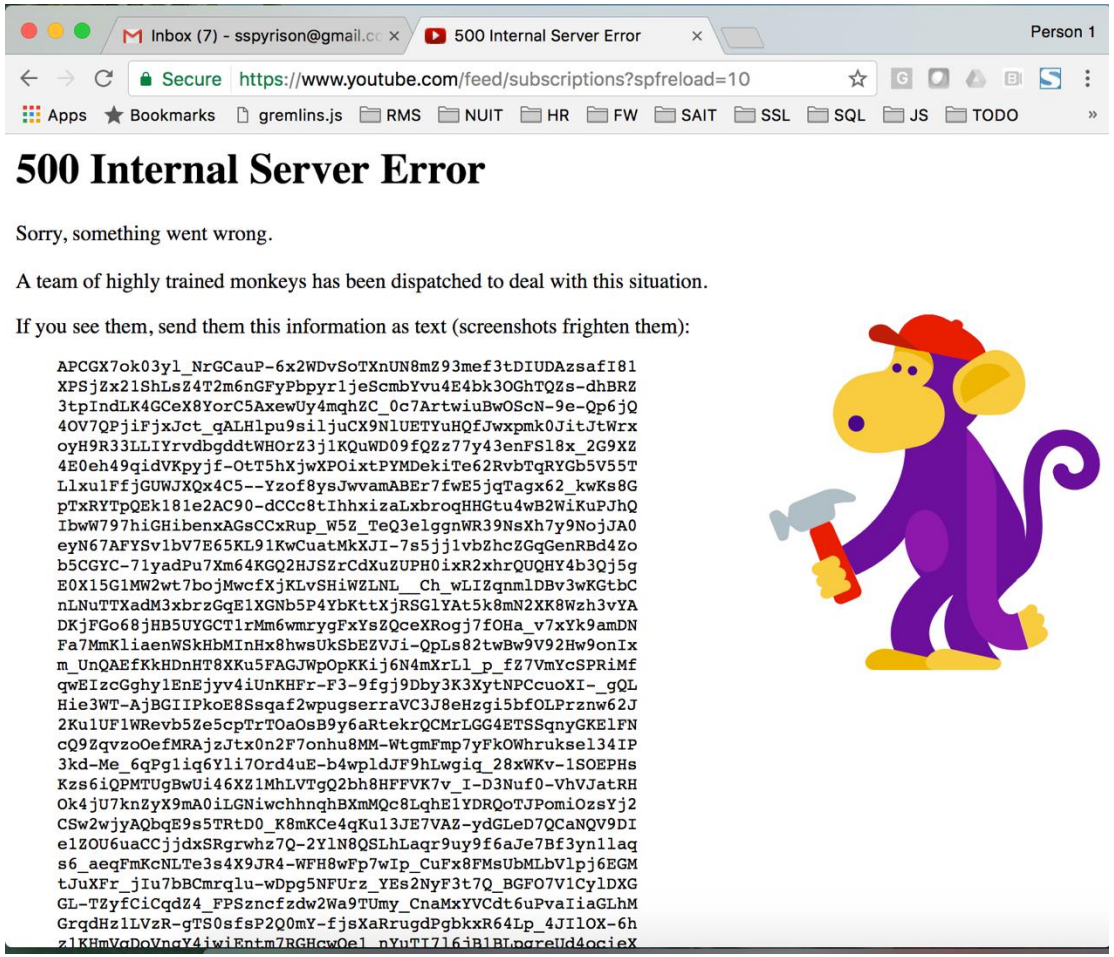
Bookmarks gremlins.js RMS NUIT HR FW SAIT SSL SQL JS TODO MAC SSO PHP HTTPD SYSADMIN »

SRVE0232E: Internal Server Error.
Exception Message: [com.dn.indiv.service.SystemUnavailableException: org.springframework.transaction.CannotCreateTransactionException: Could not open JPA EntityManager for transaction; nested exception is javax.persistence.PersistenceException: org.hibernate.exception.JDBCConnectionException: Cannot open connection]

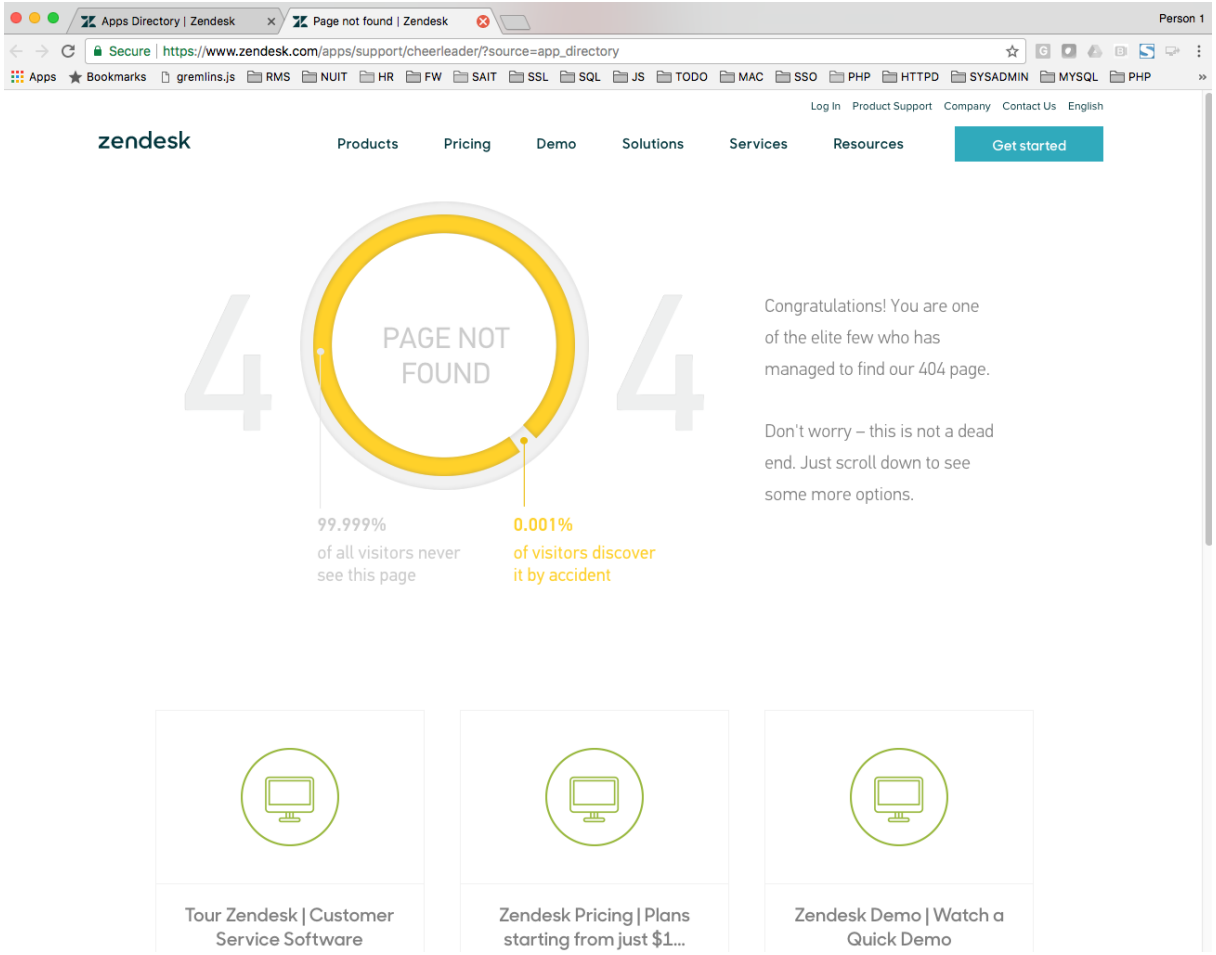
com.dn.indiv.service.SystemUnavailableException: org.springframework.transaction.CannotCreateTransactionException: Could not open JPA EntityManager for transaction; nested exception is javax.persistence.PersistenceException: org.hibernate.exception.JDBCConnectionException: Cannot open connection

IBM WebSphere Application Server

Good? Bad? Ugly?



Good? Bad? Ugly?



Summary – Forms and Servlets

- Make a form: <form …> … </form>
  - Relative URL for “action”. Textfields need “name”. Should always have submit button.
- Read data: request.getParameter("name")
  - Results in value as entered into form, not necessarily as sent over network. I.e., not URL-encoded.
- Check for missing or malformed data

Scopes

- Page Scope
  - Only available on the same page
- Request Scope
  - Interaction in a single request
- Session Scope
  - Interaction across multiple requests
- Application Scope
  - Shared state across all interactions within a web application

Session Handling

- Use session scope judiciously
- Use request scope liberally
- Session identifier travels across network, not the session itself
- Methods
  - request.getSession
  - session.getAttribute
  - session.setAttribute

Java Server Pages

- With servlets, it is easy to
  - Read form data
  - Read HTTP request headers
  - Set HTTP status codes and response headers
  - Use cookies and sessions
  - Share data among servlets
  - Remember data between requests
- But, it sure is a pain to
  - Use those println statements to generate HTML
  - Maintain HTML and collaborate with Web Designers

Java Server Pages

- Entire JSP page gets translated into a servlet (once), and servlet is what actually gets invoked (for each request)
- Ideas:
  - Use regular HTML for most of page
  - Mark servlet code with special tags
- Example:

```
<HTML>
<HEAD>
      <TITLE>Order Confirmation</TITLE>
      <LINK REL=STYLESHEET HREF="JSP-Styles.css” TYPE="text/css">
</HEAD>
<BODY>
      <H2>Order Confirmation</H2>
      Thanks for ordering
      <I><%= request.getParameter("title") %></I>!
</BODY></HTML>
```

Benefits of Java Server Pages

- Although JSP technically can't do anything servlets can't do, JSP makes it easier to:
  - Write HTML
  - Read and maintain the HTML
- JSP makes it possible to:
  - Use standard HTML tools
  - Have different members of your team do the HTML layout than do the Java programming
- JSP encourages you to Separate the (Java) code that creates the content from the (HTML) code that presents it

JSP Syntax (Old Style)

- Still Used:
  - @taglib
  - @include
  - @page
- Legacy – Don’t use on new code:
  - JSP Comment <%-- Comment --%>
  - JSP Expressions <%= expression %>
  - JSP Scriptlets <% code %>
  - JSP Declarations <%! code %>

JSTL and EL

- https://github.com/eclipse-ee4j/jsp-api
- https://github.com/eclipse-ee4j/jstl-api
- EL (Expression Language) can be accessed anywhere via ${expression}
- Familiar implicit objects

EL Implicit Objects

Implicit Object	Description
pageScope	Maps page-scoped variable names to their values
requestScope	Maps request-scoped variable names to their values
sessionScope	Maps session-scoped variable names to their values
applicationScope	Maps application-scoped variable names to their values
param	Maps a request parameter name to a single value
paramValues	Maps a request parameter name to an array of values
header	Maps a request header name to a single value
headerValues	Maps a request header name to an array of values
cookie	Maps a cookie name to a single cookie
pageContext	The context for the JSP page. Provides access to various objects: • servletContext: The context for the application's servlet and web components • session: The session object for the client • request: The request triggering the execution of the JSP page • response: The response returned by the JSP page

EL Operators

Operator	Description
.	Access a bean property or Map entry
[]	Access an array or List element, or Map entry
()	Group an expression
+	**Arithmetic:** Addition
-	**Arithmetic:** Subtraction or negation
*	**Arithmetic:** Multiplication
/ or div	**Arithmetic:** Division
% or mod	**Arithmetic:** Remainder
== or eq, != or ne	**Relational**: Equal and Not Equal
< or lt, <= or le	**Relational**: Less Than and Less Than or Equal To
> or gt, >= or ge	**Relational**: Greater Than and Greater Than or Equal
&& or and	**Logical** AND
|| or or	**Logical** OR
! or not	**Logical** NOT
empty	Prefix operation that can be used to determine whether a value is null or empty.
A ? B : C	Evaluate B or C, depending on the result of the evaluation of A

EL Operators (dot v brace)

- ${bean.map.myKey}
  - Resolves to bean.getMap().get(“myKey");
- ${bean.map[“myKey”]}
  - Resolves to bean.getMap().get(“myKey");
- ${bean.map[‘my.dotted.key’]}
  - Resolves to bean.getMap().get(“my.dotted.key");
- ${bean.map[bean2.someField]}
  - Resolves to bean.getMap().get(bean2.getSomeField());

EL Reserved Words

and	or	not	eq
ne	lt	gt	le
ge	true	false	null
instanceof	empty	div	mod

EL Expression Examples

EL Expression	Result
${!empty param.Add}	False if the request parameter named Add is null or an empty string.
${pageContext.request.contextPath}	The context path.
${sessionScope.cart.numberOfItems}	Value of the numberOfItems property of the session-scoped attribute named cart.
${param['mycom.productId']}	The value of the request parameter named mycom.productId.
${param.customerNumber}	
${header["host"]} or ${header.host}	The host header
${requestScope.customer}	The request scoped customer bean
${header["user-agent"]}	
${customer.customerNumber}	

JSTL Tag Libraries

- Used in combination with EL
- Replacements for older style JSP syntax
  - Core (Variable support, flow control, URL management)
  - Formatting (Formatting and I18N)
  - SQL (Database)
  - Functions (String manipulation, Collection length)
  - XML (XML parsing and transformation)
  - https://github.com/eclipse-ee4j/jstl-api
  - http://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/
  - http://docs.oracle.com/javaee/5/jstl/1.1/docs/api/

JSP, EL and JSTL Summary

- Be familiar with older JSP syntax due to legacy code in the marketplace
- For new JSP work, use EL and JSTL, but once we learn JSF you may prefer that
- Careful with SQL and XML tags
  - Useful for prototyping and rapid design, and perhaps for smaller, iterative applications or deadlines.

MVC

- Model (POJO/JavaBean)
- View (JSP/JSTL)
- Controller (Servlets)

Limit Code in Presentation Layer

- You have two options
  - Put 25 lines of Java code directly in the JSP page
  - Put those 25 lines in a separate Java class and put 1 line in the JSP page that invokes it
- Why is the second option much better?
  - Development. You write the separate class in a Java environment (editor or IDE), not an HTML environment
  - Debugging. If you have syntax errors, you see them immediately at compile time. Simple print statements can be seen.
  - Testing. You can write a test routine with a loop that does 10,000 tests and reapply it after each change.
  - Reuse. You can use the same class from multiple pages.

Why Combine Servlets and 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

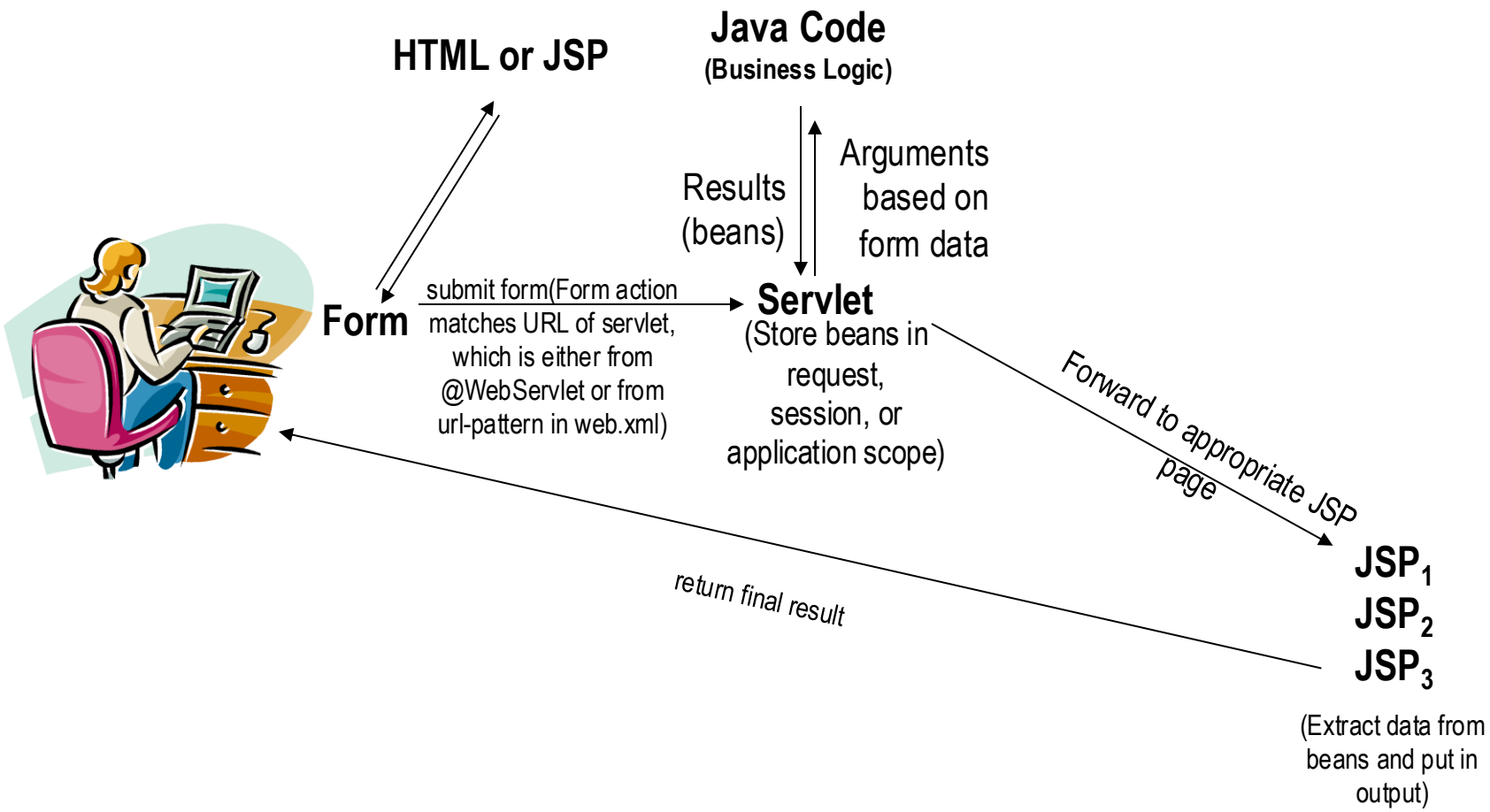
Possibilities for Handling Request

- Servlet only. Works well when:
  - Output is a binary type. E.g.: an image
  - There is no output. E.g.: you are doing forwarding or redirection as in Search Engine example.
  - Format/layout of page is highly variable. E.g.: portal.
- JSP only. Works well when:
  - Output is mostly character data. E.g.: HTML
  - Format/layout mostly fixed.
- Combination (MVC architecture). Needed when:
  - A single request will result in multiple substantially different-looking results.
  - You have a large development team with different team members doing the Web development and the business logic.
  - You perform complicated data processing, but have a relatively fixed layout.

MVC Misconceptions

- An elaborate framework is necessary
  - Frameworks are often useful
    - JSF (JavaServer Faces)
      - You should strongly consider JSF 2.x for medium/large projects!
    - Spring MVC
    - Struts
  - They are not required!
    - Implementing MVC with the builtin RequestDispatcher works very well for most simple and even moderately complex applications
- MVC totally changes your 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

MVC Flow of Control



Implement MVC with RequestDispatcher

- Define beans to represent result data
  - Ordinary Java classes with at least one getBlah method
- Use a servlet to handle requests
  - Servlet reads request parameters, checks for missing and malformed data, calls business logic, etc.
- Obtain bean instances
  - The servlet invokes business logic (application-specific code) or data-access code to obtain the results.

Implement MVC with RequestDispatcher

- 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.
- 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.

Implement MVC with RequestDispatcher (continued)

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

Error Pages

- Can be done in a standard way for Java web applications with web.xml
- Bad practice to leave application server default error pages – and can introduce security concerns by exposing too much information about your application logic

More Platform Basics

- JNDI
- JDBC Resource
- Dependency Injection

Platform Basics - JNDI and Resources

- JNDI as defined by Java EE Tutorial:
  - “The Java Naming and Directory Interface (JNDI) naming service enables components to locate other components and resources.”
- Resources, as defined by Java EE Tutorial:
  - “A program object that provides connections to other systems, such as database servers and messaging systems.”

Resources are identified by a unique and human friendly name called the JNDI name (i.e. jdbc/MyDataSource)

Platform Basics - JDBC Resource

- Connection Pool
- JDBC Resource
- Creation Options
  - NetBeans
  - glassfish-resources.xml
  - Glassfish Admin GUI
  - asadmin

Platform Basics - Injection

- Allows us to obtain references to resources without having to instantiate them directly.
- Declare the required resources via annotations (Injection Points)
- Container provides the required resources at runtime, and manages their lifecycle based on our specified scope
- Java EE Platform provides 2 types:
  - **Resource Injection**
  - **Dependency Injection**

Platform Basics - Resource Injection

- As defined in the Java EE Tutorial:
  - “Resource injection enables you to inject any resource available in the JNDI namespace into any container-managed object, such as a servlet, an enterprise bean, or a managed bean.”
- Commonly used for **DataSource and Validator**

Platform Basics - Resource Injection

Java EE 5 Way

```
try{
  InitialContext ctx = new InitialContext();
  DataSource ds =
  (DataSource)ctx.lookup("jdbc/myDataSrc");
} catch (NamingException ne) {}
```

Java EE 7+ Way

```
@Resource(lookup=”jdbc/myDataSource”)
DataSource myDs;
```

Sources Used

- The Jakarta EE Tutorial. Retrieved Aug 24, 2020, from https://eclipse-ee4j.github.io/jakartaee-tutorial/toc.html
- Juneau, J. (2020). Jakarta EE 8 Recipes. New York, NY: Apress.
- Goncalves, A. (2013). Beginning Java EE 7. New York, NY: Apress.
- Some slides adapted with permission from Marty Hall (www.coreservlets.com – JSP and Servlets)

Reference Slides

- Additional detail and information for those of you who would like to review it.

Servlet HTTP Request Headers

- Use request.getHeader for arbitrary header
  - Remember to check for null
  - Cookies, authorization info, content length, and content type have shortcut methods
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

Servlet HTTP Response Headers

- Setting response headers
  - In general, set with response.setHeader
  - In special cases, set with response.setContentType, response.setContentLength, response.addCookie, and response.sendRedirect
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

Servlet HTTP Response Codes

- Setting status codes
  - Redirect user with response.sendRedirect(someURL)
    - If you insert user-supplied data into the URL, encode with URLEncoder.encode
  - Send error pages with sendError
  - In general, set via response.setStatus
- Some sample status codes
  - 200 (default)
  - 302 (forwarding; set with sendRedirect)
  - 401 (unauthorized and needs authenticaton)
  - 403 (forbidden)
  - 404 (not found; set with sendError)
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Response_codes

Old Style JSP Expressions

- Format
  - <%= Java Expression %>
- Result
  - Expression evaluated, converted to String, and placed into HTML page at the place it occurred in JSP page
- Examples
  - Current time: <%= new java.util.Date() %>
  - Your hostname: <%= request.getRemoteHost() %>
- XML-compatible syntax
  - <jsp:expression>Java Expression</jsp:expression>
  - You cannot mix versions within a single page. You must use XML for entire page if you use jsp:expression.

Old Style JSP Scriptlets

- Format
  - <% Java Code %>
- Result
  - Code is inserted verbatim into generated servlet
- Example
  - <% if (Math.random() < 0.5) { %>
    Have a <B>nice</B> day!
    <% } else { %>
    Have a <B>lousy</B> day!
    <% } %>
- XML-compatible syntax
  - <jsp:scriptlet>Java Code</jsp:scriptlet>

Old Style JSP Declarations

- Format
  - <%! Java Code %>
- Result
  - Code is inserted verbatim into servlet's class definition, outside of any existing methods
- Examples
  - <%! private int someField = 5; %>
  - <%! private void someMethod(...) {...} %>
- Design consideration
  - Fields are clearly useful. For methods, it is usually better to define the method in a separate Java class.
- XML-compatible syntax
  - <jsp:declaration>Java Code</jsp:declaration>

Old Style JSP Import

- Format
  - <%@ page import="package.class" %>
  - <%@ page import="package.class1,...,package.classN" %>
- Purpose
  - Generate import statements at top of servlet definition
- Notes
  - Although JSP pages can be almost anywhere on server, classes used by JSP pages must be in normal servlet dirs
  - E.g.:
    …/WEB-INF/classes or
    …/WEB-INF/classes/directoryMatchingPackage
    - Always use packages for utilities that will be used by JSP!

Old Style JSP Include

- <jsp:include page="Relative URL" />
  - Output of URL inserted into JSP page at request time
  - Cannot contain JSP content that affects entire page
  - Changes to included file do not necessitate changes to pages that use it
- <%@ include file="Relative URL" %>
  - File gets inserted into JSP page prior to page translation
  - Thus, file can contain JSP content that affects entire page (e.g., import statements, declarations)
  - Changes to included file require you to manually update pages that use it

EL Expression Examples

EL Expression	Result
${1 > (4/2)}	false
${4.0 >= 3}	true
${100.0 == 100}	true
${(10*10) ne 100}	false
${'a' < 'b'}	true
${'hip' gt 'hit'}	false
${4 > 3}	true
${1.2E4 + 1.4}	12001.4
${3 div 4}	0.75
${10 mod 4}	2

JSTL Core

Tag	Description
<c:out>	Evaluates an expression and outputs the result.
<c:set>	Sets the value of an EL variable (and creates it if necessary) or the property of an EL variable in any of the JSP scopes.
<c:remove>	To remove an EL variable, you use the remove tag.
<c:if>	Allows the conditional execution of its body according to the value of the test attribute
<c:choose> <c:when> <c:otherwise>	Conditional execution similar to if, then, else
<c:forEach>	Basic iteration tag. Can be used with collections or basic loop.
<c:import> <c:param>	Access URL-based resources and include their content
<c:redirect>	Sends an HTTP redirect to the client
<c:url> <c:param>	Generate a URL with optional parameters

JSTL Formatting and I18N

Tag	Description
<fmt:formatDate>	Formats date/time
<fmt:parseDate>	Parse String representation of date/time
<fmt:formatNumber>	Formats number, currency or percent
<fmt:parseNumber>	Parse String representation of number, currency or percent
<fmt:bundle> <fmt:message>	Resource bundle tags for messages
<fmt:setLocale> <fmt:requestEncoding>	Setting Locale

JSTL SQL

Tag	Description
<sql:setDataSource>	e.g. <sql:setDataSource dataSource="jdbc/BookDB" />
<sql:query> <sql:param> <sql:dateParam>	Performs an SQL query that returns a result set
<sql:update> <sql:param> <sql:dateParam>	Used to insert or update a database row
<sql:transaction>	Used to perform a series of SQL statements atomically

JSTL Functions

Tag and Example	Description
fn:contains() fn:containsIgnoreCase()	Tests if a string contains a substring
fn:trim()	Trim whitespace from both ends of a string
fn:endsWith() fn:startsWith()	Tests if a string starts or ends with a prefix/suffix
fn:indexOf()	Returns the index of first substring occurence
fn:join() fn:split()	Joins elements of an array to a string, or splits a string into an array of substrings
fn:length()	String or collection length
fn:replace()	Replaces characters in a string
fn:substring() fn:substringAfter() fn:substringBefore()	Substring functions
fn:toLowerCase() fn:toUpperCase()	String case functions