

Web应用开发 之 JSP技术模型

赵小敏

浙江工业大学计算机科学与技术学院

本节内容

- 3.3 page指令属性
- 3.4 JSP脚本元素
- 3.5 JSP隐含变量

3.3 page指令属性

- page指令用于告诉容器关于JSP页面的总体特性，该指令适用于整个转换单元而不仅仅是它所声明的页面。

```
<%@ page language="java" contentType="text/html;  
    charset=UTF-8" pageEncoding="UTF-8"%>
```

3.3 page指令属性

➤ page指令的常用属性:

- import属性
- contentType属性
- pageEncoding属性
- session属性
- errorPage属性
- isErrorPage属性
- language属性

3.3.1 import属性

- import属性的功能类似于Java程序的import语句，它是将import属性值指定的类导入到页面中。
- 在转换阶段，容器对使用import属性声明的每个包都转换成页面实现类的一个import语句。
- 可以在一个import属性中导入多个包，包名用逗号分开即可：

```
<%@ page import="java.util.*, java.io.*,  
java.text.*, com.demo.*" %>
```

3.3.1 import属性

➤ 为了增强可读性也可以使用多个page指令：

- `<%@ page import="java.util.*" %>`
- `<%@ page import="java.text.*" %>`
- `<%@ page import="com.demo.*" %>`

➤ Java中import语句的顺序是没有关系的，import属性的顺序也没有关系

➤ 容器总是默认导入以下包：

- `java.lang.*`
- `javax.servlet.*`
- `javax.servlet.http.*`
- `javax.servlet.jsp.*`

3.3.2 contentType属性

- `contentType`属性用来指定JSP页面输出的MIME类型和字符集，MIME类型的默认值是`text/html`，字符集的默认值是`ISO-8859-1`。

```
<%@ page contentType="text/html;charset =ISO-8859-1" %>
```

- 上述代码与在Servlet中的下面一行等价：

- `response.setContentType("text/html;charset = ISO-8859-1");`

- 如果页面需要显示中文，字符集应该指定为`UTF-8`，如下所示。

```
<%@ page contentType="text/html;charset=UTF-8" %>
```

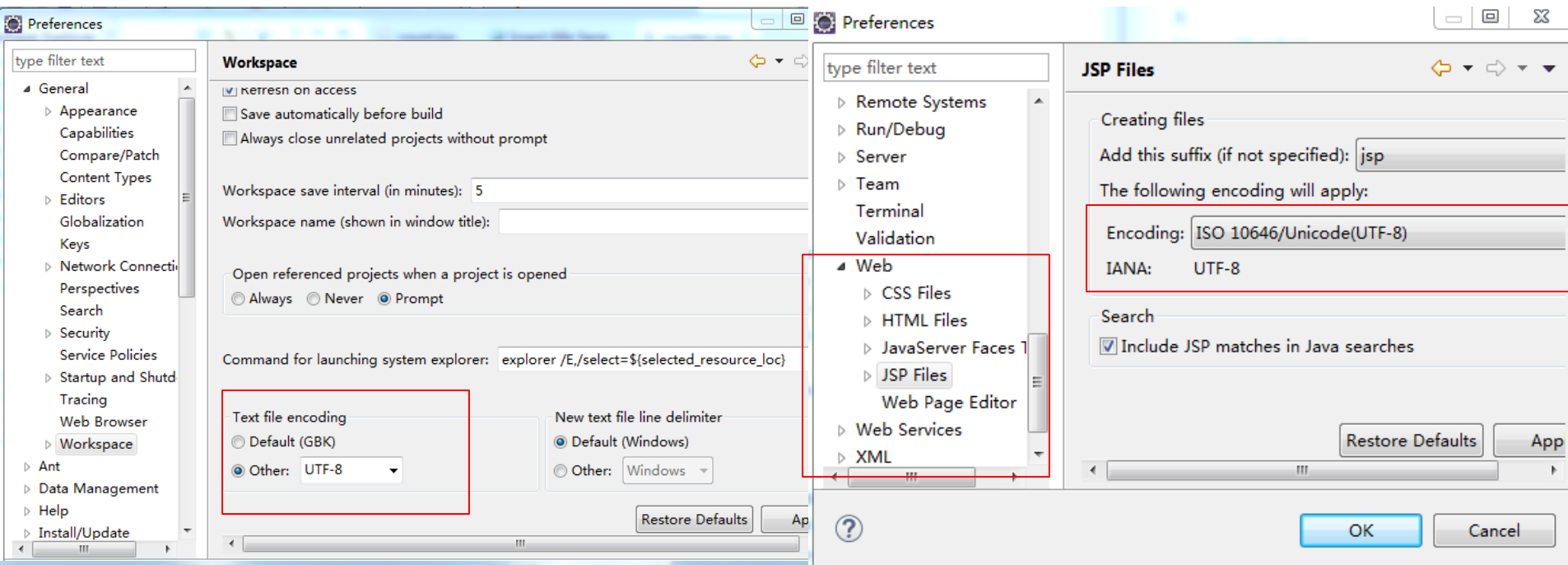
3.3.3 pageEncoding属性

- `pageEncoding`属性指定JSP页面的字符编码，它的默认值为ISO-8859-1。
- 如果设置了该属性，则JSP页面使用该属性设置的字符集编码；如果没有设置这个属性，则JSP页面使用contentType属性指定的字符集。
- 如果页面中含有中文，应该将该属性值指定为UTF-8，如下：

```
<%@ page pageEncoding="UTF-8" %>
```

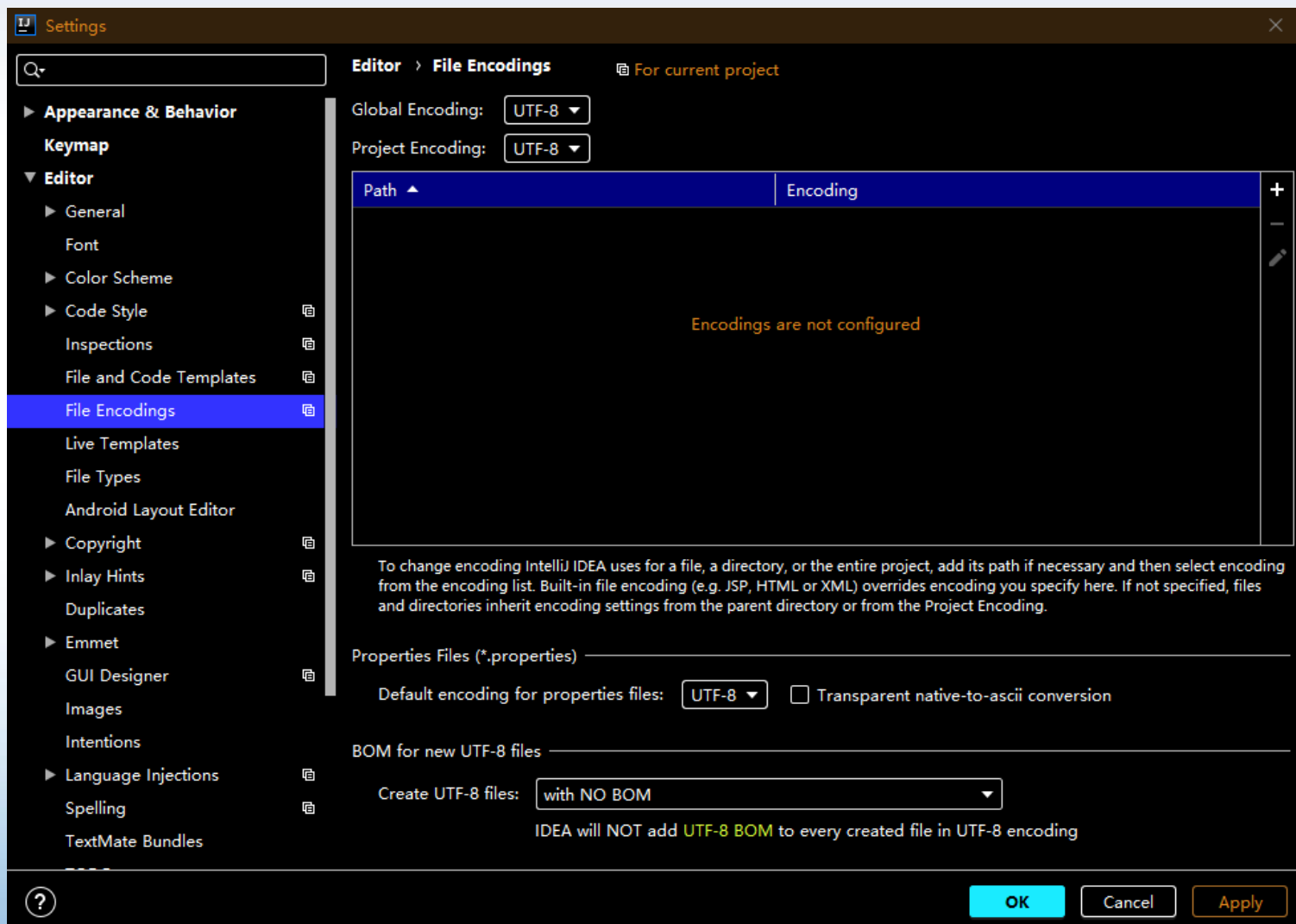

eclipse中设置UTF-8

- 1. windows->Preferences 打开“首选项”对话框；
- 2. 选择general->Workspace, 右侧Text file encoding, 选择Other, 改变为UTF-8。
- 3. 选择Web, 分别打开CSS、HTML、JSP、JavaScript、XML等设置为UTF-8。



IDEA中设置UTF-8

- file->settings..., 在Editor->File Encodings中设置



3.3.4 session属性

- `session`属性指示JSP页面是否参加HTTP会话，其默认值为`true`，在这种情况下容器将声明一个隐含变量`session`。
- 如果不希望页面参加会话，可以加入下面一行：
`<%@ page session = "false" %>`

3.3.5 `errorPage`属性

- 在页面执行过程中，嵌入在页面中的Java代码可能抛出异常。
- JSP规范定义了一种可以使错误处理代码与主页面代码分离的方法，从而提高异常处理机制的可重用性。
- 使用page指令的`errorPage`属性将异常代理给另一个包含错误处理代码的JSP页面。

3.3.5 `errorPage`属性

- 示例: `helloUser.jsp`页面中,
`errorHandler.jsp`被指定为错误处理页面
- 对该JSP页面的请求如果指定了`name`请求参数值,
该页面将正常输出, 如果没有指定`name`请求参数值,
将抛出一个异常, 但它本身并没有捕获异常, 而是
通过`errorPage`属性指示容器将错误处理代理给页
面`errorHandler.jsp`。

例子代码

helloUser.jsp:

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ page errorPage="errorHandler.jsp"%>
<html>
<body>
<%
    if (request.getParameter("name") == null) {
        throw new RuntimeException("没有指定name
请求参数。");
    }
%>
Hello, <%=request.getParameter("name")%>
</body>
</html>
```

errorHandler.jsp:

```
<%@ page contentType="text/html;
charset=UTF-8" %>
<%@ page isErrorPage="true" %>
<html>
<body>
请求不能被处理:
<%=exception.getMessage()%><br>
请重试!
</body>
</html>
```

http://localhost:8080/chapter03/helloUser.jsp

请求不能被处理: 没有指定name 请求参数。
请重试!

http://localhost:8080/chapter03/helloUser.jsp?name=jack

Hello, jack

3.3.5 `errorPage`属性

- `errorPage`的属性值不必是JSP页面，它也可以是静态的HTML页面，例如：

```
<%@ page errorPage="errorHandler.html" %>
```

- 显然，在`errorHandler.html`文件中不能编写小脚本或表达式产生动态信息。

3.3.6 isErrorPage属性

- `isErrorPage`属性指定当前页面是否作为其他JSP页面的错误处理页面。
- `isErrorPage`属性的默认值为`false`。如上例使用的`errorHandler.jsp`页面中该属性必须明确设置为`true`

3.3.7在web.xml中配置错误页面

- 可以在web.xml文件中为整个Web应用配置错误处理页面。使用这种方法还可以根据异常类型的不同或HTTP错误码的不同配置错误处理页面。
- 在web.xml中使用<error-page>元素，它的子元素有<exception-type>，<error-code>和<location>，它们分别指定处理错误的异常类型、HTTP错误码和错误处理页面。其中，前两个元素不能同时出现。

3.3.7在web.xml中配置错误页面

- 声明一个处理算术异常的错误页面。

```
<error-page>
```

```
    <exception-type>
```

```
        java.lang.ArithmeticException
```

```
    </exception-type>
```

```
    <location>/error/arithmeticError.jsp</location>
```

```
</error-page>
```



http://localhost:8080/chapter03/arithmeticExceptionDemo.jsp

出现算术异常错误!

3.3.7在web.xml中配置错误页面

- 声明一个通用的处理页面：

```
<error-page>
```

```
    <exception-type>
```

```
        java.lang.Throwable
```

```
    </exception-type>
```

```
    <location>/error/errorPage.jsp</location>
```

```
</error-page>
```

- Throwable类是所有异常类的根类，因此对没有明确指定错误处理页面的异常，都将由该页面处理。

3.3.7在web.xml中配置错误页面

- 为HTTP的状态码404配置一个错误处理页面:

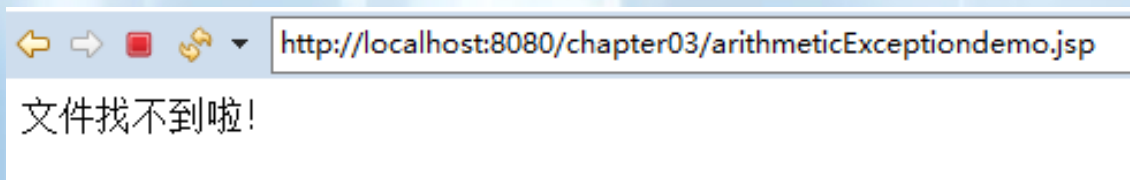
```
<error-page>
```

```
<error-code>404</error-code>
```

```
<location>/error/notFoundError.jsp</location>
```

```
</error-page>
```

- `<location>`元素的值必须以“/”开头，它是相对于Web应用的上下文根目录。另外，如果在JSP页面中使用page指令的errorPage属性指定了错误处理页面，则errorPage属性指定的页面优先。



3.3.8 language属性

- language属性指定在页面的声明、小脚本及表达式中使用的语言，默认值是java。

```
<%@ page language="java" %>
```

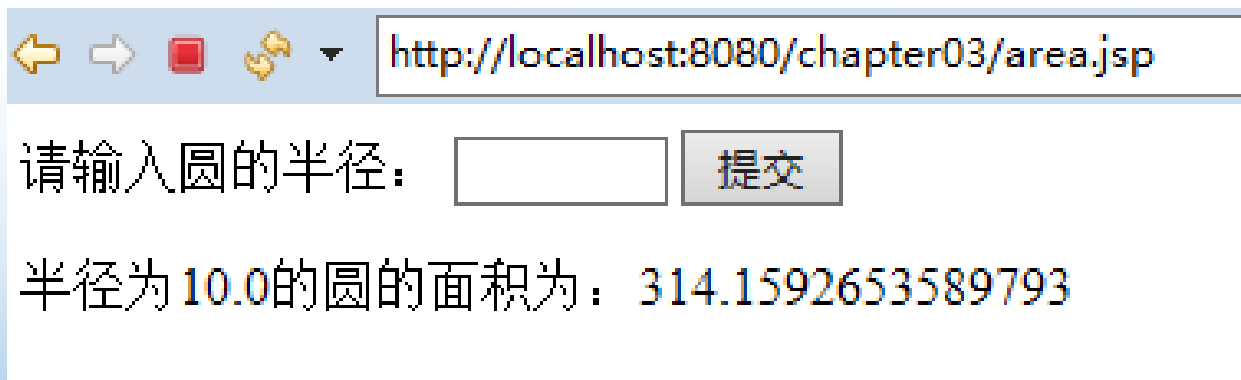
3.4 JSP脚本元素

- 声明、小脚本和表达式允许在页面中编写脚本语言代码，这些元素统称为脚本元素。
- JSP页面使用Java语言作为脚本语言，因此脚本元素中代码的编译和运行行为受到Java编程语言的控制。

3.4.1 变量的声明及顺序

1. 声明的顺序

- 在JSP页面的声明中定义的变量和方法都变成产生的Servlet类的成员，因此它们在页面中出现的顺序无关紧要。
- 在程序[area.jsp](#)中，尽管半径 r 、求面积的方法`area()`是在使用之后定义的，但页面仍然能够转换、编译和运行。



← → □ 🔍 ▼ <http://localhost:8080/chapter03/area.jsp>

请输入圆的半径:

半径为10.0的圆的面积为: 314.1592653589793

2. 小脚本的顺序

- 小脚本被转换成页面实现类的 `_jspService()` 的一部分，小脚本中声明的变量成为该方法局部变量，故它们出现的顺序很重要。

```
<% String s = s1 + s2; %>
```

```
<%! String s1 = "hello"; %>
```

```
<% String s2 = "world"; %>
```

```
<% out.print(s); %>
```

- 该例中，`s1`是在声明中定义的，它成为页面实现类的成员变量，`s`与`s2`是在小脚本中声明的，它们成为 `_jspService()` 的局部变量。
- `s2`在声明之前使用，因此该代码将不能被编译。

3. 变量的初始化

- 在Java语言中，实例变量被自动初始化为默认值，而局部变量使用之前必须明确赋值。
- 在JSP声明中声明的变量被初始化为默认值，而在JSP小脚本中声明的变量使用之前必须明确初始化。

<%! int i; %>

<% int j; %>

The value of i is <%= ++i %>

The value of j is <%= ++ j %>

- 变量i是使用声明（<%!...%>）声明的，它成为产生的Servlet类的实例变量并被初始化为0。变量j是使用小脚本（<%...%>）声明的，它变成产生的_jspService()的局部变量并没有被初始化。

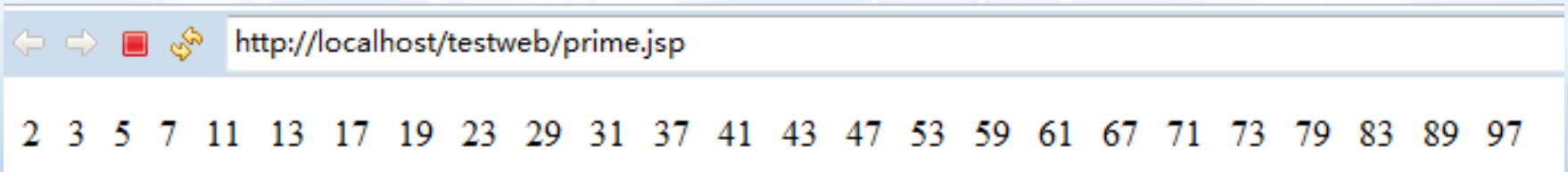
3.4.2 使用条件和循环语句

- 小脚本用来在JSP页面中嵌入计算逻辑，通常这种逻辑包含条件和循环语句。
- 下面的脚本代码使用了条件语句检查用户的登录状态，并基于该状态显示适当的消息。

```
<%  
    String username = request.getParameter("username");  
    String password = request.getParameter("password");  
    boolean isLoggedIn = false;  
    if(username.equals("admin")&&password.equals("admin"))  
        isLoggedIn = true;  
    else  
        isLoggedIn = false;  
    if(isLoggedIn)  
        out.print("<h3>欢迎你, "+username+"访问该页面! </h3>");  
    else{  
        out.println("你还没有登录! <br>");  
        out.println("<a href='login.jsp'>登录</a>");  
    }  
%>
```

3.4.2 使用条件和循环语句

- 例[prime.jsp](#): 使用循环计算并输出100以内的素数

[illegible]

3.4.2 使用条件和循环语句

- 如果在条件语句中包含大量HTML代码，可以使条件语句跨越JSP页面多个小脚本，从而避免书写多个out.println()语句。

```
<%  
    if (isLoggedIn){  
%>  
<h3>欢迎你, <%=username%> 访问该页面! </h3>  
    这里可包含其他HTML代码!  
%>  
    }else{  
%>  
你还没有登录! <br>  
    <a href="login.jsp">登录</a>  
    这里可包含其他HTML代码!  
%>  
    }  
%>
```

3.4.3 请求时属性表达式的使用

- JSP表达式并不总是写到页面的输出流中，它们也可以用来向JSP动作传递属性值，例如：

```
<%! String pageURL = "copyright.jsp"; %>
```

```
<jsp:include page="<%= pageURL %>" />
```

- JSP表达式<%= pageURL %>的值并不发送到输出流，而是在请求时计算出该值，然后将它赋给<jsp:include>动作的page属性。
- 以这种方式向动作传递一个属性值使用的表达式称为请求时属性表达式。

3.4.3 请求时属性表达式的使用

- 请求时属性表达式不能用在指令的属性中，因为指令具有转换时的语义，即容器仅在页面转换期间使用指令。
- 下例中的指令是非法的：

```
<%! String pageURL = "copyright.html"; %>
```

```
<%@ include file="<%= pageURL %>" %>
```

3.5 JSP隐含变量

- 在JSP页面的转换阶段，Web容器在 `_jspService()` 中声明并初始化一些变量，可在JSP页面小脚本中或表达式中直接使用这些变量：

`<%`

`out.print("<h1>Hello World! </h1>");`

`%>`

- `out`对象是由容器隐含声明的，所以一般被称为**隐含对象**（implicit objects），这些对象是由容器创建，可象变量一样使用，因此也被叫做**隐含变量**（implicit variables）。

JSP页面中可使用的隐含变量

隐含变量	类或接口	说明
application	javax.servlet.ServletContext接口	引用Web应用程序上下文
session	javax.servlet.http.HttpSession接口	引用用户会话
request	javax.servlet.http.HttpServletRequest接口	引用页面的当前请求对象
response	javax.servlet.http.HttpServletResponse接口	用来向客户发送一个响应
out	javax.servlet.jsp.JspWriter类	引用页面输出流
page	java.lang.Object类	引用页面的Servlet实例
pageContext	javax.servlet.jsp.PageContext类	引用页面上下文
config	javax.servlet.ServletConfig接口	引用Servlet的配置对象
exception	java.lang.Throwable类	用来处理错误

3.5 JSP隐含变量

- 以todayDate.jsp页面为例，在转换阶段，Web容器自动将该页面转换成一个名为todayDate_jsp.java的类文件，该文件_jspService()中声明了8个变量

```
public void _jspService(final javax.servlet.http.HttpServletRequest request,
    final javax.servlet.http.HttpServletResponse response)
    throws java.io.IOException, javax.servlet.ServletException {
    //其他代码
    final javax.servlet.jsp.PageContext pageContext;
    javax.servlet.http.HttpSession session = null;
    final javax.servlet.ServletContext application;
    final javax.servlet.ServletConfig config;
    javax.servlet.jsp.JspWriter out = null;
    final java.lang.Object page = this;
    javax.servlet.jsp.JspWriter _jspx_out = null;
    javax.servlet.jsp.PageContext _jspx_page_context = null;
    //其他代码
}
```

3.5.1 request与response变量

- `request`是`HttpServletRequest`类型的隐含变量, `response`是`HttpServletResponse`类型的隐含变量, 当页面的`Servlet`向客户提供服务时, 它们作为参数传递给`_jspService()`。
- 在JSP页面中使用它们与在`Servlet`中使用完全一样, 即用来分析请求和发送响应, 如下代码所示。

3.5.1 request与response变量

<%

```
String remoteAddr = request.getRemoteAddr();  
response.setContentType("text/html;charset=UTF-8");
```

%>

你的IP地址为: <%=remoteAddr%>

你的主机名为: <%=request.getRemoteHost()%>

3.5.2 out变量

- out是`javax.servlet.jsp.JspWriter`类型的隐含变量，打印输出所有的基本数据类型、字符串以及用户定义的对象。
- 可以在小脚本中直接使用out，也可在表达式中间使用它产生HTML代码，例如：

```
<% out.print("Hello World!"); %>
```

```
<%= "Hello User!" %>
```

- 对上面两行代码，在页面实现类中都使用`out.print()`语句输出。

3.5.2 out变量

- 下面的脚本使用out变量的print()打印输出不同类型的数据。

```
<% int anInt = 3;  
    Float aFloatObj = new Float(5.6);  
  
    out.print(anInt);  
    out.print(anInt > 0);  
    out.print(anInt*3.5/100-500);  
    out.print(aFloatObj);  
    out.print(aFloatObj.floatValue());  
    out.print(aFloatObj.toString());  
%>
```

3.5.3 application变量

- application是

javax.servlet.ServletContext类型的隐含变量，它是JSP页面所在的Web应用程序的上下文的引用（ServletContext接口），下面两段小脚本是等价的。

```
<%  
    String path = application.getRealPath("/WEB-INF/count.db");  
    application.log("绝对路径为:"+path);  
%>  
  
<%  
    String path = getServletContext().getRealPath("/WEB-INF/count.db");  
    getServletContext().log("绝对路径为:"+path);  
%>
```

3.5.4 session变量

- `session`是
`javax.servlet.http.HttpSession`类型的隐含变量，它在JSP页面中表示会话对象。要使用会话对象，必须要求JSP页面参加HTTP会话，即要求将JSP页面的`page`指令的`session`属性值设置为`true`。
- 默认情况下，`session`属性的值为`true`。如果明确将`session`属性设置为`false`，容器将不会声明该变量，对该变量的使用将产生错误，如下所示。

```
<%@ page session = "false" %>
<html><body>
    会话ID = <%=session.getId()%>
</body></html>
```

3.5.5 pageContext变量

- `pageContext`是`javax.servlet.jsp.PageContext`类型的隐含变量，它是一个页面上下文对象。
- `PageContext`类是一个抽象类，容器厂商提供了一个具体子类（如`JspContext`），有下面三个作用：
 - （1）存储隐含对象的引用。`pageContext`对象是作为管理所有在JSP页面中使用的其他对象的一个地方，包括用户定义的和隐含的对象，并且它提供了一个访问方法来检索它们。
 - （2）提供了在不同作用域内返回或设置属性的方便的方法。
 - （3）提供了`forward()`和`include()`实现将请求转发到另一个资源和将一个资源的输出包含到当前页面中的功能

3.5.5 pageContext变量

- `public void include(String relativeURL)`
 - ✓ 将另一个资源的输出包含在当前页面的输出中，与 `RequestDispatcher()` 接口的 `include()` 功能相同。
- `public void forward(String relativeURL)`
 - ✓ 将请求转发到参数指定的资源，与 `RequestDispatcher` 接口的 `forward()` 功能相同。

3.5.5 pageContext变量

- 从Servlet中将请求转发到另一个资源：

```
RequestDispatcher rd =  
request.getRequestDispatcher("other.jsp");  
rd.forward(request, response);
```

- 在JSP页面中，使用pageContext变量将请求转发到另一个资源：

```
pageContext.forward("other.jsp");
```

3.5.6 page变量

- `page`变量是`java.lang.Object`类型的对象，声明如下

```
Object page = this;
```

- 生成的Servlet实例，该变量很少被使用。

3.5.7 config变量

- `config`是`javax.servlet.ServletConfig`类型的隐含变量。可通过`web.xml`文件为Servlet传递一组初始化参数，然后在Servlet中使用`ServletConfig`对象检索这些参数。
- 可以为JSP页面传递一组初始化参数，这些参数在JSP页面中可以使用`config`隐含变量来检索。
- 在`web.xml`中使用`<servlet-name>`声明一个Servlet，然后使用`<jsp-file>`元素使其与JSP文件关联。

3.5.7 config变量

```
<servlet>
  <servlet-name>initTestServlet</servlet-name>
  <jsp-file>/initTest.jsp</jsp-file>
  <init-param>
    <param-name>company</param-name>
    <param-value>Baidu CO., LTD</param-value>
  </init-param>
  <init-param>
    <param-name>email</param-name>
    <param-value>zxm@zjut.edu.cn</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>initTestServlet</servlet-name>
  <url-pattern>/initTest.jsp</url-pattern>
</servlet-mapping>
```

3.5.7 config变量

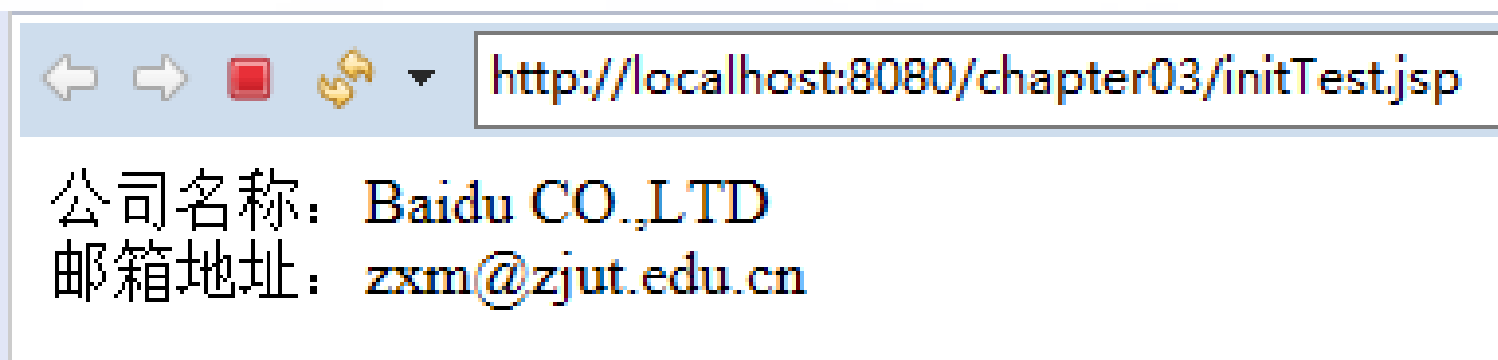
- 在initTest.jsp文件中使用隐含变量config检索到web.xml的参数。

公司名称:

```
<%=config.getInitParameter("company") %><br>
```

邮箱地址:

```
<%=config.getInitParameter("email") %>
```



3.5.8 exception变量

- exception是java.lang.Throwable类型的隐含变量，它被用来作为其他页面的错误处理器。
- 为使页面能够使用exception变量，必须在page指令中将isErrorPage的属性值设置为true。

程序errorPage.jsp

```
<%@ page isErrorPage='true' %>
<%@ page contentType="text/html;charset=utf-8"%>
<html><body>
页面发生了下面错误: <%=exception.toString()%>
</body></html>
```

小 结

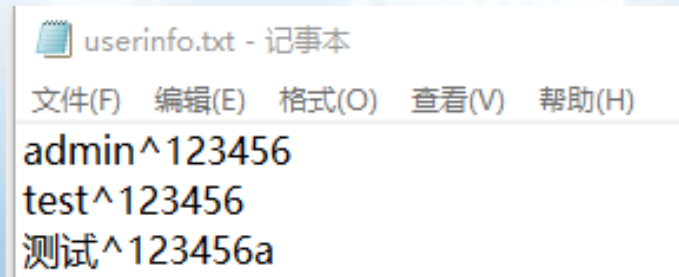
- page指令用于告诉容器关于JSP页面的总体特性，该指令适用于整个转换单元而不仅仅是它所声明的页面。

```
<%@ page contentType="text/html;  
    charset=UTF-8" pageEncoding="UTF-8"%>
```

- 声明、小脚本和表达式允许在页面中编写脚本语言代码，这些元素统称为脚本元素。
- JSP页面中有9个隐含变量：application、session、request、response、page、pageContext、out、config和exception等。

练习

- 1、假设用户登录页面login.jsp在web工程web2020的hw目录下，即访问login.jsp的URL地址是
`http://localhost:8080/web2020/hw/login.jsp`，将login.jsp提交的用户名和密码由FirstServlet（映射地址为/servlet/firstServlet.do）获取进行判断，如果登录成功，则跳转至SecondServlet（映射地址为/servlet/secondServlet.do）显示用户名信息，如果登录不成功，则跳转至登录失败页面hw/failed.jsp。
- 要求：用户名和密码存在WEB-INF/userinfo.txt文件中，若页面出现404错误则统一跳转至404.html页面，500错误则统一跳转至500.html页面，并提示出错信息。



A screenshot of a Notepad window titled "userinfo.txt - 记事本". The window contains three lines of text: "admin^123456", "test^123456", and "测试^123456a". The text is in a standard monospaced font.

```
admin^123456
test^123456
测试^123456a
```