J2EE与中间件技术

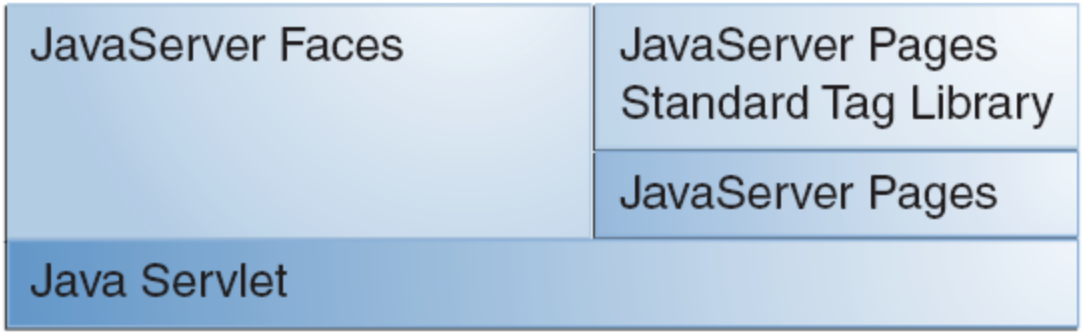
——JSP技术

- *The Java EE 5 Tutorial* at http://docs.oracle.com/javaee/5/tutorial/doc/

Servlet 技术

- Java Servlet technology is the foundation of all the web application technologies.

Figure 7–2 Java Web Application Technologies



JavaServer Pages Technology

- JavaServer Pages (JSP) technology allows you to easily create web content that has both static and dynamic components. JSP technology makes available all the dynamic capabilities of Java Servlet technology but provides a more natural approach to creating static content.
- The main features of JSP technology are as follows:
  - A language for developing JSP pages, which are text-based documents that describe how to process a request and construct a response
  - An expression language for accessing server-side objects
  - Mechanisms for defining extensions to the JSP language

What Is a 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, WML, and XML), and JSP elements, which construct dynamic content.
- The recommended file extension for the source file of a JSP page is .jsp.
- The JSP elements in a JSP page can be expressed in two syntaxes, standard and XML, though any given file can use only one syntax.

HelloJsp.jsp

```
<html>
    <head>
        <title>Hello World</title>
    </head>
    <body>
        <%
            out.print("<p><b>Hello World!</b>");
        %>
    </body>
</html>
```

The Life Cycle of a JSP Page

- A JSP page services requests as a servlet. Thus, the life cycle and many of the capabilities of JSP pages (in particular the dynamic aspects) are determined by Java Servlet technology.
- When a request is mapped to a JSP page, the web container first checks whether the JSP page's servlet is older than the JSP page. If the servlet is older, the web container translates the JSP page into a servlet class and compiles the class. During development, one of the advantages of JSP pages over servlets is that the build process is performed automatically.

Translation and Compilation

- During the translation phase each type of data in a JSP page is treated differently.
  - Static data is transformed into code that will emit the data into the response stream.
  - JSP elements are treated as follows:
    - Directives are used to control how the web container translates and executes the JSP page.
    - Scripting elements are inserted into the JSP page's servlet class.
    - Expression language expressions are passed as parameters to calls to the JSP expression evaluator.
    - jsp:[set|get]Property elements are converted into method calls to JavaBeans components.
    - jsp:[include|forward]elements are converted into invocations of the Java Servlet API.
    - The jsp:plugin element is converted into browser-specific markup for activating an applet.
    - Custom tags are converted into calls to the tag handler that implements the custom tag.

Translation and Compilation

- Both the translation and the compilation phases can yield errors that are observed only when the page is requested for the first time.
- If an error is encountered during either phase, the server will return JasperException and a message that includes the name of the JSP page and the line where the error occurred.

The Life Cycle of a JSP Page

- After the page has been translated and compiled, the JSP page's servlet (for the most part) follows the servlet life cycle:
  - 1. If an instance of the JSP page's servlet does not exist, the container:
    - a. Loads the JSP page's servlet class
    - b. Instantiates an instance of the servlet class
    - c. Initializes the servlet instance by calling the jspInit method
  - 2. The container invokes the _jspService method, passing request and response objects.
  - If the container needs to remove the JSP page's servlet, it calls the jspDestroy method.

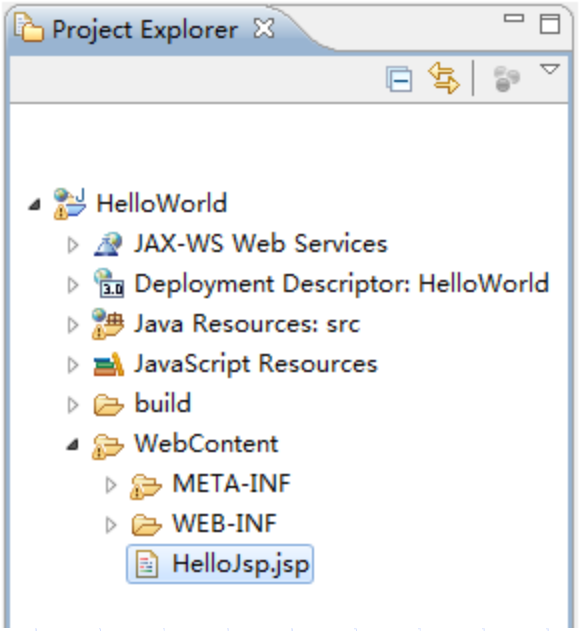
目录结构

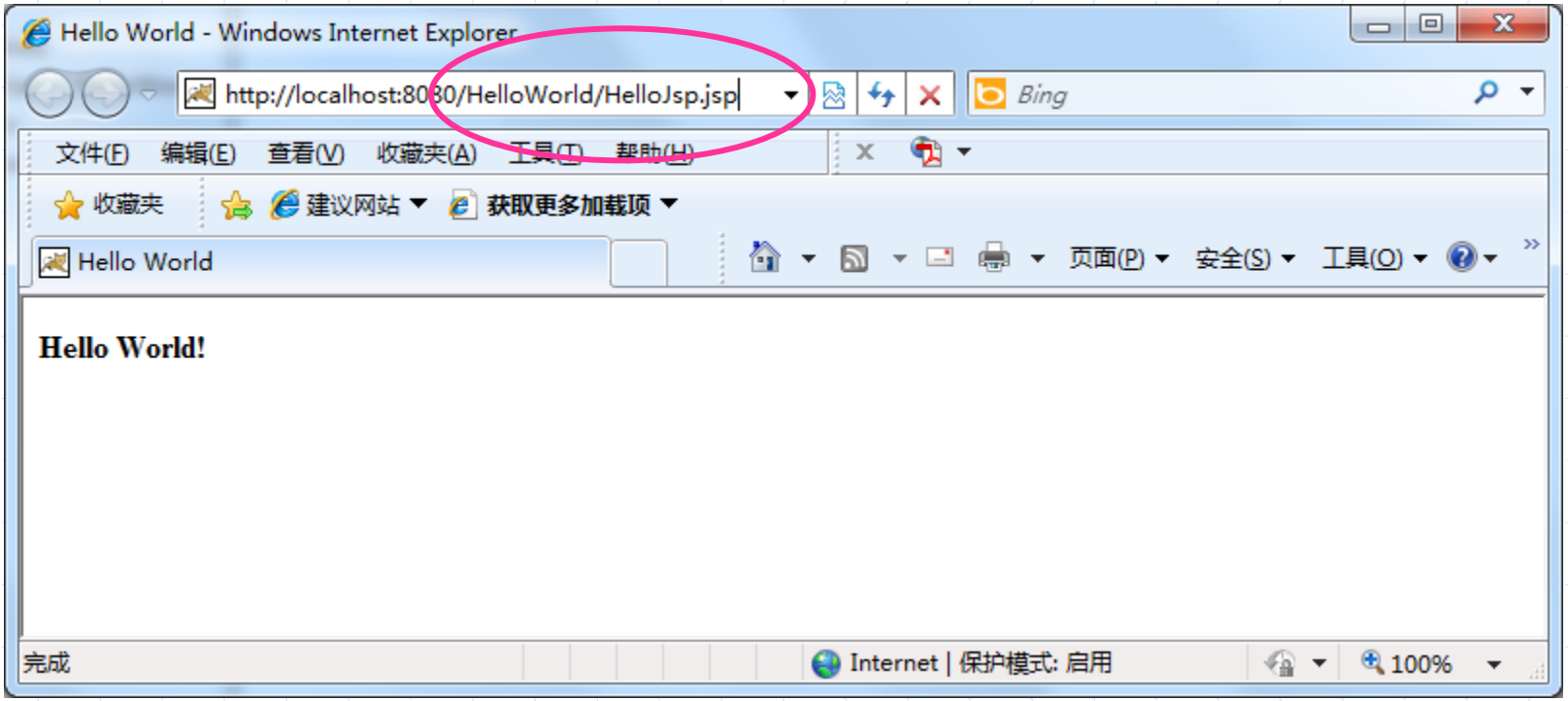
- HelloWorld
  - ——HelloJsp.jsp
  - ——WEB-INF
    - ————classes

Servlet存放目录

- <Tomcat_home>\work\Catalina\localhost\HelloJsp\org\apache\jsp

Eclipse


Project Explorer
HelloWorld
JAX-WS Web Services
Deployment Descriptor: HelloWorld
Java Resources: src
JavaScript Resources
build
WebContent
META-INF
WEB-INF
HelloJsp.jsp


Hello World - Windows Internet Explorer
http://localhost:8080/HelloWorld/HelloJsp.jsp
Bing
文件(F) 编辑(E) 查看(V) 收藏夹(A) 工具(T) 帮助(H)
收藏夹
建议网站
获取更多加载项
Hello World
页面(P)
安全(S)
工具(O)
Hello World!
完成
Internet | 保护模式: 启用
100%

JSP技术

- 加速了创建动态和个性化内容的Web应用的过程
  - 程序开发组：编写业务逻辑和表示业务逻辑方面的代码
  - Web页面设计组：建立HTML页面和有关的图形
- 如果某JSP页面有所调整，下一个对该页面的请求会触发重编译——请求慢
- 只有运行时才能发现页面中的一些错误

JavaServer Pages Documents

- A *JSP document* is a JSP page written in XML syntax.
- when you save the file with a .jspx extension, this page is a JSP document.
- Because it is written in XML syntax, a JSP document is also an XML document and therefore gives you all the benefits offered by the XML standard.

Standard Syntax Versus XML Syntax

TABLE 6–1 Standard Syntax Versus XML Syntax

Syntax Elements	Standard Syntax	XML Syntax
Comments	`<%--.. --%>`	`<!-- .. -->`
Declarations	`<%! ..%>`	`<jsp:declaration> .. </jsp:declaration>`
Directives	`<%@ include .. %>`	`<jsp:directive.include .. />`
	`<%@ page .. %>`	`<jsp:directive.page .. />`
	`<%@ taglib .. %>`	`xmlns:prefix="tag library URL"`
Expressions	`<%= ..%>`	`<jsp:expression> .. </jsp:expression>`
Scriptlets	`<% ..%>`	`<jsp:scriptlet> .. </jsp:scriptlet>`

HelloJsp.jspx

```
<jsp:root
xmlns:jsp="http://java.sun.com/JSP/Page"
version="2.0" >
    <html>
        <head>
            <title>Hello World</title>
        </head>
        <body>
            <jsp:scriptlet>
                out.print("Hello World!");
            </jsp:scriptlet>
        </body>
    </html>
</jsp:root>
```

Directives

- Directives are used to control how the Web container translates and executes the JSP page.

	XML语法	标准语法
page指令	<jsp:directive.page attribute list/>	<%@ page property-attribs %>
include指令	<jsp:directive.include file="filename"/>	<%@ include file="filename" %>
Taglib指令	xmlns:prefix="tag library URL"	<%@ taglib uri="uri" prefix="tagPrefix" %>

page directive

- You can control various JSP page execution parameters by using page directives.
  - session, import, extends, contentType, buffer, ThreadSafe, errorPage

page指令

- <%@ page attribute list %>
- 属性— attribute：描述了对JSP容器有意义的设置
- 例如：
  - < %@ page session= " true " %>
    - 指定当前的JSP页面访问请求应包括在一个HTTP会话中，缺省选项" true "
    - request.getSession(true)——安全问题！

page指令

- 可以包含多条属性：
  - <% @ page session= " true " import= " java.util.* " % >
    - 包括一个HTTP会话；
    - 导入java.util包中的所有类，例如：
      - <% @ page import= " package.* " %>
      - < % @ page import= " java.util.Hashtable, java.util.Vector " % >

import

- 下列包是隐含导入到JSP页面中的
  - java.lang.*
  - javax.servlet.*
  - javax.servlet.jsp.*
  - javax.servlet.http.*

extends

- <%@ page extends= "
  com.learnweblogic.JspCommon " %>
  - 扩展其他类

contentType

- <% @ page contentType= "text/html,charset=GBK" %>
  - 设置页面的文本类型和字符编码
  - The purpose of the contentType directive is to allow the browser to correctly interpret the resulting content.
- Creating Static Content
  - Static content can be expressed in any text-based format, such as HTML, WML, and XML. The default format is HTML.
  - If you want to use a format other than HTML, at the beginning of your JSP page you include a page directive with the contentType attribute set to the content type.
- You also use the contentType attribute to specify the encoding of the response

buffer

- When a JSP page is executed, output written to the response object is automatically buffered. You can set the size of the buffer using the following page directive:
  - <%@ page buffer="none|*xxx*kb" %>
- A larger buffer allows more content to be written before anything is actually sent back to the client, thus providing the JSP page with more time to set appropriate status codes and headers or to forward to another web resource.
- A smaller buffer decreases server memory load and allows the client to start receiving data more quickly.

Servlet:设置缓冲区

- 缺省情况下：写入输出流的内容会立即发送给客户
- 当发生错误时，客户看到的是页面的一部分和错误页面组成的页面
- 需要缓冲页面内容
  - response.setBufferSize(8192);
  - PrintWriter out= response.getWriter();

errorPage

- Any number of exceptions can arise when a JSP page is executed. To specify that the web container should forward control to an error page if an exception occurs, include the following page directive at the beginning of your JSP page:
  - <%@ page errorPage="*file-name*" %>
- <%@ page errorPage="errorpage.jsp"%>
  - The following page directive at the beginning of errorpage.jsp indicates that it is serving as an error page:
    - <%@ page isErrorPage="true" %>
  - This directive makes an object of type javax.servlet.jsp.ErrorData available to the error page so that you can retrieve, interpret, and possibly display information about the cause of the exception in the error page.

Servlet:错误处理

- 指定容器对某种异常返回特定的错误页面
- web.xml
  - <error-page>
    - <exception-type>*exceptionname*</ exception-type >
    - <location>/*errorpagename*</ location >
  - </error-page >

isThreadSafe

- 其它属性
  - isThreadSafe= "true|false"
    - false意味着一次只能有一个线程执行JSP中的代码

include directive

- The include directive is processed when the JSP page is *translated* into a servlet class.
- The effect of the directive is to insert the text contained in another file (either static content or another JSP page) into the including JSP page.
- You would probably use the include directive to include banner content, copyright information, or any chunk of content that you might want to reuse in another page.
- The syntax for the include directive is as follows:
  - <%@ include file="filename" %>

include指令

- 实现JSP页面的模块化，使JSP的开发和维护变得相当简单
  - 例如：一个站点设计，常规的重用部分（如导航条或Web站点上部的扉页），使用单独的JSP页面生成这些组成部分，然后使用include指令，把这些页面加到自己的页面中

include指令

- <%@ include file= " myresponse.jsp"%>
  - 把myresponse.jsp文件的源代码文本加入到当前的JSP页面中，然后再编译它

Servlet:包含其他Web资源

- RequestDispatcher dispatcher=getServletContext().getRequestDispatcher("/banner.jsp");
- if(dispatcher!=null)
  - dispatcher.include(request,response);

Scripting in JSP Pages

- JSP scripting elements allow you to use Java programming language statements in your JSP pages.
- Scripting elements are typically used to create and access objects, define methods, and manage the flow of control.
- Many tasks that require the use of scripts can be eliminated by using custom tag libraries, in particular the JSP Standard Tag Library.
- Because one of the goals of JSP technology is to separate static data from the code needed to dynamically generate content, very sparing use of JSP scripting is recommended.

create and use objects

- There are three ways to create and use objects in scripting elements:
  - Instance and class variables of the JSP page's servlet class are created in *declarations* and accessed in *scriptlets* and *expressions*.
  - Local variables of the JSP page's servlet class are created and used in *scriptlets* and *expressions*.
  - Attributes of scope objects are created and used in *scriptlets* and *expressions*.

JSP Declarations

- A *JSP declaration* is used to declare variables and methods in a page's scripting language. The syntax for a declaration is as follows:
  - <%! *scripting-language-declaration* %>
- When the scripting language is the Java programming language, variables and methods in JSP declarations become declarations in the JSP page's servlet class.

声明

- 声明的变量只在当前页面中可用
  - <%! int foo=3; %>
- 声明方法:
  - < %!
    - private int inc(int x){
      - return x++;
    - }
  - %>

Initializing and Finalizing a JSP Page

- You can customize the initialization process to allow the JSP page to read persistent configuration data, initialize resources, and perform any other one-time activities; to do so, you override the jspInit method of the JspPage interface. You release resources using the jspDestroy method.
- The methods are defined using JSP declarations.
  - <%!
    - public void jspInit() {
      - .....
    - }
  - %>

JSP Scriptlets

- A *JSP scriptlet* is used to contain any code fragment that is valid for the scripting language used in a page.
- The syntax for a scriptlet is as follows:
  - <%
    - *scripting-language-statements*
  - %>
- When the scripting language is set to java, a scriptlet is transformed into a Java programming language statement fragment and is inserted into the service method of the JSP page's servlet. A programming language variable created within a scriptlet is accessible from anywhere within the JSP page.

Scriptlet

- 例如:
  - <%
    - foo=int(foo);
  - % >

JSP Expressions

- A *JSP expression* is used to insert the value of a scripting language expression, converted into a string, into the data stream returned to the client.
- When the scripting language is the Java programming language, an expression is transformed into a statement that converts the value of the expression into a String object and inserts it into the implicit out object.
- The syntax for an expression is as follows:
  - <%= *scripting-language-expression* %>
- Note that a semicolon is not allowed within a JSP expression, even if the same expression has a semicolon when you use it within a scriptlet.

隐式对象

- 脚本元素允许指定任意的Java代码，让容器执行（嵌入JSP页面的Java代码段）
  - 与servlet相比：JSP包含许多预定义的隐式对象，不必编写代码生成输出流，查找ServletContext等

Using Implicit Objects

- *Implicit objects* are created by the web container and contain information related to a particular request, page, session, or application.
- Many of the objects are defined by the Java servlet technology underlying JSP technology.
  - servletContext: The context for the JSP page's servlet and any web components contained in the same application.
  - 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.

隐式对象out

- javax.servlet.jsp.JspWriter类的实例；
- 做PrintWriter对象能做的一切事情；
- 调用print()/println()方法，把信息回送给客户端浏览器
  - out.print("<p>Hello World!");
- 作用域是当前页面（page）
  - 每个JSP页面有一个out对象的实例
- 缺省采用缓存；可以使用page指令调整其大小

request/response对象

- javax.servlet.HttpServletRequest/HttpServletResponse的实例
- 使用request对象得到请求信息中的参数
  - Enumeration foo=request.getParameter();
  - while(foo.hasMoreElements()){
    - out.print(foo.NextElement());
  - }
- 使用response对象发送重定向、修改HTTP头、指定URL重写

其他

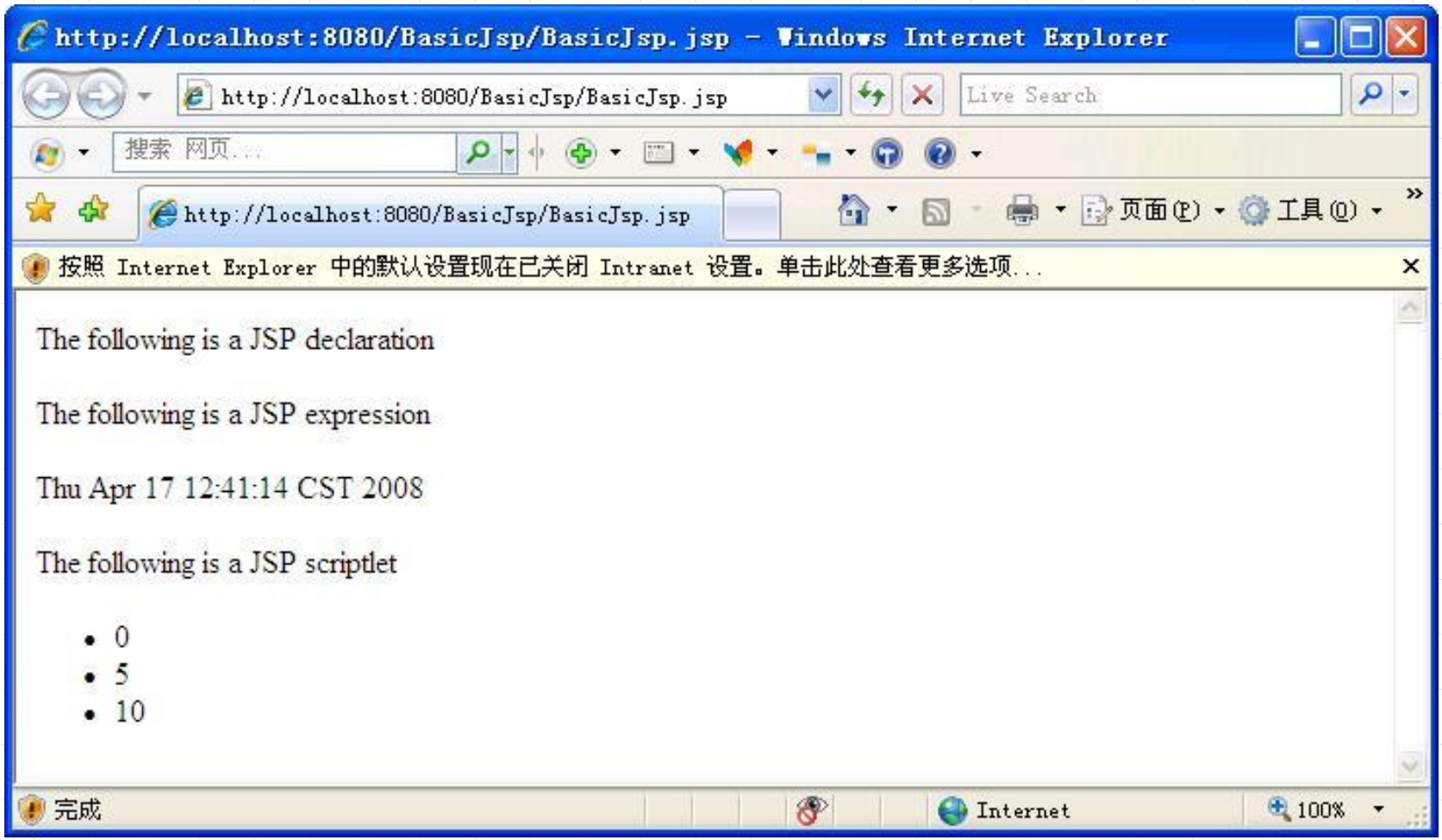
- session对象：
  javax.servlet.http.HttpSession的实例
- application对象：从web.xml获取初始化参数、访问RequestDispatcher
- pageContext对象：对页面作用域的属性的访问

BasicJsp.jsp（标准语法）

```
<!doctype html public "-//w3c/dtd HTML 4.0//en">
<html>
<body>
<p>The following is a JSP declaration</p>
<%!
    int x=5;              //定义实例变量
    private int aMethod(int y){
        return x*y;
    }
%>
```

BasicJsp.jsp

```
<p>The following is a JSP expression </p>
<%=  new java.util.Date() %>
<p>The following is a JSP scriptlet </p>
<UL>
<%
   for(int i=0;i<3;i++){
%>
<LI> <%= aMethod(i) %> </LI>
<%
   }
%>
</UL>
</BODY></html>
```


http://localhost:8080/BasicJsp/BasicJsp.jsp - Windows Internet Explorer
http://localhost:8080/BasicJsp/BasicJsp.jsp
Live Search
搜索 网页...
http://localhost:8080/BasicJsp/BasicJsp.jsp
页面(P)
工具(O)
按照 Internet Explorer 中的默认设置现在已关闭 Intranet 设置。单击此处查看更多选项...
The following is a JSP declaration
The following is a JSP expression
Thu Apr 17 12:41:14 CST 2008
The following is a JSP scriptlet
0
5
10
完成
Internet
100%

BasicJsp.jspx （XML语法）

```
<jsp:root
xmlns:jsp="http://java.sun.com/JSP/Page"
version="2.0" >
    <html>
    <body>
        <p>The following is a JSP declaration</p>
        <jsp:declaration>
            int x=5;
                private int aMethod(int y){
            return x*y;
            }
        </jsp:declaration>
        <p>The following is a JSP expression </p>
        <jsp:expression>   new java.util.Date() </jsp:expression>
```

BasicJsp.jspx

```
        <p>The following is a JSP scriptlet </p>
        <jsp:scriptlet>
            <![CDATA[for(int i=0;i<3;i++){]]>
        </jsp:scriptlet>
        <LI> <jsp:expression> aMethod(i)
        </jsp:expression> </LI>
        <jsp:scriptlet>
            <![CDATA[}]]>
        </jsp:scriptlet>
    </body>
    </html>
</jsp:root>
```

CDATA语法

- XML语法
- Character data: 字符数据，将CDATA标记下的所有文本都当作字符看待
- <![CDATA[

  文本内容
-]]>

动作

- 可以使用scriptlet执行动作提供的任何功能
- 动作改进了scriptlet，易于使用，帮助摆脱java代码
- 类型：标准动作、自定义动作
- 标准动作：
  - <jsp:include>
  - <jsp:forward>
  - <jsp:plugin>
  - <jsp:useBean>, < jsp:setProperty >, < jsp:getProperty >

<jsp:include>

- The jsp:include element is processed when a JSP page is *executed*.
- <jsp:include page= " Copyright.html" />
- The include action allows you to include either a static or a dynamic resource in a JSP file.
  - If the resource is static, its content is inserted into the calling JSP file.
  - If the resource is dynamic, the request is sent to the included resource, the included page is executed, and then the result is included in the response from the calling JSP page.

<jsp:include>

- 与include指令<%@ include file=%>的差别：
  - include指令：包括其他页面，编译时把其他页面的内容加进来，比include动作快
  - include标准动作：使用RequestDispatcher，运行时把其他页面的内容加进来（包括到输出流中）

<jsp:include>

- 被包含的资源为变量时：
  - <jsp:include page= " %=dynamicRef %" />
    - 使用了表达式

<jsp:forward>

- 把当前JSP请求转发到另一个资源上
- <jsp:forward page= " foo.jsp" />
- 与HTTP重定向的差别：
  - HTTP重定向：response.sendRedirect(*myNewURL*)；
    - 发送的请求信息又回送给客户机，让客户机再转发到另一个资源上，新的URL出现在Web浏览器中，需要在服务器和客户机之间增加一次通信
  - forward标准动作：使用RequestDispatcher ，JSP的转发功能是在服务器本身上实现的

jsp:param Element

- When an include or forward element is invoked, the original request object is provided to the target page. If you wish to provide additional data to that page, you can **append parameters** to the request object by using the jsp:param element:
  - <jsp:include page= " /itemdetail.jsp" >
    - <jsp:param name= " itemid " value= " %=itemId" />
  - </jsp: include >
- The scope of the new parameters is the jsp:include or jsp:forward call; that is, in the case of an jsp:include the new parameters (and values) will not apply after the include.

<jsp:plugin>

- You can include an applet or a JavaBeans component in a JSP page by using the jsp:plugin element. This element generates HTML that contains the appropriate client-browser-dependent construct (<object> or <embed>) that will result in the download of the Java Plug-in software (if required) and the client-side component, and in the subsequent execution of any client-side component.
- <jsp:plugin type= “bean|applet”
  code= “classname”
  codebase= “codebase-uri”
  [….]>
- </jsp:plugin>

Bean动作

- <jsp:useBean>
- < jsp:setProperty >
- < jsp:getProperty >

JavaBean Components

- JavaBeans components are Java classes that can be easily reused and composed together into applications. Any Java class that follows certain design conventions is a JavaBeans component.
- JavaServer Pages technology directly supports using JavaBeans components with standard JSP language elements. You can easily create and initialize beans and get and set the values of their properties.

JavaBeans Component Design Conventions

- A JavaBeans component property can be:
  - Read/write, read-only, or write-only
  - Simple, which means it contains a single value, or indexed, which means it represents an array of values
- It must simply be accessible using public methods that conform to the following conventions:
  - For each readable property, the bean must have a method of the form:
    - `PropertyClass getProperty() { ... }`
  - For each writable property, the bean must have a method of the form:
    - `setProperty(PropertyClass pc) { ... }`
- In addition to the property methods, a JavaBeans component must define a constructor that takes no parameters.

UserSearchBean.java

```
package javabean;
public class UserSearchBean{
    public UserSearchBean() {}
    private String keywords;
    public String getKeywords() {
        return this.keywords;
    }
    public void setKeywords(String keywords) {
        this.keywords = keywords;
    }
}
```

Creating and Using a JavaBeans Component

- To declare that your JSP page will use a JavaBeans component, you use a jsp:useBean element.
- There are two forms:
  - <jsp:useBean id="*beanName*" class="*fully-qualified-classname*" scope="*scope*"/>
  - and
  - <jsp:useBean id="*beanName*" class="*fully-qualified-classname*" scope="*scope*">
    - <jsp:setProperty .../> (initializing bean properties)
  - </jsp:useBean>

jsp:useBean

- The jsp:useBean element declares that the page will use a bean that is stored within and is accessible from the specified scope, which can be application, session, request, or page.
- If no such bean exists, the statement creates the bean and stores it as an attribute of the scope object.
- The value of the id attribute determines the *name* of the bean in the scope and the *identifier* used to reference the bean in EL expressions, other JSP elements, and scripting expressions.

jsp:useBean

- The value supplied for the class attribute must be a fully qualified class name. Note that beans cannot be in the unnamed package. Thus the format of the value must be *package-name.class-name*.
- The following element creates an instance of mypkg.myLocales if none exists, stores it as an attribute of the application scope, and makes the bean available throughout the application by the identifier locales:
  - <jsp:useBean id="locales" scope="application" class="mypkg.MyLocales"/>

<jsp:useBean>

- JSP容器将自动的创建相应的JavaBean的实例
- JSP容器将自动的处理JavaBean的清理事项

scriptlet vs JavaBean

- 创建书店的购物车，并保存在一个会话属性中
- scriptlet ：
  - <%
    - ShoppingCart cart=(ShoppingCart)session.getAttribute(" cart ");
    - if(cart==null){
      - cart=new ShoppingCart();
      - session.setAttribute(" cart " ,cart);
    - }
  - %>

scriptlet vs JavaBean

- JavaBean:
  - <jsp:useBean id= "cart " scope= " session " class= " cart.ShoppingCart " / >

Setting JavaBeans Component Properties

- The standard way to set JavaBeans component properties in a JSP page is by using the jsp:setProperty element.
- The syntax of the jsp:setProperty element depends on the source of the property value.

jsp:setProperty

TABLE 5–6 Valid Bean Property Assignments from String Values

Value Source	Element Syntax
String constant	`<jsp:setProperty name="beanName" property="propName" value="string-constant" />`
Request parameter	`<jsp:setProperty name="beanName" property="propName" param="paramName" />`
Request parameter name that matches bean property	`<jsp:setProperty name="beanName" property="propName" />` `<jsp:setProperty name="beanName" property="*" />`

Expression	`<jsp:setProperty name="beanName" property="propName" value="expression" />`
	`<jsp:setProperty name="beanName" property="propName" >`
	`<jsp:attribute name="value">`
	`expression`
	`</jsp:attribute>`
	`</jsp:setProperty>`

jsp:setProperty

- Syntax rules of attribute values used in this table:
  - 1. *beanName* must be the same as that specified for the id attribute in a useBean element.
  - 2. There must be a *setPropName* method in the JavaBeans component.
  - 3. *paramName* must be a request parameter name.

<jsp:setProperty>

- 当Web表单生成HTTP请求时，JSP规范允许使用请求参数自动填充JavaBean属性
  - <jsp:setProperty name="searchBean" property="keywords" param="*param_name*"/>
- 通常，property和param名称相同
  - <jsp:setProperty name="searchBean" property="keywords"/>
- 如果一个JavaBean有多个属性，每个property和param名称匹配，可采用通配符
  - <jsp:setProperty name="searchBean" property="*"/>

jsp:setProperty

- A property set from a constant string or request parameter must have one of the types listed in Table.

TABLE 5–7 Valid Property Value Assignments from String Values

Property Type	Conversion on String Value
Bean Property	Uses `setAsText(`*string-literal*`)`
`boolean` or `Boolean`	As indicated in `java.lang.Boolean.valueOf(String)`
`byte` or `Byte`	As indicated in `java.lang.Byte.valueOf(String)`
`char` or `Character`	As indicated in `java.lang.String.charAt(0)`
`double` or `Double`	As indicated in `java.lang.Double.valueOf(String)`
`int` or `Integer`	As indicated in `java.lang.Integer.valueOf(String)`
`float` or `Float`	As indicated in `java.lang.Float.valueOf(String)`
`long` or `Long`	As indicated in `java.lang.Long.valueOf(String)`
`short` or `Short`	As indicated in `java.lang.Short.valueOf(String)`
`Object`	new `String(`*string-literal*`)`

Retrieving JavaBeans Component Properties

- The jsp:getProperty element converts the value of the property into a String and inserts the value into the response stream:
  - <jsp:getProperty name="*beanName*" property="*propName*"/>
- Note that *beanName* must be the same as that specified for the id attribute in a useBean element, and there must be a get*PropName* method in the JavaBeans component.

search.html——welcome-file

```
<html>
    <head>
        <title>Using Search Bean</title>
    </head>
    <body>
        <h1>Using Search Bean</h1>
        <form action="emptysearch.jsp">
            <p>
                Search keywords:
                <input type="text" size="30" name="keywords"/>
            </p>
            <input type=submit value="Submit">
        </form>
    </body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems,
Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >
<web-app>
    <welcome-file-list>
        <welcome-file>search.html</welcome-file>
    </welcome-file-list>
</web-app>
```


Using Search Bean - Windows Internet Explorer
http://localhost:8080/DataBean/
Live Search
搜索 网页...
Using Search Bean
页面(P)
工具(O)
按照 Internet Explorer 中的默认设置现在已关闭 Intranet 设置。单击此处查看更多选项...
Using Search Bean
Search keywords:
Submit
完成
Internet
100%

emptysearch.jsp

emptysearch.jsp——<jsp:useBean>

```
<%@ page session="false" %>
<jsp:useBean id="searchBean"
scope="request"
class="javabean.UserSearchBean">
    <jsp:setProperty name="searchBean"
    property="keywords"/>
</jsp:useBean>
……
```

<jsp:setProperty> vs *setPropName* method

- 或者

```
<jsp:useBean id="searchBean"
scope="request"
class="javabean.UserSearchBean">
    <%
        searchBean.setKeywords(request.getParameter("keywords"));
    %>
</jsp:useBean>
```

emptysearch.jsp——<jsp:getProperty>

```
<html>
    <head>
        <title>Using Search Bean: no items found</title>
    </head>
    <body>
    <h1>No items found</h1>
    <form action="emptysearch.jsp">
        <p style="color: red;">
            Your search for "
            <jsp:getProperty name="searchBean" property="keywords"/>
            " returned no items. Try another search to find a match.
        </p>
    ......
```

<jsp:getProperty> vs *getPropName* method

- 或者

```
<form action="emptysearch.jsp">
  <p style="color: red;">
    Your search for "
    <%= searchBean.getKeywords() %>
    " returned no items. Try another search to
    find a match.
  </p>
... ...
```

emptysearch.jsp——<jsp:getProperty>

```
    ……
    <p>
        Search keywords:
        <input type="text" size="30" name="keywords" value="
        <jsp:getProperty name="searchBean"
        property="keywords"/>
        "/>
    </p>
    <input type=submit value="Submit">
</form>
</body>
</html>
```

<jsp:getProperty> vs *getPropName* method

- 或者

```
<p>
    Search keywords:
    <%
        out.print("<input type=text size=30 name=keywords
        value=\"");
        out.print(searchBean.getKeywords());
        out.print("\"/>");
    %>
    或者
        <input type= “text”  size= “30”  name= “keywords”
        value= “<%=searchBean.getKeywords()%>” />
</p>
……
```

部署

DataBean

——search.html

——emptysearch.jsp

——WEB-INF

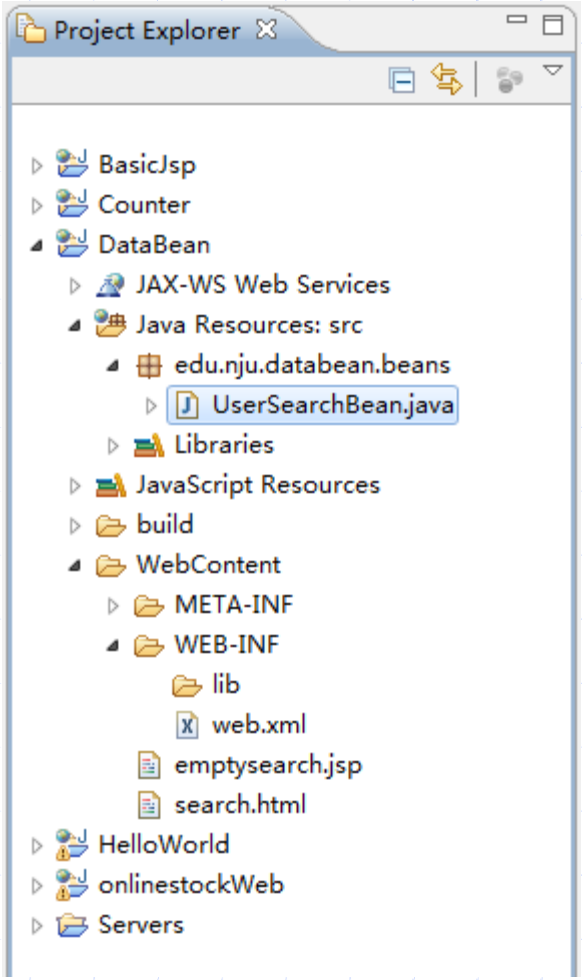
————classes

————javabean

————UserSearchBean.class

————web.xml

Eclipse


Project Explorer
BasicJsp
Counter
DataBean
JAX-WS Web Services
Java Resources: src
edu.nju.databean.beans
UserSearchBean.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
lib
web.xml
emptysearch.jsp
search.html
HelloWorld
onlinestockWeb
Servers

emptysearch.jspx

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
version="2.0" >
<jsp:directive.page session="false"/>
 <jsp:useBean id="searchBean"
                scope="request"
                class="javabean.UserSearchBean">
    <jsp:setProperty name="searchBean" property="keywords"/>
 </jsp:useBean>
 <html>
    <head>
    <title>Using Search Bean: no items found</title>
    </head>
 <body>
 <h1>No items found</h1>
```

emptysearch.jspx

```
<form action="emptysearch.jspx">
    <p style="color: red;">
    Your search for "
    <jsp:getProperty name="searchBean" property="keywords"/>
    " returned no items. Try another search to find a match.</p>
    <p>Search keywords:
    <jsp:text><![CDATA[<input type="text" size="30" name="keywords"
    value="]]></jsp:text>
    <jsp:getProperty name="searchBean" property="keywords"/>
    <jsp:text><![CDATA["/>]]></jsp:text>
    </p>
    <jsp:text><![CDATA[<input type=submit
    value="Submit"/>]]></jsp:text>
</form>
</body>
</html>
</jsp:root>
```

jsp:text

- XML语法独特：
- <jsp:text>——text动作
  - output static data that is not well formed.
  - must not contain any other elements. if you need to nest a tag inside jsp:text, you must wrap the tag inside CDATA.

MVC架构

- MVC.ppt