Persistent

Java Server Pages I

Adding dynamism to static content

Agenda

- Introducing JSP
- A sample JSP
- Lifecycle
- JSP API
- The container generated servlet
- Config and context parameters
- JSP implicit objects
- Attributes and scopes
- Directives

JSP

- What is JSP ?
  - JSP stands for Java Server Pages
  - It is a specification drafted by Sun which allows web developers and designers to rapidly develop, easily maintain, information rich dynamic web pages.
  - JSP technology allows programmers to intermix html with Java code.

- Why JSP ?
  - Cleanly separate the presentation or user interface from content generation.
  - Avoid ugly, error-prone sequence of print() and println() statements in a servlet.

What does a JSP page contain ?

- Template text
  - Static html content or tags delivered as is in a response
- Comments
  - Descriptive/Informational blocks of general text.
- Directives
  - Special instructions to a container at page translation time
- Scriptlets
  - Blocks of Java code
- Declarations
  - Instance or class variable declarations, instance or class methods and nested class definitions in a JSP pages' servlet class.

What does a JSP page contain ?

- Expression
  - Any Java language statement that produces a result. This result is then inserted in the response along with other template text.
- Standard actions
  - A basic set of tags provided by container vendor for various operations like working with Java Beans components, embedding applets, forwarding or including another web component etc.
- JSTL
  - Java Server Pages Standard Tag Library is a standard set of tags which provide some very basic functionality like flow control, manipulating XML documents, internationalization, database access and commonly used methods.
- Custom actions/tags
  - User-defined tags
- EL
  - Expression Language makes it possible to easily access application data stored in JavaBeans components, various scopes and write simple expressions.

A sample JSP

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%> <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
        <head>
                <title>A sample JSP</title>
        </head>
        <body>
                <%-- This is a JSP comment --%>
                <%!
                        private int instanceVariable = 100;
                %>
                <%
                        out.println("This message came from a scriptlet");
                %>
```

Directiv

Template text

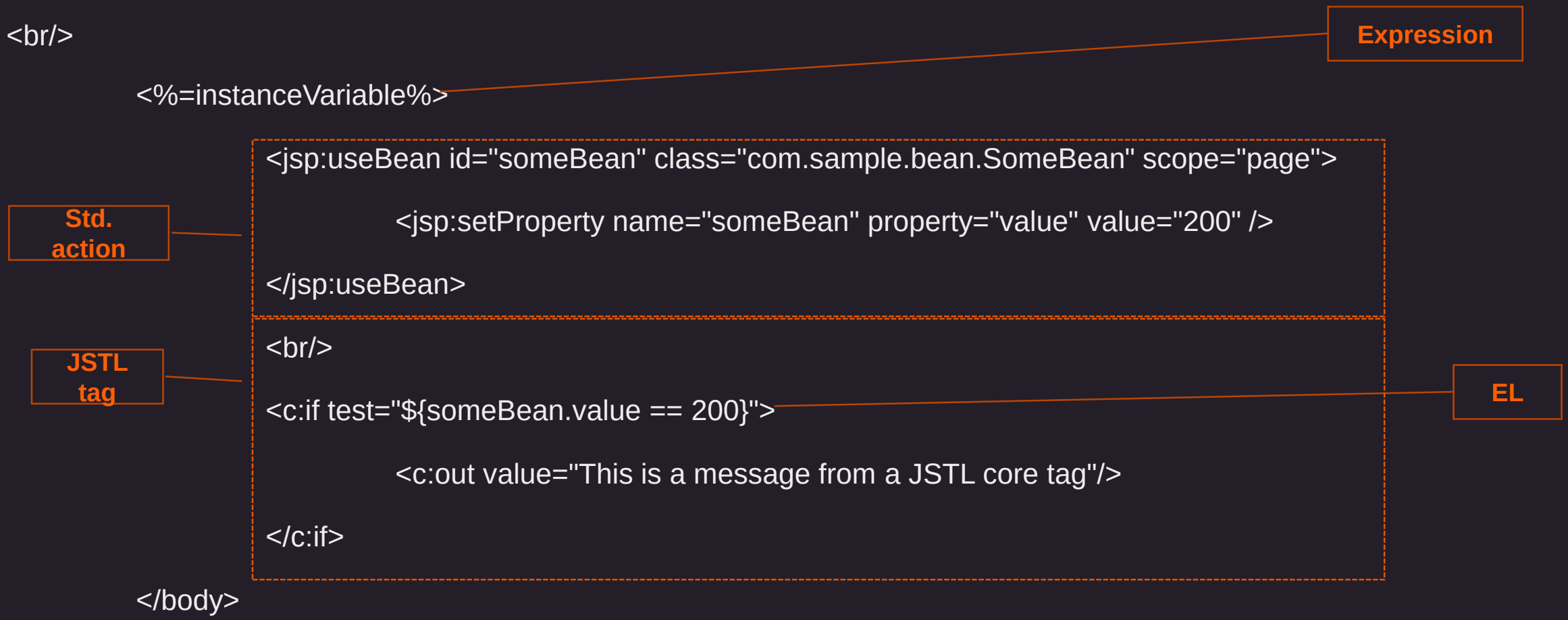
Comment

Declaration

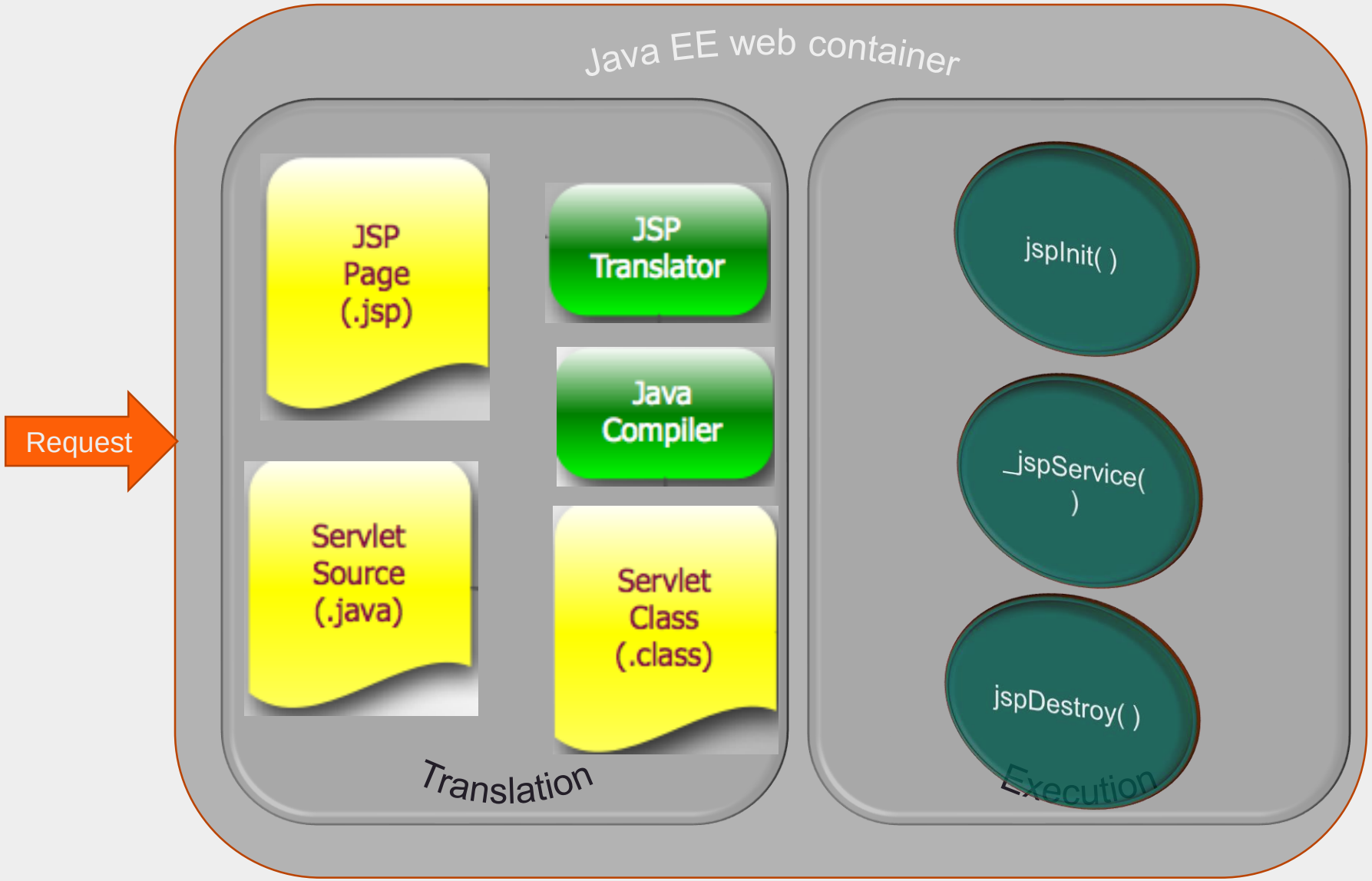
Scriptlet

Continued

```
<br/>
        <%=instanceVariable%>
        <jsp:useBean id="someBean" class="com.sample.bean.SomeBean" scope="page">
                <jsp:setProperty name="someBean" property="value" value="200" />
        </jsp:useBean>
        <br/>
        <c:if test="${someBean.value == 200}">
                <c:out value="This is a message from a JSTL core tag"/>
        </c:if>
</body>
</html>
```



Lifecycle of a JSP



Persistent

So, what happened to my JSP page

```
package org.apache.jsp;
// container generated import statements
public final class sample_jsp extends org.apache.jasper.runtime.HttpJspBase implements
org.apache.jasper.runtime.JspSourceDependent{
        private int instanceVariable = 100;
        // some container generated code
        public void _jspInit() { /* some container generated code */ }
        public void _jspDestroy() { /* some container generated code */ }
        public void _jspService(HttpServletRequest request, HttpServletResponse response) throws
                                                        java.io.IOException, ServletException {
                // some container generated code
                out.write("<html>\r\n");
out.write("\t<head>\r\n");
                out.write("\t\t<title>A sample JSP</title>\r\n");
```

Container specific implementation of the JSP API

JSP declarations

Continued

```
        out.write("\t</head>\r\n");
        out.write("\t<body>\r\n");
        out.println("This message came from a scriptlet");
                out.print(instanceVariable);
        com.sample.bean.SomeBean someBean = null;
        synchronized (_jspx_page_context) {
                someBean = (com.sample.bean.SomeBean) _jspx_page_context.getAttribute("someBean",
                                                                PageContext.PAGE_SCOPE);
                if (someBean == null) { /* some container generated code */ }
        }
        // some container generated code
    }
    // some container generated code
}
```

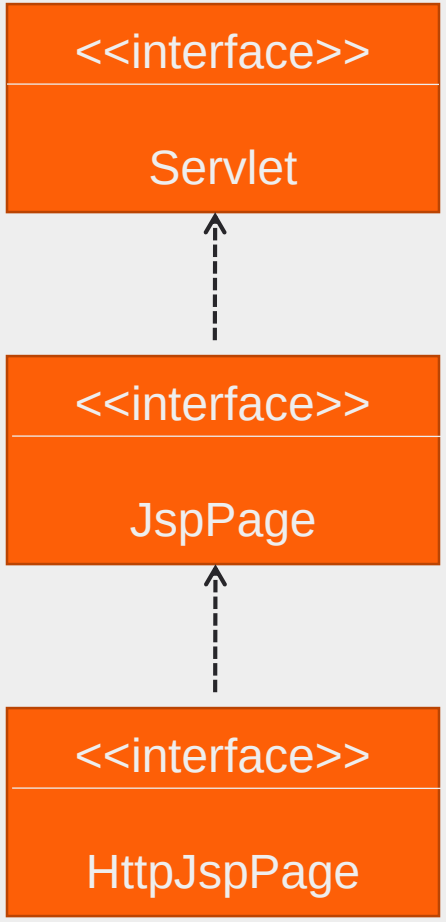
Template text being written to the response

Scriptlet

Expression

Container generated code for the `<jsp:useBean>` standard action

JSP API



Passing init. parameters to a JSP

- You can pass initialization parameters to a JSP.
- **In the web.xml**

```
<web-app ……>
        <servlet>
        <servlet-name>SampleJSP</servlet-name>
                <jsp-file>/sample.jsp</jsp-file>
                <init-param>
                        <param-name>JSP_INIT_PARAM</param-name>
                        <param-value>Sample jsp init. param val.</param-value>
                </init-param>
        </servlet>
        <servlet-mapping>
                <servlet-name>SampleJSP</servlet-name>
                <url-pattern>/sample.jsp</url-pattern>
```

Continued

```
</servlet-mapping>
</web-app>
```

In the JSP page

```
<%
        String value = config.getInitParameter("JSP_INIT_PARAM");
        out.println(value);
%>
```

config is a reference to an object of javax.servlet.ServletConfig. It is one of the many implicit objects available in a JSP.

Accessing context params. in a JSP

- **In the web.xml**

```
<web-app…….>
        <context-param>
                <param-name>WEBMASTER_EMAIL</param-name>
                <param-value>master@email.web</param-value>
        </context-param>
</web-app>
```

- **In the JSP**

```
<%=application.getInitParameter("WEBMASTER_EMAIL") %>
```

- application is a reference to an object of javax.servlet.ServletContext. It is one of the many implicit objects available in a JSP.

JSP implicit objects

- Every JSP has access to the following objects
  - request — Reference to an object of javax.servlet.http.HttpServletRequest
    - One of the two parameters to the _jspService( ) method.
  - response — Reference to an object of javax.servlet.http.HttpServletResponse
    - Second parameter to the _jspService( ) method.
  - config — Reference to an object of javax.servlet.ServletConfig
  - application — Reference to an object of javax.servlet.ServletContext
  - session — Reference to an object of javax.servlet.http.HttpSession
  - out — Reference to an object of javax.servlet.jsp.JspWriter
  - exception — Reference to an object of java.lang.Throwable
    - This object is only available to designated error pages.
  - pageContext — Reference to an object of javax.servlet.jsp.PageContext
    - Encapsulates other implicit objects
  - page — Reference to an object of java.lang.Object
    - The current servlet object. page == this.
- These are known as implicit objects in JSP. They are declared and initialized in the _jspService() method.

Attributes in a JSP

- For a JSP, attributes are data values stored in one of either page, request, session, or application scope.
- A JSP has one additional scope i.e. page scope as compared to a servlet.
- Attributes are commonly used to share information across multiple web components in a JavaEE web application.
- Each attribute is a key-value pair.
- The lifetime of an attribute directly maps to the scope in which it is stored.
- javax.servlet.jsp.PageContext object is used to store page scope attributes
- javax.servlet.http.HttpServletRequest object is used to store request scope attributes
- javax.servlet.http.HttpSession object is used to store session scope attributes
- javax.servlet.ServletContext object is used to store application scope attributes

PageContext methods - I

- setAttribute(String, Object)
  - Stores an attribute in the page scope
  - **Example:**

    <% pageContext.setAttribute(“SOME_KEY”, ”Some value”); %>

- setAttribute(String, Object, int)
  - Stores an attribute in the specified scope
  - **Example:**

    <% pageContext.setAttribute(“KEY”, someObject,

    PageContext.REQUEST_SCOPE); %>

- Object getAttribute(String)
  - Retrieves the value of an attribute from the page scope
  - **Example:**

    pageContext.getAttribute(“SOME

- Object getAttribute(String, int)
  - Retrieves the value of an attribute from the specified scope
  - **Example:**

    <%=pageContext.getAttribute(“KEY“, PageContext.REQUEST_SCOPE) %>

PageContext methods - II

- Object findAttribute(String)
  - Searches for an attribute in the page scope followed by request, session and application scope. Returns a value from wherever it is found first.
  - Example:

    <%=findAttribute(“SOME_KEY”) %>

- removeAttribute(String)
  - Removes the named attribute from all scopes.

    <% removeAttribute(“KEY”); %>

- removeAttribute(String, int)
  - Removes an attribute from the specified scope.

    <% removeAttribute(“SOME_KEY,

    PageContext.PAGE_SCOPE); %>

PageContext methods - III

- The page context implicit object can also be used to retrieve other implicit objects through the following methods:
  - getOut( )
  - getRequest( )
  - getResponse( )
  - getServletConfig( )
  - getSession( )
  - getServletContext( )

Directives

- JSP Directives are instructions given to the container during the translation phase. There are three directives namely,
  - <%@page ..... %>
  - <%@include ..... %>
  - <%@taglib ..... %>

- Each of these directives have one or more attributes.

Page directive attributes - I

<%@page autoFlush=“true” buffer=“16kb” contentType=“text/html; charset=ISO-8859-1” errorPage=“some-error-page.jsp”

isErrorPage=“false” import=“java.util.*, java.io.*” isThreadSafe=“true” language=“java” session=“true” %>

autoFlush = “true | false”

Determines whether the output buffer will be flushed automatically or will raise a “Buffer overflow” exception when the buffer is full. Default value is true.

buffer = “none | size in kb”

Set the size of the output buffer

contentType

Set the content type and the character set of the response

errorPage

Defines URL of the error page to which control would be forwarded in case an exception occurs

isErrorPage = “true | false”

Sets the current page as an error page. Control would be forwarded to this page in case an exception occurs on some other page. The default value is false. When true, the page has access to the exception implicit object.

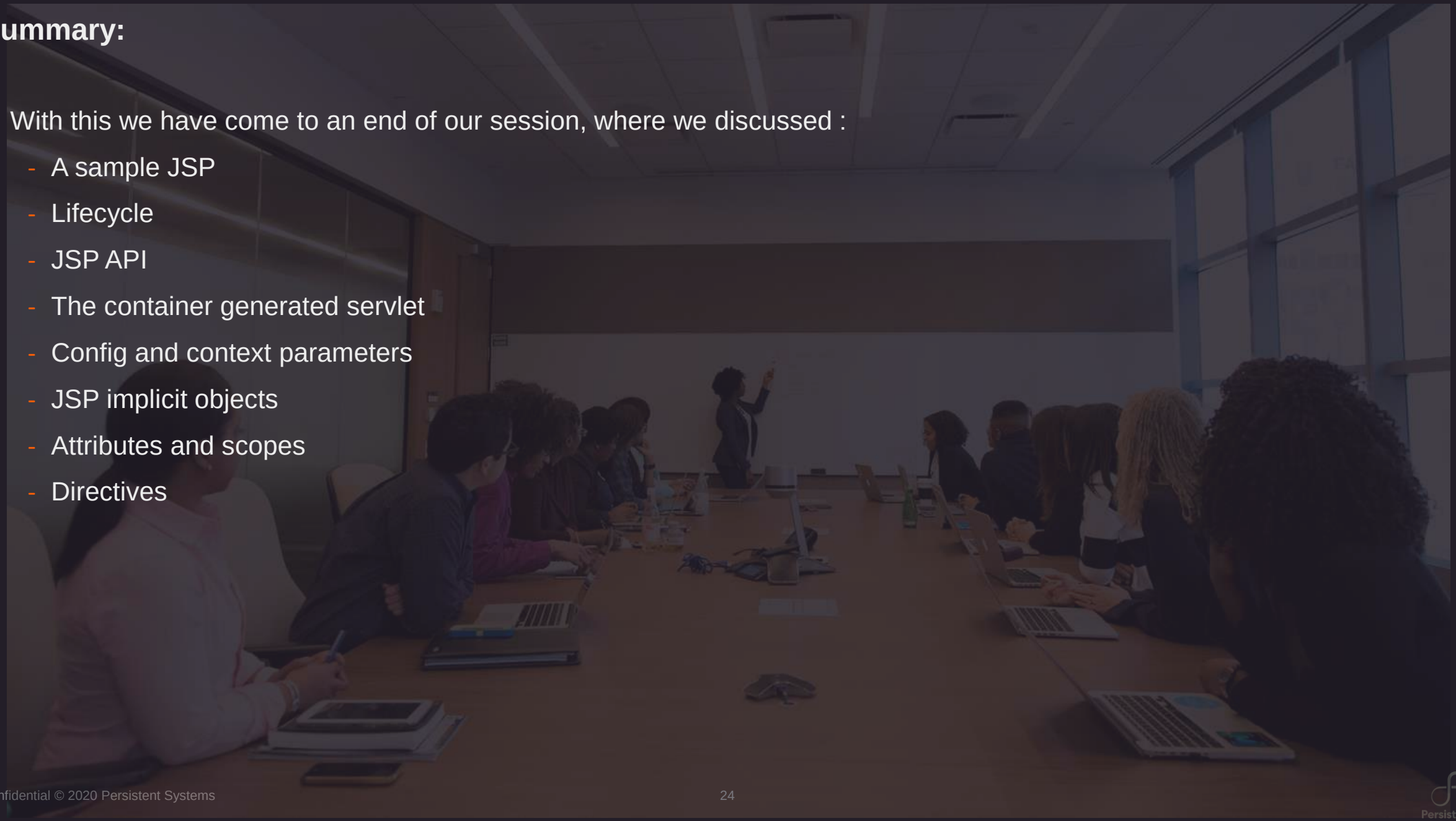
Page directive attributes - II

- import
  - Sets the list of packages which might be needed by this JSP. Two or more package name can be separated by a comma.
- language = “java”
  - Sets the language used in scriptlets, expressions and declarations. Right now only “java” is supported.
- session = “true | false”
  - Sets whether the page will have access to the implicit session object. Default value is true.
- extends
  - Defines the superclass of the current JSP’s generated servlet class. Overrides the class hierarchy provided by the container.
- isELIgnored = “true | false”
  - Defines whether EL expressions are ignored when the page is translated. Default value is false.
- isThreadSafe = “true | false”
  - Specifies whether the page can service more than one request at a time. Default value is true. If set to false the generated servlet is made to implement the javax.servlet.SingleThreadModel interface. This ensures that only one thread can invoke the servlet’s service method at a time.

Include directive

- <%@include file=“some-page.jsp” %>
- Include the contents of another page.
- The include directive works by literally inserting contents of the specified JSP during the translation phase.
- This is much different from other forms of so-called “programmatic includes” which work during the execution phase.

Summary:



- With this we have come to an end of our session, where we discussed :
  - A sample JSP
  - Lifecycle
  - JSP API
  - The container generated servlet
  - Config and context parameters
  - JSP implicit objects
  - Attributes and scopes
  - Directives

Appendix

Thank You

Persistent

Thank you