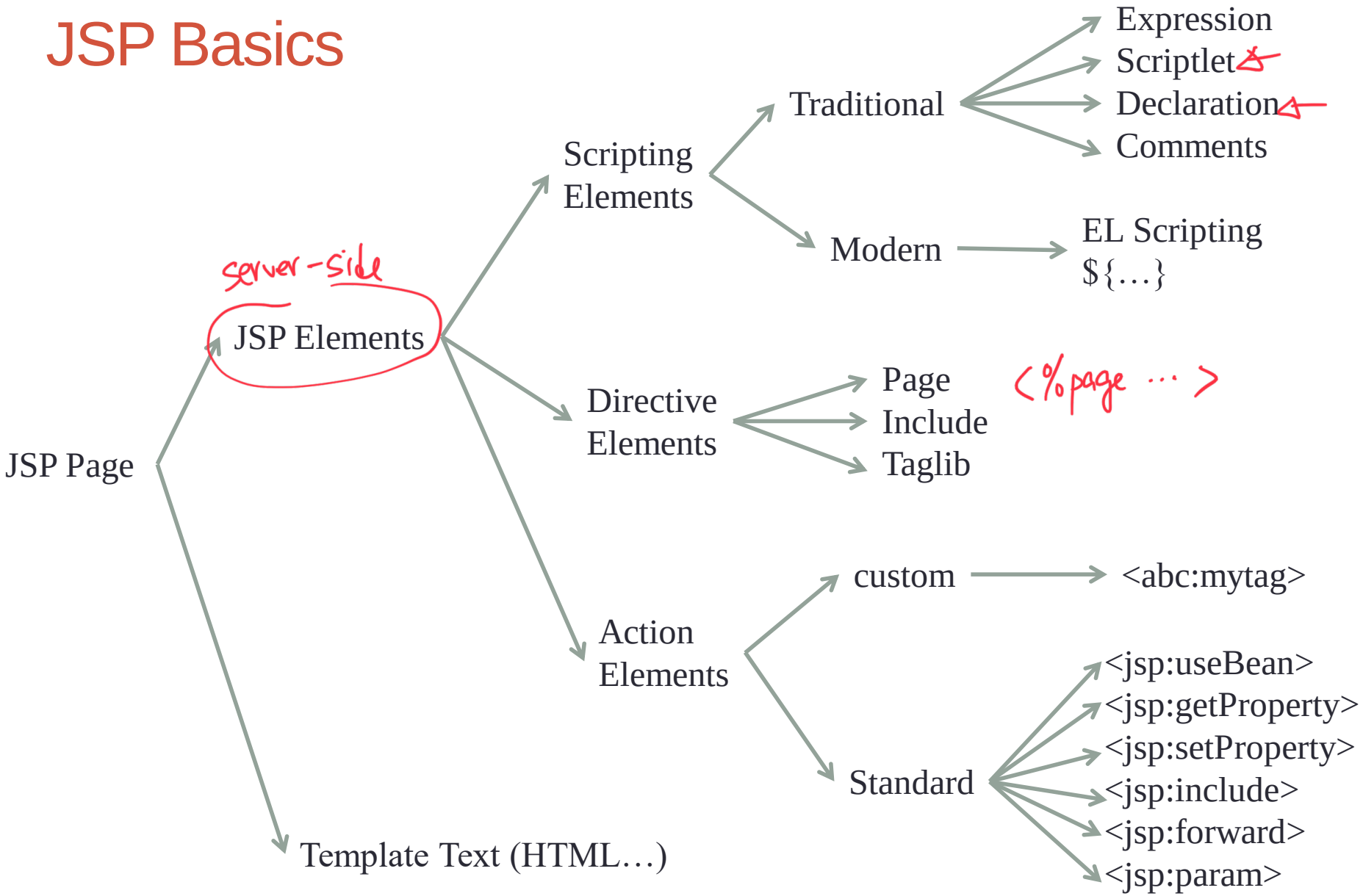
COMP S380F Lecture 3: JSP, JavaBean

Dr. Keith Lee
School of Science and Technology
Hong Kong Metropolitan University

JSP Basics


JSP Page
JSP Elements
server-side
Template Text (HTML…)
Scripting Elements
Traditional
Expression
Scriptlet
Declaration
Comments
Modern
EL Scripting
${…}
Directive Elements
Page
Include
Taglib
<%page … >
Action Elements
custom
<abc:mytag>
Standard
<jsp:useBean>
<jsp:getProperty>
<jsp:setProperty>
<jsp:include>
<jsp:forward>
<jsp:param>

Overview of this lecture

- JSP
- Lifecycle of a JSP page
- Directive elements:

  page, include, taglib
- Configuring JSP properties in web.xml
- Scripting elements:

  expression, scriptlet, declaration, comment
- JavaBean
- Action elements:

  include, forward, useBean, setProperty, getProperty

Deployment

JavaServer Pages (JSP)

- JavaServer Pages (JSP) is a web technology that allows you to easily create web content that has both static and dynamic components.

- JSP has all the dynamic capabilities of a Java Servlet, but provides a more natural approach for creating static content.
  - JSP lets the programmer separate HTML coding from business logic (Java coding).

Servlet

presentation

- JSP is similar to PHP, but it uses the Java programming language.

- To deploy and run a JSP page, a compatible web server with a servlet container is required.
  - E.g., Apache Tomcat

Main Features of JSP

1. A language for developing JSP pages, which are text-based documents that describe how to process a request and construct a response

2. An Expression Language (EL) for accessing server-side objects

3. Mechanisms for defining extensions to the JSP languages
   - JSP Standard Tag Library (JSTL)

We will cover Features 2 and 3 in Lecture 5.

JSP Page

A JSP page is a text document that contains two types of text:

- Static data:
  - which can be expressed in any text-based format (such as HTML, SVG, and XML)
- JSP elements:
  - Construct dynamic content.
  - The recommended file extension for the source file of a JSP page is .jsp
  - The recommended extension for the source file of a fragment of a JSP page is .jspf

HelloServlet.java (from Lecture 2)

```
package hkmu.comps380f;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Date;

public class HelloServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");
        PrintWriter out = response.getWriter();
        Date today = new Date();
        out.println("<!DOCTYPE html><html><body>");
        out.println("<h1>It is now: " + today + "</h1>");
        out.println("</body></html>");
    }
}
```

An Equivalent JSP Page: hello_page.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8" language="java" %>
<%@page import="java.util.*" %>

<!DOCTYPE html><html><body>
<h1>It is now: <%=new Date()%></h1>
</body></html>
```

The JSP page is much simpler than the Servlet class:

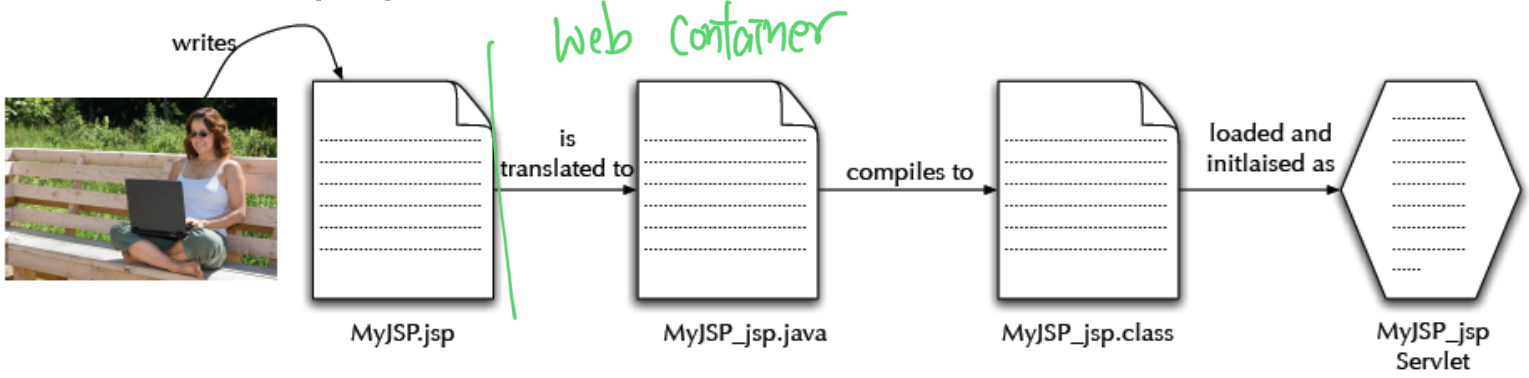
- No explicit import of the Servlet packages
- No declaration of the Servlet class HelloServlet
- No setContentType
- No PrintWriter

JSP Tag Examples

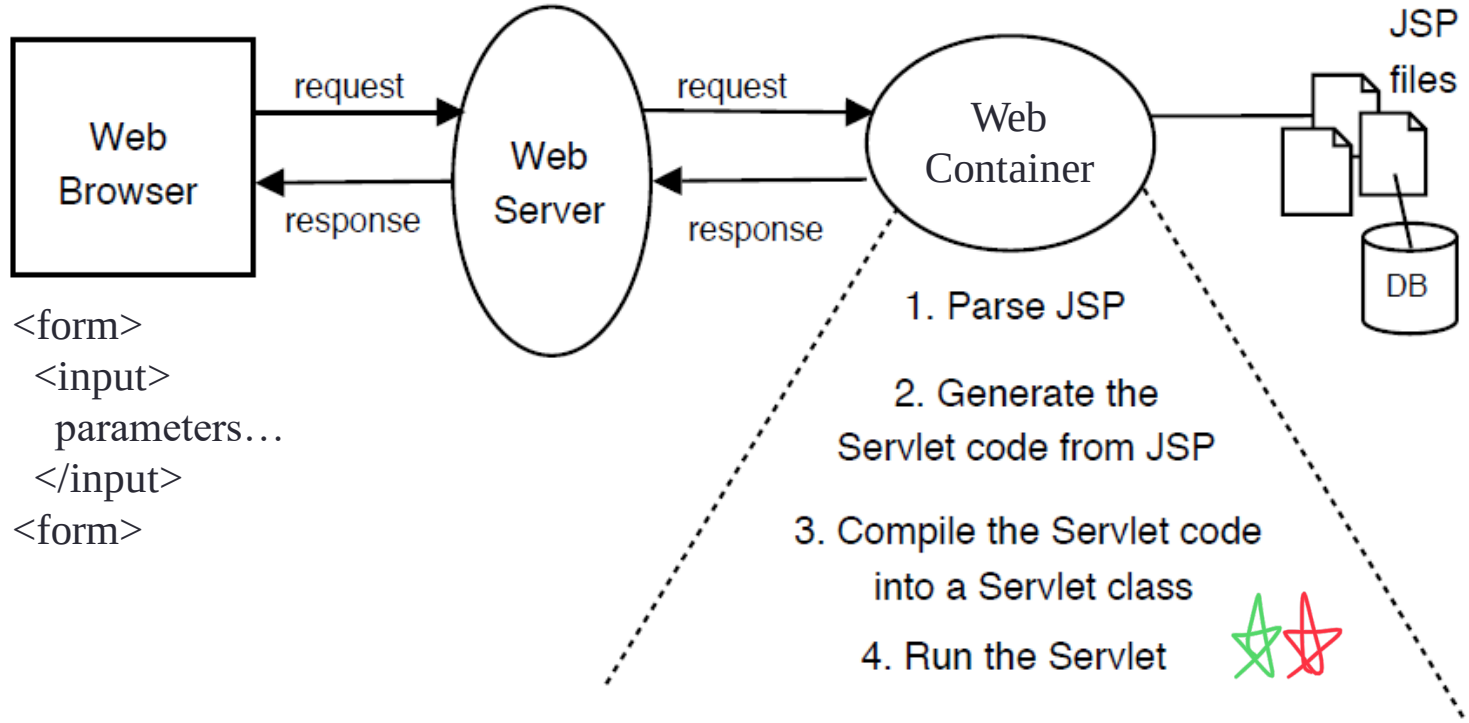
- <%= … %> is used for expressions.
  - <%= request.getParameter ("email") %>
- <%! … %> is used for declarations.
  - <%! String name, email; %>
- <% … %> is used for straight Java code.
  - <% if (x > 5) { … %>
- <%@ … %> is used to include another file such as an HTML file or a package such as java.util.*.
  - <%@ page contentType="text/html; charset=UTF-8" %>
  - <%@ include file="footer.html" %>
  - <%@ page import="java.util.*, mypackage.util" %>
  - <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

Lifecycle of a JSP Page

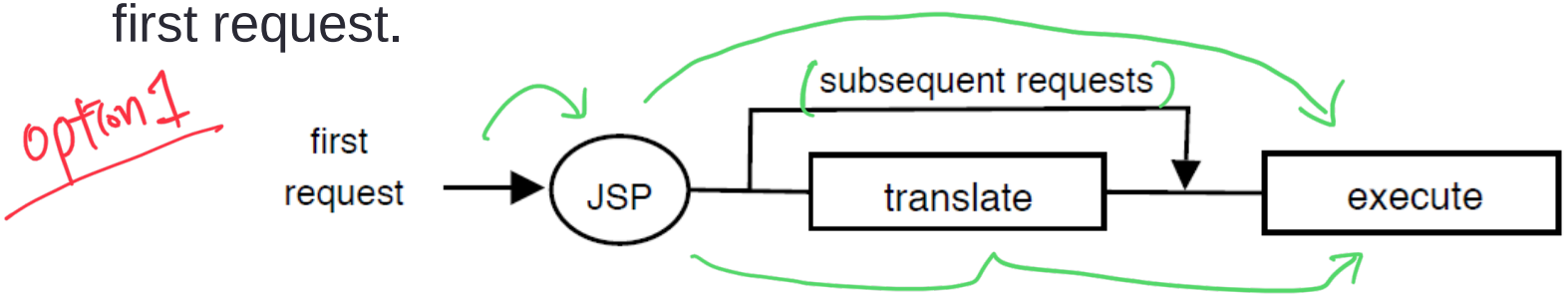
- The lifecycle of JSP is maintained by Web container.
- **JSP is just a servlet** (and it becomes one eventually).
- Web container converts a JSP to a Servlet **at run time**.
  1. Some web containers (e.g., Tomcat) translates and compiles the JSP when the first request to that JSP arrives.
     For future requests, the JSP is already compiled and ready to use.
  2. Many web containers also have the option of precompiling all JSP pages when deploying the web app.
     - Option 1 slows down runtime performance, while Option 2 slows down the deployment.



Lifecycle of a JSP Page (cont')



- Parsing, generating, compiling of JSP only happens on the first request.



Where do JSP pages go in the web app?

1. JSP pages can be placed **under the application root** with other static content.
   - E.g., /hello_page.jsp
     We can directly access it using the URL:
     <web app base URL>/hello_page.jsp

2. In web.xml, JSPs can also be mapped to virtual URLs just like Servlets and also with Servlet init parameters.

web.xml

```
<servlet>
  <servlet-name>helloJSP</servlet-name>
  <jsp-file>/hello_page.jsp</jsp-file>
  <init-param>
     <param-name>Course</param-name>
     <param-value>comps380f</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>helloJSP</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

Example's URL pattern: /Lecture03/hello

Where do JSP pages go in the web app? (cont')

- A user may access JSP pages in the application root directly from their browser. index.jsp → Base URL = server URL + Context Root
- You may want to prevent browsers to access JSP pages directly, such as JSPs that rely on *session and request attributes* provided by a forwarding Servlet.
- We can place JSP pages in the **WEB-INF folder**, which is protected from web access.

web.xml

```
<servlet>
  <servlet-name>helloJSP</servlet-name>
  <jsp-file>/WEB-INF/jsp/hello_page.jsp</jsp-file>
</servlet>
```

Recap: Request attributes and Request dispatching

- We use **request attribute** when we want some other component of the web application taking over all or part of your request.

```
// code in doGet()
String postcode = getPostcode(request.parameter("suburb"));
request.setAttribute("pc", postcode);

RequestDispatcher view =
            request.getRequestDispatcher("DisplayPostcode.jsp");
view.forward(request, response);
```

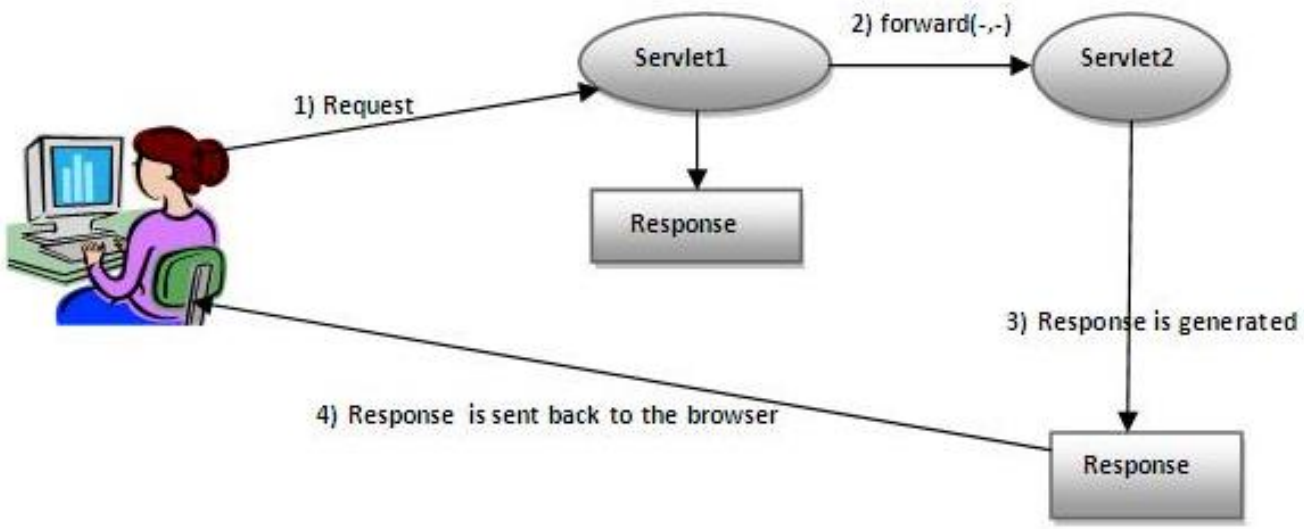
- Here, we use the RequestDispatcher interface to dispatch the request to another JSP page.
- The JSP will use the attribute “pc” in the request object to access the postcode.

Recap: RequestDispatcher in Servlet

- The RequestDispatcher interface provides the facility of dispatching the request to another resource, e.g., servlet, jsp, or html.
- This interface can also be used to include the content of another resource also.
- It is one of the way of servlet collaboration.

The RequestDispatcher interface provides two methods: **forward** and **include**

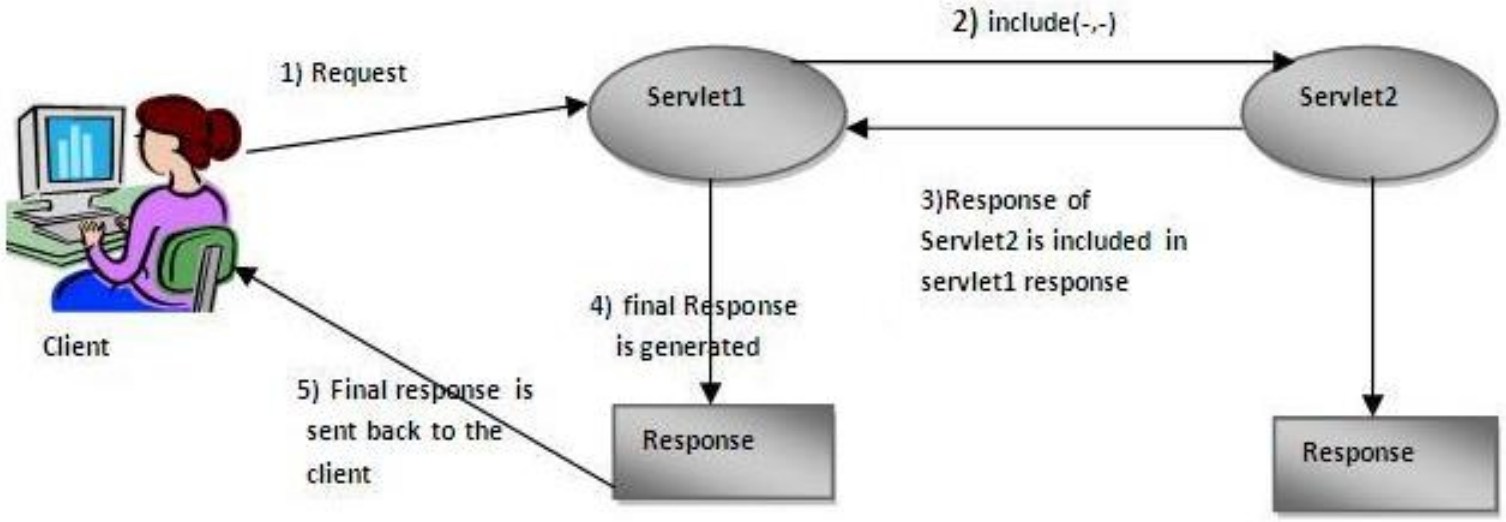
- **Forward:** Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.




```
public void forward(ServletRequest request,ServletResponse response)
```

Recap: RequestDispatcher in Servlet (cont')

- **Include:** Includes the content of a resource (servlet, JSP page, or HTML file) in the response.




```
public void include(ServletRequest request,ServletResponse response)
```

Attributes in a JSP page

- JSP provides some implicit objects, e.g., Defined inside JSP Service()
  - **`application`**: the ServletContext object
  - **`request`**: the HttpServletRequest object
  - **`session`**: the HttpSession object associated with the request
- Application scope (**context attribute**):
  - Servlet: `getServletContext().setAttribute("foo", barObj);`
  - JSP: `<% application.setAttribute("foo", barObj); %>`
- Request scope (**request attribute**):
  - Servlet: `request.setAttribute("foo", barObj);`
  - JSP: `<% request.setAttribute("foo", barObj); %>`
- Session scope (**session attribute**):
  - Servlet: `request.getSession().setAttribute("foo", barObj);`
  - JSP: `<% session.setAttribute("foo", barObj); %>`

Separating Business logic and Presentation

- A more natural practice in web application development is to separate the business logic and the presentation.
  - Business logic: Servlets
  - Presentation: JSP pages

- Here is another example of hello_page.jsp:

web.xml

```
<servlet>
  <servlet-name>helloServlet</servlet-name>
  <servlet-class>hkmu.comps380f.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>helloServlet</servlet-name>
  <url-pattern>/greeting</url-pattern>
</servlet-mapping>
```

Example's URL pattern: /Lecture03/greeting

Collaboration of Servlet and JSP page

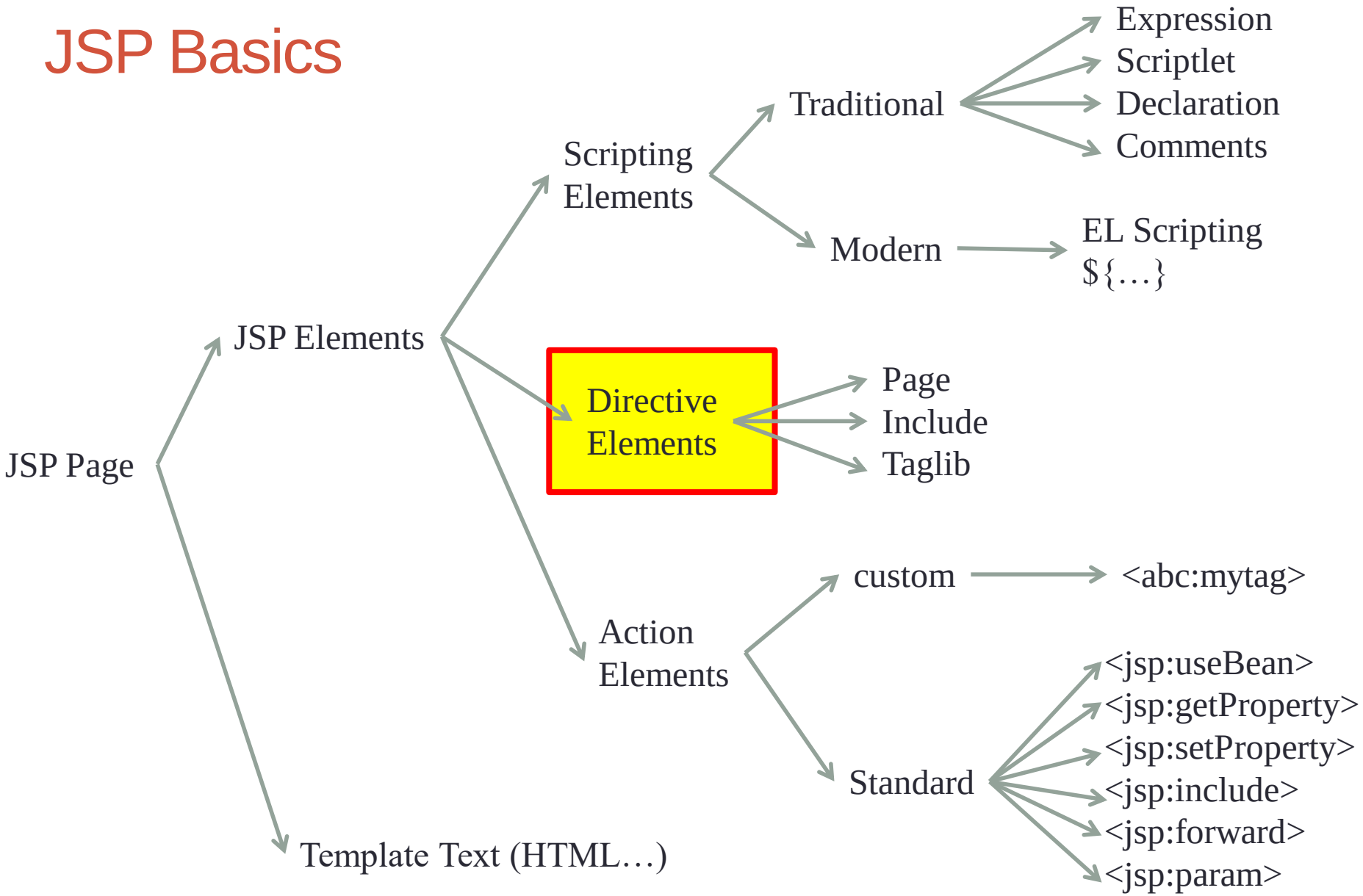
HelloServlet.java

```
package hkmu.comps380f;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Date;

public class HelloServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        Date today = new Date();
        request.setAttribute("today", today);

        RequestDispatcher view =
                request.getRequestDispatcher("/WEB-INF/jsp/hello_page.jsp");
        view.forward(request, response);
    }
}
```

hello_page.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8" language="java" %>
<!DOCTYPE html><html><body>
<h1>It is now: <%= request.getAttribute("today")%></h1>
</body></html>
```

JSP Basics


JSP Page
JSP Elements
Scripting Elements
Traditional
Expression
Scriptlet
Declaration
Comments
Modern
EL Scripting
${…}
Directive Elements
Page
Include
Taglib
Action Elements
custom
<abc:mytag>
Standard
<jsp:useBean>
<jsp:getProperty>
<jsp:setProperty>
<jsp:include>
<jsp:forward>
<jsp:param>
Template Text (HTML…)

JSP Directives

JSP directives gives special information about the page to the Web container.

- Syntax: <%@ directive ... %>

- **page** directive: processing information for the page
  - <%@ page contentType="text/html; charset=UTF-8" %>
    This sets the content type returned by the page.
  - <%@ page import="java.util.*, mypackage.util" %>
  - <%@ page errorPage="/errors/GenericError.jsp" %>
  - <%@ page info="a string for servlet's getServletInfo()" %>
  - <%@ page session="false" %>

Prevent user view error from web container

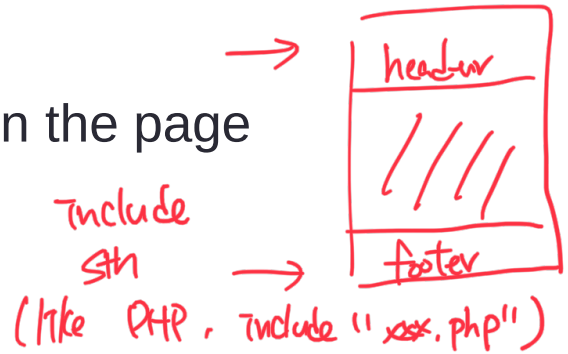
<%= this.getServletInfo() %>

tag attrb => using " "

JSP Directives (cont')

- **include** directive: files to be included in the page

```
<%@ include file="footer.html" %>
```



  - This lets you insert a piece of text (JSP or HTML code) into the main page before the main page is translated into a Servlet.

- **taglib** directive: tag library to be used in the page, e.g., JSTL

```
<%@ taglib uri="uri-for-the-library" prefix="c" %>
```

  - This imports custom tag libraries.
  - JSTL extends the JSP specification by adding a tag library of JSP tags for common tasks, such as conditional execution, loops, and database access. (To be covered in Lecture 5)

Configuring JSP properties in web.xml

- We may have many JSPs with similar properties.
- It is cumbersome to place the page directive at the top of every JSP file.
- We can configure common JSP properties in web.xml.
- Suppose we want to include the following JSP fragment to every JSP pages in the directory /WEB-INF/jsp/admin/.

/WEB-INF/jsp/base.jspf

```
<%@ page import="package.SomeServlet, package.AnotherClass" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

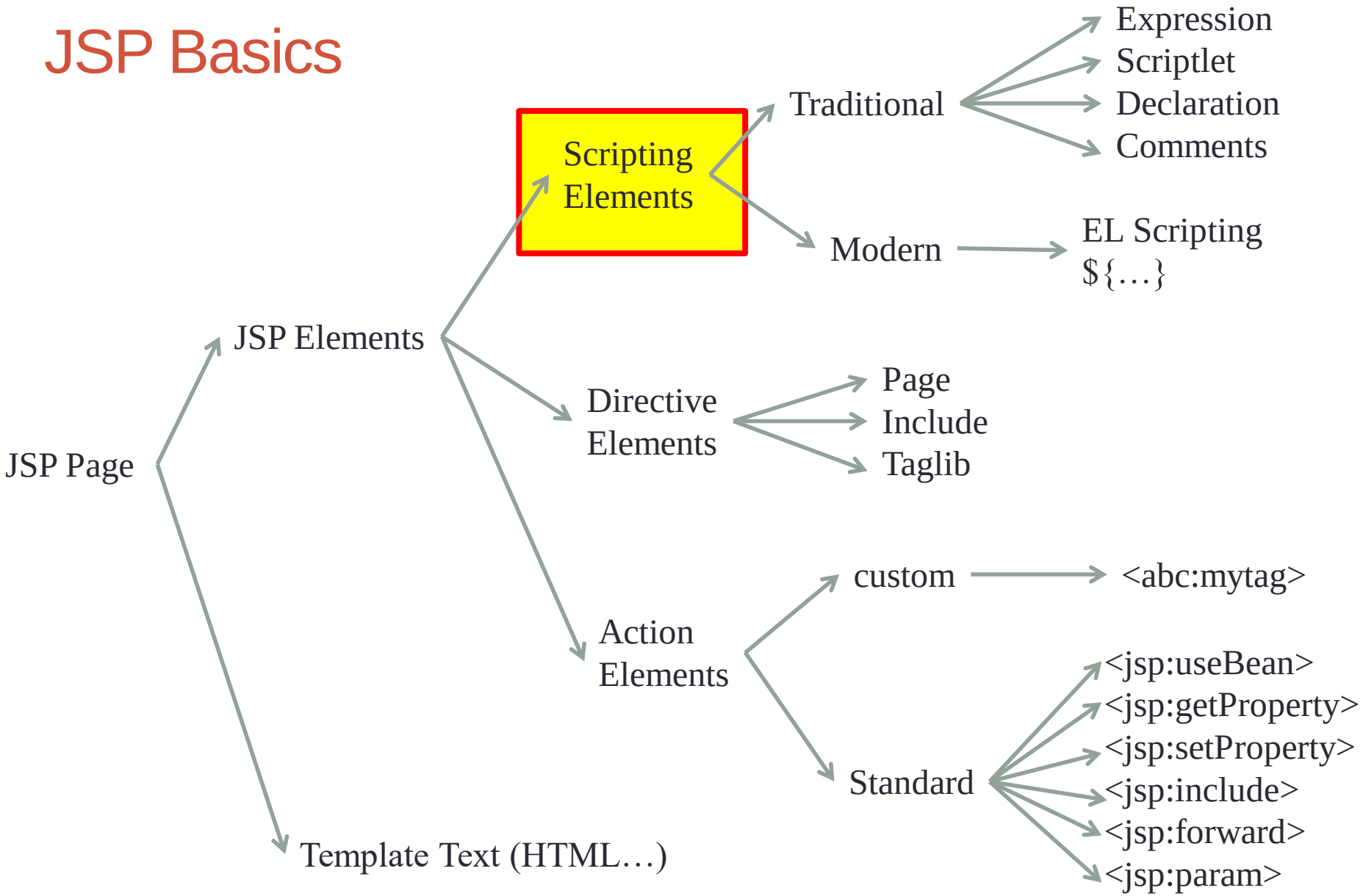
web.xml

```
<jsp-config>
    <jsp-property-group>
        <url-pattern>/WEB-INF/jsp/admin/*</url-pattern>
        <page-encoding>UTF-8</page-encoding>
        <include-prelude>/WEB-INF/jsp/base.jspf</include-prelude>
        <default-content-type>text/html</default-content-type>
    </jsp-property-group>
</jsp-config>
```

jsp config

(apply to all page match this)

JSP Basics


JSP Page
JSP Elements
Scripting Elements
Traditional
Expression
Scriptlet
Declaration
Comments
Modern
EL Scripting
${…}
Directive Elements
Page
Include
Taglib
Action Elements
custom
<abc:mytag>
Standard
<jsp:useBean>
<jsp:getProperty>
<jsp:setProperty>
<jsp:include>
<jsp:forward>
<jsp:param>
Template Text (HTML…)

JSP Scripting: Expression

JSP expression is evaluated and converted into ***a string,*** and is inserted in the page (into the translated Servlet, inside out.println(…)).

- Syntax: <%= expression %>

- An expression must result in a String. E.g.,
  - <h1>Hello <%= username %></h1>
  - Discounted price is: <%= (total * 0.9) %>

- The evaluation is performed **at run time**.
  - E.g., Current time: <%= new java.util.Date() %>
  - The above will display the time when the page is requested.

JSP Elements: Using the implicit objects

- **request:** the HttpServletRequest object
- **response:** the HttpServletResponse object
- **session:** the HttpSession object associated with the request
- **out:** the PrintWriter object
- **config:** the ServletConfig object
- **application:** the ServletContext object

Example:

```
<html><body>
   <h1>JSP expressions</h1>
  <ul>
     <li>Current time is: <%= new java.util.Date() %>
     <li>Server Info: <%= application.getServerInfo() %>
     <li>Servlet Init Info: <%= config.getInitParameter("WebMaster") %>
     <li>This Session ID: <%= session.getId() %>
     <li>The value of <code>TestParam</code> is:
     <%= request.getParameter("TestParam") %>
  </ul>
</body></html>
```

Example's URL pattern: /Lecture03/expr

JSP Scripting: Scriptlet

JSP scriptlets are inserted verbatim into the translated servlet code.

- The scriptlet can contain any number of language statements, variable or method declarations, or expressions that are valid in the page scripting language.

- Within a scriptlet, you can do any of the following:
  - Declare variables or methods to use later in the JSP page.
  - Write expressions valid in the page scripting language.
  - Use any of the implicit objects.
  - Write any other statement valid in the scripting language used in the JSP page.

- Syntax: <% ... Java code ... %>

Remember that JSP expressions contain 'String values', but JSP scriptlets contain 'Java statements'.

JSP Scripting: Scriptlet - Example 1

```
<!DOCTYPE html>
<html>
<body>
  <%
     // This scriptlet declares and initializes "date"
     java.util.Date date = new java.util.Date();
  %>
   Hello! The time is:
  <%
     out.println( date );
     out.println( "<br />Your machine's address is: " );
     out.println( request.getRemoteHost());
  %>
</body>
</html>
```

JSP Scripting: Scriptlet - Example 2

The following three examples generate the same output.

One

```
<%
String queryData = request.getQueryString();
out.println("Query string passed in: " + queryData);
%>
```

Two

```
<% String queryData = request.getQueryString(); %>
Query string passed in:  <%= queryData %>
```

Three

```
Query string passed in: <%= request.getQueryString() %>
```

JSP Scripting: Scriptlet – Conditional Statements

- You can also use the scriptlet to **conditionally generate HTML**.
- Code inside a scriptlet is inserted within the method _jspService of the JSP Servlet class (~ the service method of a Servlet).
- Static HTML before and after a scriptlet goes inside out.write().

passfail.jsp

```
<!DOCTYPE html>
<html><body>
If you don't attend lecture,
<% if (Math.random() < 0.5) { %>
you would still pass the course.
<% } else { %>
you will definitely fail the course.
<% } %>
</body></html>
```

In JSP Servlet class

```
…
out.write("If you don't attend lecture,\n");
if (Math.random() < 0.5) {
    out.write("\n");
    out.write("you would still pass the course.\n");
} else {
    out.write("\n");
    out.write("you will definitely fail the course.\n")
}
…
```

JSP Scripting: Declaration

JSP declarations are used to declare something within the scope of the JSP Servlet class, e.g., **instance** variables, methods, or classes within a declaration tag.

outside JSPservlet()

- Syntax: <%! declarations %>

- **JSP scriptlet and JSP declaration are different**:

  - JSP scriptlet only allows you to define ***local variables*** (which is inside a method of the Servlet class).
    JSP declaration only allows you to define ***instance variables*** of the Servlet class.

  - In JSP scriptlet, you can use conditional statements, manipulate objects, and perform arithmetic, which you cannot do within a JSP declaration.

JSP Scripting: Comment

JSP comments are developers' comments that are not sent to the client page.

- Syntax: <%-- our comment --%>
- E.g., <%-- This needs to be fixed. A bug is found. --%>

- Note that HTML comments are shown to the clients in the resultant page.
  - HTML comment:
    <!-- This needs to be fixed. A bug is found. -->

It won't show in source code from browser

JSP Scripting: Comment (cont')

Example's URL pattern: /Lecture03/passfail.jsp

- Check the following JSP page and the resulted HTML source code.

passfail.jsp

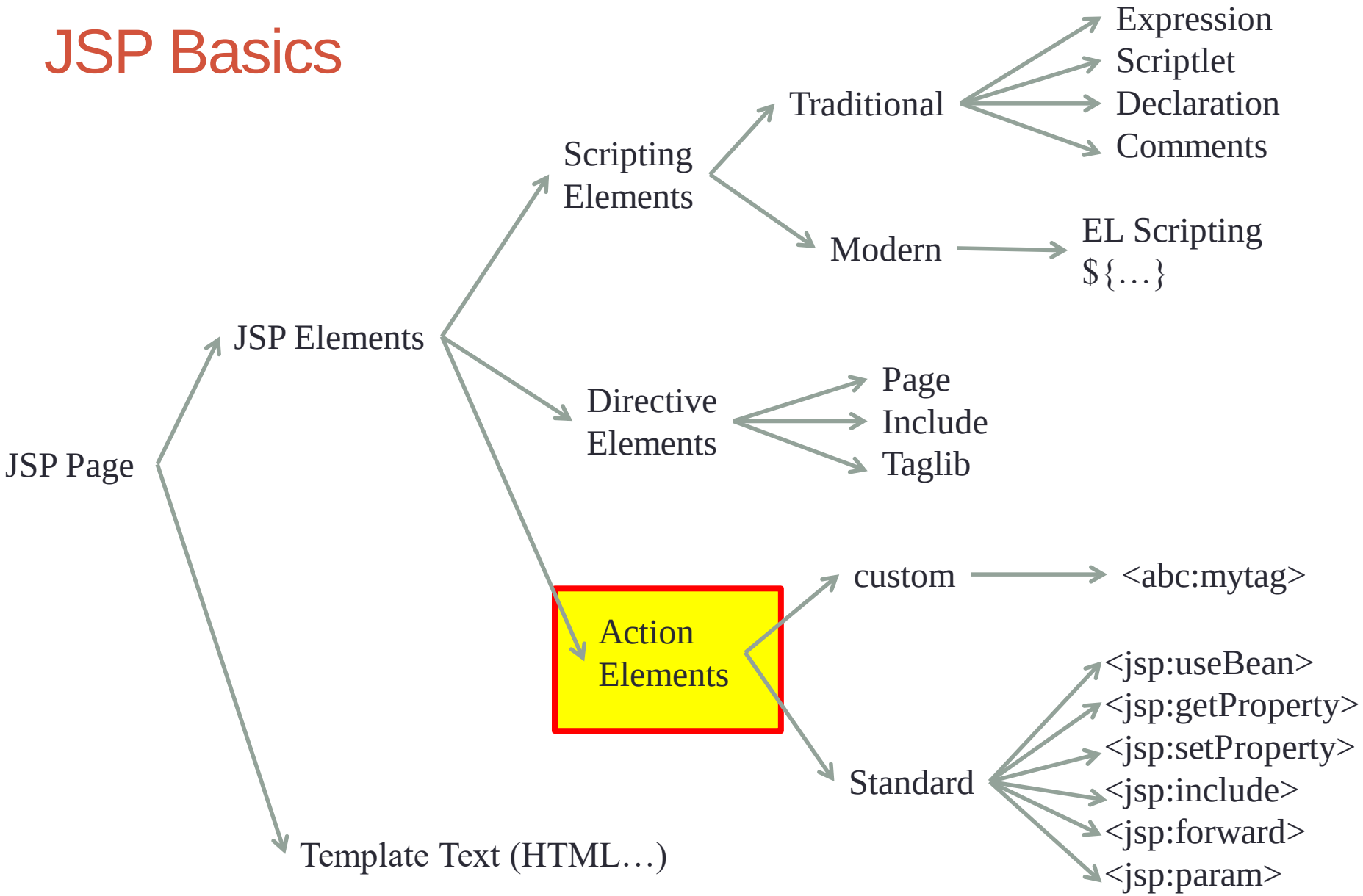
```
<!DOCTYPE html>
<html><body>
<!-- This is a HTML comment -->
<%-- This is a JSP comment --%>
If you don't attend lecture,
<% if (Math.random() < 0.5) { %>
you would still pass the course.
<% } else { %>
you will definitely fail the course.
<% } %>
</body></html>
```

HTML result

```
<!DOCTYPE html>
<html><body>
<!-- This is a HTML comment -->

If you don't attend lecture,

you will definitely fail the course.

</body></html>
```

- JSP directives, declarations, scriptlets, and other JSP tags would become **empty lines** in the HTML response output.

JSP Basics


JSP Page
JSP Elements
Scripting Elements
Traditional
Expression
Scriptlet
Declaration
Comments
Modern
EL Scripting
${…}
Directive Elements
Page
Include
Taglib
Action Elements
custom
<abc:mytag>
Standard
<jsp:useBean>
<jsp:getProperty>
<jsp:setProperty>
<jsp:include>
<jsp:forward>
<jsp:param>
Template Text (HTML…)

JSP Actions

JSP standard action: dynamic behaviour of the page

- Syntax: <jsp:action-name />

- <jsp:include page="next.jsp" />
- <jsp:forward page="next.jsp" />
- <jsp:useBean ... />
- <jsp:setProperty ... />
- <jsp:getProperty ... />

JSP Actions: include

jsp:include action lets you include the output of a page at *request time*. Any changes in the 'included' page can be picked up without changing the main page.

include directive
→ import the code
JSP include
→ import the executed output

You can include

- The content of an HTML page
- The content of a plain text document
- The output of a JSP page
- The output of a Servlet

Note: You cannot include JSP code (unlike include directive). The behaviour is the same as RequestDispatcher.include().

include header and footer

```
<jsp:include page="/common/header.jsp"/>
... some more jsp code ...
<jsp:include page="/common/footer.jsp"/>
```

→ execute header.jsp (2)
→ execute footer.jsp (3)

jsp:include vs. include directive

	jsp:include action	**include directive**
Syntax?	<jsp:include page= .../>	<%@ include file= %>
When does inclusion occur?	Request time	Page translation time
What is included?	Output of the included page	Actual content of file
How many servlets result?	More than one (main page, and one for each included page)	One (included file is inserted into the main page, then the page is translated)
maintenance?	changes in included pages does not require recompilation of the main page	the main page needs to be recompiled when the included page changes

JSP Actions: forward

jsp:forward action has the same behaviour as RequestDispatcher.forward().

jsp forwarding

```
<% String destination;
   if (Match.random() > 0.5) {
      destination = "/examples/page1.jsp";
   } else {
      destination = "/examples/page2.jsp";
    }
%>
<jsp:forward page="<%= destination %>" />
```

To use JSP forward, the main page cannot commit any output to the client (same as RequestDispatcher.forward()).

It is generally recommended that 'dispatching' tasks to be done using RequestDispatcher.forward() rather than JSP.

jsp:param Element

jsp:param element can append new *request parameters* to the request object, when a jsp:include or jsp:forward element is invoked.

```
<jsp:include page="..." >
    <jsp:param name="param1" value="value1"/>
</jsp:include>
```

- The included page or forwarded page will see the original request object, with the original parameters augmented with the new parameters.
  - New values take precedence over existing values when applicable.

JavaBeans

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.

- It provides **a default, no-argument constructor.**
- It should be **serializable** and implement the Serializable interface.
- It may have a number of **properties** (i.e., variables) which can be read or written.
- All variables have **accessor** (get) and **mutator** (set) **methods**.

JavaBeans are usually used to represent a Data Transfer Object (DTO) or a process.

A JavaBean Example

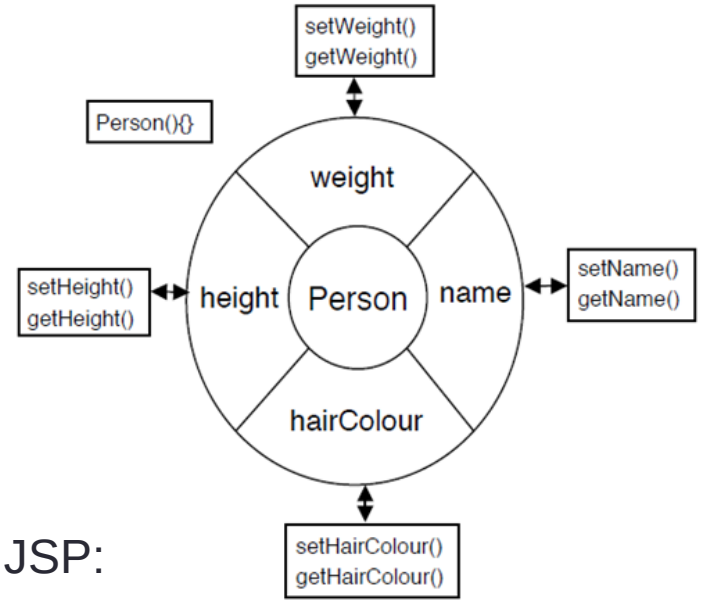
```
package hkmu.comps380f;
import java.io.Serializable;

public class HelloBean implements Serializable {
    private String name;
    private String email;

    public HelloBean() {}

    public String getName() { return name; }
    public String getEmail() { return email; }

    public void setName(String name) { this.name = name; }
    public void setEmail(String email) { this.email = email; }
}
```

No argument, default constructor

- Each JSP page can be associated with one or more JavaBeans.
- They are reside on the server.
  - There is a constructor without parameter.
  - It is serializable.
  - All variables have accessor (get) and mutator (set) methods.

JSP Actions: useBean

- A bean is an object with private *properties*.
- Access to a JavaBean is strictly through "set" or "get" methods.
- The setter and getter methods follow a naming convention.



JSP actions provide strong support using beans in JSP:

- jsp:useBean: create a new instance of a bean

```
<jsp:useBean id="beanName" class="package.Class" />
```

- jsp:getProperty: read and output the value of a bean property (i.e., calling a getter method)

```
<jsp:getProperty name="beanName" property="propertyName" />
```

- jsp:setProperty: modify a bean property (i.e., calling a setter method)

```
<jsp:setProperty name="beanName" property="propertyName"
value="propertyValue" />
```

JSP Actions: useBean (cont')

```
<jsp:useBean id="customer" class="com.enterprise.Customer" />
```

- The above example reads "instantiate an object of the class '**com.enterprise.Customer**' and bind it to the variable specified by **id**."
- The variable **customer** is created inside the service() method, i.e., it is a local variable.
- You must use the fully qualified class name in the **class** attribute.
  - Even if you use `<%@ page import … %>`
- The JavaBean classes can be placed under:
  - /WEB-INF/classes/*appropriate package directory*/
  - /WEB-INF/lib/*inside jar files*

Sharing Beans: Using scope attribute

```
<jsp:useBean id="customer" class="com.enterprise.Customer"
                                 scope="request" />
```

- When **`jsp:useBean`** instantiates a bean, you can ask the web container to share it across various scopes: **page, request, session, application**

- The web container looks for an existing bean of the specified name in the designated location.
  - If it fails to locate the bean in the location, it creates a new bean instance.
  - Otherwise, it uses the existing bean instance.

- It is generally recommended that you use different variable names (i.e., the **`id`** attribute) for different scopes.

Sharing Beans: Using scope attribute (cont')

- **`<jsp:useBean ... scope="page" />`**
  This is the default value. The bean is not shared. Thus, a new bean is created for each request.

- **`<jsp:useBean ... scope="request" />`**
  The bean is bound to the HttpServletRequest object. The bean is shared for the duration of the current request (e.g., forwarding).

- **`<jsp:useBean ... scope="session" />`**
  The bean is bound to the HttpSession object associated with the current request.

- **`<jsp:useBean ... scope="application" />`**
  The bean is bound to the ServletContext object. The bean is shared by all JSP pages in the application.

Revisit the EmailServlet Example

EmailServlet.java

```
package hkmu.comps380f;

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

public class EmailServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        RequestDispatcher view
                = request.getRequestDispatcher("/WEB-INF/jsp/email_form.jsp");
        view.forward(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        RequestDispatcher view
                = request.getRequestDispatcher("/WEB-INF/jsp/email_echo.jsp");
        view.forward(request, response);
    }
}
```

web.xml

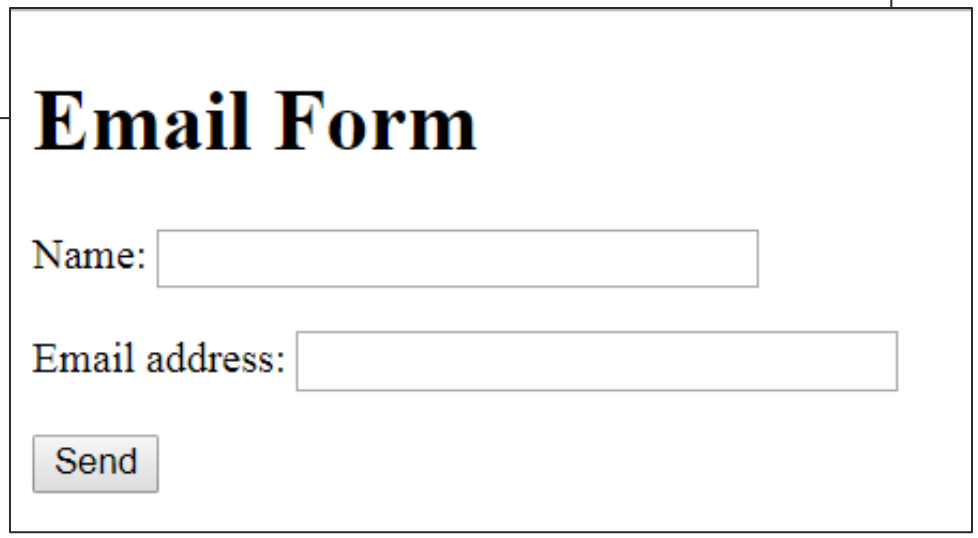
```
<servlet>
    <servlet-name>emailServlet</servlet-name>
    <servlet-class>hkmu.....EmailServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>emailServlet</servlet-name>
    <url-pattern>/echoEmail</url-pattern>
</servlet-mapping>
```

Example's URL pattern: /Lecture03/echoEmail

Revisit the EmailServlet Example (cont')

email_form.jsp

```
<!DOCTYPE html>
<html>
    <head>
        <title>Email Form</title>
    </head>
    <body>
        <h1>Email Form</h1>
        <form method ="POST" action="echoEmail">
            <p>Name: <input type="text" name="name" size="30"/></p>
            <p>Email address: <input type="text" name="email" size="30"/></p>
            <p><input type="submit" value="Send"/></p>
        </form>
    </body>
</html>
```



Revisit the EmailServlet Example (cont')

email_echo.jsp

```
<!DOCTYPE html>
<html>
   <body>
      <jsp:useBean id="hello" scope="session"
                                   class="hkmu.comps380f.HelloBean" />
      <jsp:setProperty name="hello" property="name"
                                   value='<%=request.getParameter("name")%>' />
      <jsp:setProperty name="hello" property="email"
                                    value='<%=request.getParameter("email")%>' />
      Hello, your name is <jsp:getProperty name="hello" property="name" />!
      Your email address is <jsp:getProperty name="hello" property="email" />.
   </body>
</html>
```

```
<!DOCTYPE html>
<html>
   <body>


      Hello, your name is Keith!
      Your email address is lklee@hkmu.edu.hk.
   </body>
</html>
```

Hello, your name is Keith! Your email address is lklee@hkmu.edu.hk.

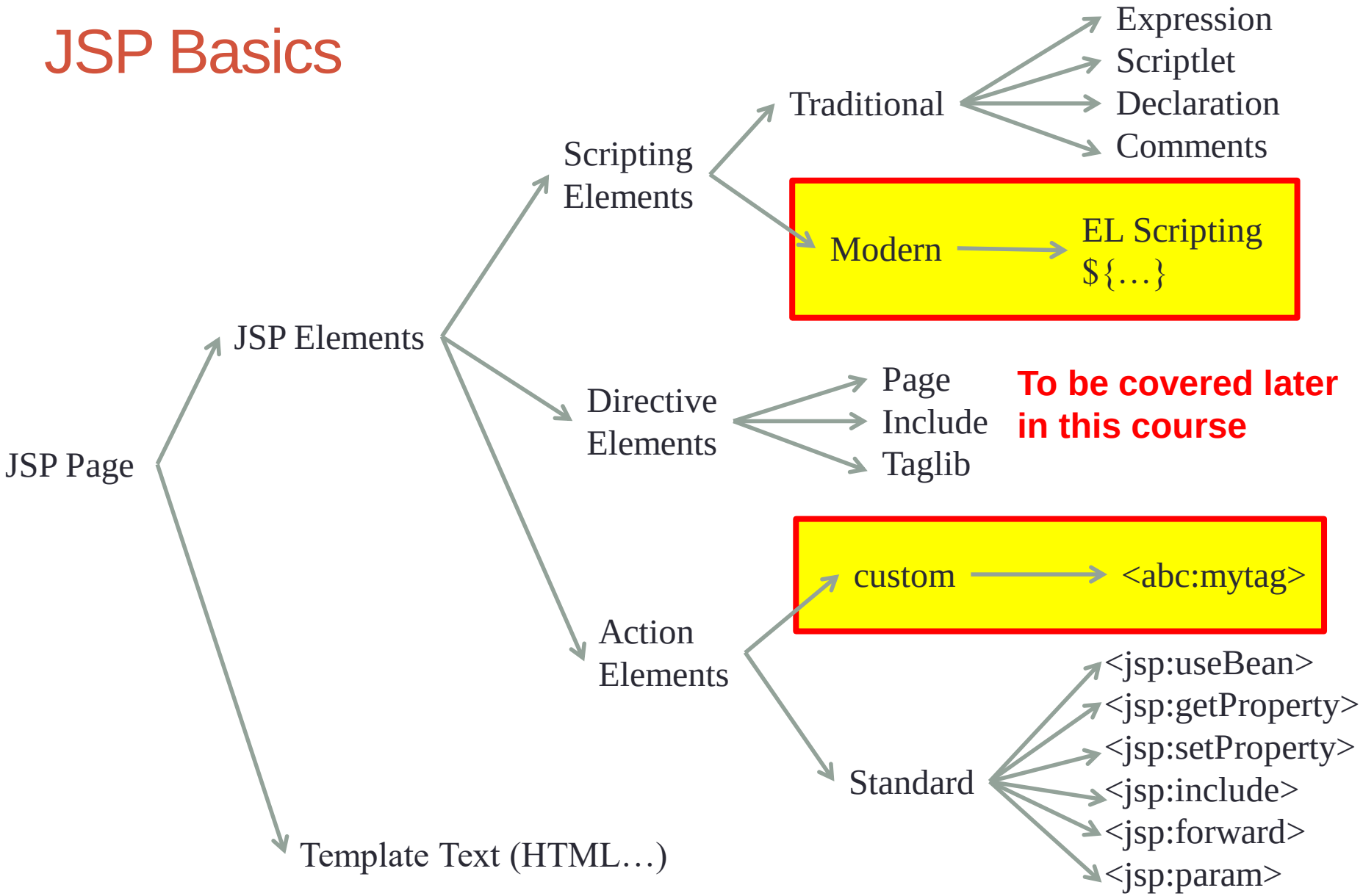
- JSP directives, declarations, scriptlets, and other JSP tags would become empty lines in the HTML response output.

Configuring more JSP properties in web.xml

```
<jsp-config>
   <jsp-property-group>
      <url-pattern>*.jsp</url-pattern>
      <url-pattern>*.jspf</url-pattern>
      <page-encoding>UTF-8</page-encoding>
      <scripting-invalid>false</scripting-invalid>
      <include-prelude>/WEB-INF/jsp/base.jspf</include-prelude>
      <trim-directive-whitespaces>true</trim-directive-whitespaces>
      <default-content-type>text/html</default-content-type>
   </jsp-property-group>
</jsp-config>
```

- Setting **<trim-directive-whitespaces>** to true instructs the JSP translator to remove from the response output any white space only text created by JSP directives and other JSP tags (e.g., by JSTL).
- Setting **<scripting-invalid>** to true allows us to completely disable Java within JSPs (e.g., for security reason).
- Note that we need to *follow the above property order exactly* in web.xml.

JSP Basics


JSP Page
JSP Elements
Template Text (HTML…)
Scripting Elements
Traditional
Expression
Scriptlet
Declaration
Comments
Modern
EL Scripting
${…}
Directive Elements
Page
Include
Taglib
To be covered later in this course
Action Elements
custom
<abc:mytag>
Standard
<jsp:useBean>
<jsp:getProperty>
<jsp:setProperty>
<jsp:include>
<jsp:forward>
<jsp:param>