第3章 JSP语法基础

本章要点：

- JSP 页面结构；
- JSP 语法基础；
- JSP 内置对象。

JSP 是一种运行在服务器端的脚本语言，JSP 页面又是基于 HTML 网页的程序，它是 Java Web 开发技术的基础。本章介绍 JSP 页面的结构，包括 JSP 指令、脚本元素、注释和动作以及 JSP 内置对象等内容。

3.1 JSP 页面的基本结构

在传统的 HTML 页面中加入 Java 的程序段和 JSP 标签就构成了一个 JSP 页面。一个 JSP 页面可以由以下 5 个部分组成。

- HTML 元素；
- 注释：包括 JSP 注释和 HTML 注释；
- 脚本元素：声明、表达式、脚本片段等；
- 指令：包括 page 指令、taglib 指令、include 指令等；
- 动作：包括<jsp:useBean>、<jsp:forward>、<jsp:include>等动作标记。

3.1.1 JSP 注释

JSP 的注释内容放在“<%--”与“--%>”中间，其语法格式如下：

```
<% --此处为注释内容 -- %>
```

与 HTML 注释不同的是，JSP 引擎忽略 JSP 注释，用户在客户端看不到 JSP 注释，因此又把 JSP 注释称为隐藏注释。

由于 HTML 注释在客户端是可见的，因此也把 HTML 注释称为输出注释。

3.1.2 脚本元素

在 JSP 页面中可以加入一些 Java 程序段，还可以声明变量、表达式等。

1. 声明变量

在JSP页面中，把要声明的变量放置在标记符“<%!”与“%>”之间即可，变量的类型可以是Java允许的任何类型。其语法格式如下：

```
<%! 声明语句; %>
```

例如下面的代码声明一个整型变量 *i* 和一个Date类型的变量。

```
<%!
    int i = 0;
    java.util.Date date = new java.util.Date();
%>
```

在“<%!”与“%>”之间声明的变量相当于Java类中的成员变量。

2. 声明方法

同理，在JSP页面中也可以把声明方法的语句放在“<%!”与“%>”之间，例如下面是一个声明求两个整数之和的方法。

```
<%!
int add(int opt1,int opt2)
{
    return opt1 + opt2;
}
%>
```

注意：

使用标记符“<%!”与“%>”声明的变量和方法是页面级的，即它们在声明语句所在的页面有效。因为Tomcat服务器将JSP页面转换为Java类时，声明的变量将作为类的成员变量，而声明的方法即为类的方法。成员变量在执行过程中是被所有用户共享使用的，即多个用户访问该JSP页面，当用户改变该页面中成员变量的值时将影响其他用户使用此成员变量。

3.1.3 JSP页面中的表达式

在JSP页面中可以将一个Java表达式放在<%=与%>之间，表达式在Tomcat服务器运算后将结果转换为字符串，并且输出到JSP页面中，其语法格式如下：

```
<% =表达式 %>
```

需要注意的是，在表达式后面不需要加分号(;)，而且“<%=”是一个整体，各字符之间不能有空格。

3.1.4 JSP 页面中的 Java 程序段

用户可以在“<%”与“%>”之间插入 Java 程序段，其语法格式如下：

```
<%    Java 程序段   %>
```

当 JSP 页面被客户请求时，该 JSP 页面中的 Java 程序段就会被执行。JSP 页面会被转换为 Java 类(即 Servlet)，而 Java 程序段将被放置在 Servlet 的 service()方法中。Java 程序段可以包含多个 JSP 语句，也可以声明变量等。一个 JSP 页面可以有多个 Java 程序段，这些程序段按先后顺序执行。

注意：

Java 程序段中声明的变量是局部变量，该局部变量在 JSP 页面后继的所有程序段及表达式中均有效。另外，不同用户在访问 JSP 页面中相同名称的局部变量时，这些局部变量互不影响，即一个用户改变 Java 程序段中的局部变量值不会影响其他用户的 Java 程序段中的局部变量。

下面的程序段是一个 for 循环。

```
<%
    for(int i = 1; i<6; i++)
    {
        out.println("打印了" + i + "次<br>");
    }
 %>
```

3.1.5 JSP 指令

JSP 指令主要有 page、include 和 taglib，JSP 指令负责提供 JSP 页面的相关信息以及设置 JSP 页面的属性等。

1. page 指令

page 指令用来设置 JSP 页面的属性，其语法格式如下：

```
<%@ page language = "java"
      contentType = "MIMETpye; charset = characterSet"
      pageEncoding = "characterSet"
      import = "package.class"
      extends = "package.class"
      buffer = "none|size kb|8kb"
      errorPage = "URL"
      autoFlush = "false|true"
      session = "false|true"
      isThreadSafe = "false|true"
      isErrorPage = "true|false"
      isELIgnored = "true|false"
      %>
```

page 指令设置的 JSP 页面属性如表 3.1 所示。

表 3.1 page 指令设置的 JSP 页面属性

属性名称	说明
language	声明该 JSP 页面脚本语言的名称，目前只能为 java
contentType	声明该 JSP 页面的 MIME 类型和字符编码集，默认值为“text/html;charset=iso-8859-1”
pageEncoding	设定 JSP 页面的字符编码，默认值为 iso-8859-1
import	导入该 JSP 页面所使用的 Java API，若用到多个 Java API，中间用逗号隔开
extends	定义此 JSP 页面产生的 Servlet 继承自哪个父类，该父类必须为实现 javax.servlet.jsp.HttpJspPage 接口的类，在一般情况下不需要进行设置，默认父类为 HttpJspBase
buffer	设定输出流缓存的大小，默认为 8kb
errorPage	指定该 JSP 页面发生错误时网页被重定向指向的错误处理页面
autoFlush	指定输出流缓存区的内容是否自动清除，默认为 true
session	指定该 JSP 页面是否需要一个 HTTP 会话，默认为 true
isThreadSafe	指定该 JSP 页面是否支持多个用户同时请求(即多线程同步请求)，默认为 true
isErrorPage	指定该 JSP 页面是否为错误处理页面，默认为 false
isELIgnored	指定是否忽略 EL 表达式，默认为 false
info	该属性可设置为任意字符串，例如当前页面的作者或其他有关的页面信息，可通过 Servlet.getServletInfo()方法获取设置的字符串

无论 page 指令出现在 JSP 页面的哪个位置，page 指令出现多少次都没有限制，但是一般把 page 指令放在 JSP 页面的顶部，而且 page 指令中的诸多属性最多只能出现一次(import 属性除外)，否则程序将报错。

注意：

当 JSP 页面中包含中文时，需要把 contentType 和 pageEncoding 属性中的字符集设定为中文字符集编码，例如 GB2312、GB18030、GBK、UTF-8 等，推荐使用 UTF-8。字符集名称中的字母是不区分大小写的，例如 gbk 和 GBK 具有相同的效果。

example3_1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" import="java.util.Date" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_1</title>
</head>
<body>
<% Date date = new Date(); %>
当前的系统日期为：<%= date %>
<br>
<%
for(int i=1;i<6;i++)
    out.print("打印了" + i + "次<br>");
%>
</body>
</html>
```

运行结果如图 3.1 所示。

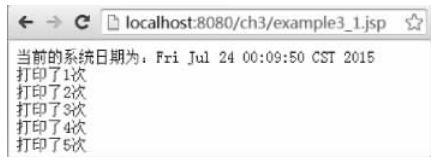

图 3.1 example3_1.jsp 的运行结果

2. include 指令

include 指令是页面包含指令，在 JSP 页面中可以使用 include 指令包含另一个文件。包含的文件可以是 HTML 页面，也可以是 JSP 页面甚至是普通文本文件，其语法格式如下：

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

include 指令只有一个属性，即 file 属性，file 属性值是一个包含文件的 URL。include 指令将会在 JSP 页面编译时插入包含的文件，它是静态的。

下面的例子是将文件 sub.jsp 包含在 JSP 页面 example3_2.jsp 中。

主页面 example3_2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_2</title>
</head>
<body>
<%@ include file="sub.jsp" %>
----------------------------------------<br>
这是主文件
</body>
</html>
```

包含文件 sub.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>包含页面 sub</title>
```

```
</head>
<body>
这是包含文件<br>
</body>
</html>
```

运行 example3_2.jsp，结果如图 3.2 所示。

图 3.2　example3_2.jsp 的运行结果

从本例中可以看出，在运行 example3_2.jsp 页面后，sub.jsp 页面被插入到 example3_2.jsp 的 include 指令代码处。注意这两个 JSP 页面 page 指令的 contentType 属性值务必一致，否则将导致错误。

注意：

如果主页面与包含文件不在同一个目录，包含文件中引用的图片等资源位置要以主页面的相对路径为依据，否则包含文件中的这些引用资源在主页面上无法正常显示。

3. taglib 指令

taglib 指令的作用是指定该 JSP 页面使用自定义标签，使 JSP 页面更加个性化，其语法格式如下：

```
<%@ taglib uri = "tagURI" prefix = "prefix" %>
```

可以看到，taglib 指令只有两个属性，具体说明如下。

- uri：指定自定义标签文件的路径，可以是绝对路径或者相对路径，也可以是标签库的描述文件；
- prefix：指定自定义标签的前缀，注意前缀名称不能使用保留字，例如 java、javax、jsp、servlet、sun 等。

在使用 taglib 指令时，首先应该定义标签文件，或者使用第三方标签库（确认第三方标签库的 URI），然后在 JSP 页面中使用 taglib 指令引用该标签文件，这样就可以在该 JSP 页面中使用自定义标签了。

```
<!-- 使用 taglib 指令引用 Struts2 标签库，前缀为 s -->
<%@ taglib uri = "/struts - tags" prefix = "s" %>
<!-- 使用 property 标签 -->
<s:property value = "user.name" />
```

上面的代码使用了 Struts 2 标签库中的＜s:property＞标签，Struts2 标签前缀为“s”，同时指定了该标签库文件的路径为“/struts-tags”。

3.1.6 JSP 动作

JSP 动作标记有 20 多种，本节重点介绍<jsp:include>、<jsp:param>、<jsp:forward>、<jsp:useBean>、<jsp:setProperty>和<jsp:getProperty>6 种动作标记。

1. <jsp:include>动作标记

<jsp:include>动作标记的作用是将一个指定的页面包含到使用此动作标记的 JSP 页面中，<jsp:include>动作标记的语法格式有下面两种。

方式一：

```
<jsp:include page="文件的URL | <%=表达式%>" flush="true" />
```

方式二：

```
<jsp:include page="文件的URL | <%=表达式%>" flush="true">
嵌套的子标记
</jsp:include>
```

该动作标记各属性的含义如下。

- page：指定包含页面的相对路径(URL)，或者是表示相对路径的表达式；
- flush：如果使用 flush 属性，若该属性值为 true，表示缓存将会被清空。在 JSP 1.2 中，flush 的默认值为 false。

如果需要传递参数，可以使用<jsp:param>动作标记作为子标记嵌套在<jsp:include>标记中，此时应该使用上述的第二种语法形式；否则，当<jsp:include>不需要子标记时必须使用上述的第一种形式。

注意：

<jsp:include>动作标记和 include 指令标记的作用非常类似，它们的主要区别如下。

include 指令是静态包含，执行时间是在编译阶段，引入的内容为静态文件，在编译为 Servlet 时和主页面融合在一起(注意只是将两个页面按照 include 指令的位置简单地合并在一起)，而且 file 属性值不能是一个变量，也不能传递参数。

<jsp:include>动作标记是动态包含的，执行时间是在请求处理阶段，引入的内容在执行页面被请求时动态生成后再包含到页面中。另外读者要注意，在书写此标记时“jsp”“:”和“include”之间不能有空格。

主页面 example3_3.jsp

```
⋮
<body>
以下是包含文件 include.jsp 中的内容：<br>
----------<br>
<jsp:include page="include.jsp" flush="true"/>
<br>
```

```
以下为主文件：<br>
-------------------------------------------<br>
测试 include 动作标记的用法
<!-- 我们在这里用 include 的两种不同形式来引入 date.jsp 这个文件. -->
</body>
⋮
```

包含文件 include.jsp

```
<%@ page language="java" import="java.util.*"
contentType="text/html;charset=utf-8"%>
<%
Date date=new Date();
Calendar cal = Calendar.getInstance();
cal.setTime(date);
String date_cn="";
String dateStr="";
switch(cal.get(Calendar.DAY_OF_WEEK))
{
case 1:date_cn="日";break;
case 2:date_cn="一";break;
case 3:date_cn="二";break;
case 4:date_cn="三";break;
case 5:date_cn="四";break;
case 6:date_cn="五";break;
case 7:date_cn="六";break;
}
dateStr = cal.get(Calendar.YEAR)+"年"+(cal.get(Calendar.MONTH)+1)+"月"+cal.get
(Calendar.DAY_OF_MONTH)+
"日(星期"+date_cn+")";
out.print(dateStr);
%>
```

运行结果如图 3.3 所示。

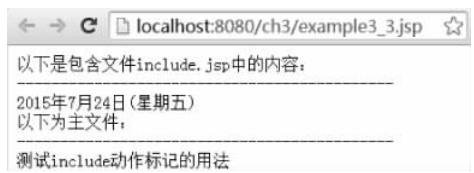

图 3.3 example3.3.jsp 的运行结果

2. <jsp:param>动作标记

<jsp:param>动作标记主要用来传递参数，其语法格式如下：

```
<jsp:param name="参数名称" value="参数值"/>
```

其中参数的作用如下。

- name：表示传递参数的名称；
- value：表示传递参数的值。

＜jsp:param＞动作标记不能单独使用，一般嵌套在＜jsp:include＞、＜jsp:forward＞等动作标记内，用于向这些动作标记传递参数。

主页面 example3_4.jsp

```
⋮
<body>
测试 param 动作标记的用法,以下是包含 calculate 的内容:<br>
<!-- 向 calculate.jsp 传递两个参数 opt1,opt2 -->
<jsp:include page="calculate.jsp">
    <jsp:param name="opt1" value="50" />
    <jsp:param name="opt2" value="100" />
</jsp:include>
</body>
⋮
```

包含文件 calculate.jsp

```
⋮
<body>
<%!
int sum(int opt1,int opt2)
{
    return opt1 + opt2;
}
int sumVal = 0;
%>
<%
    //使用 request 对象的 getparameter()方法获取<jsp:param>参数值
    int opt1 = Integer.parseInt(request.getParameter("opt1"));
    int opt2 = Integer.parseInt(request.getParameter("opt2"));
    sumVal = sum(opt1 , opt2);
    out.print(opt1 + " + " +  opt2 + " = " + sumVal);
%>
</body>
```

运行结果如图 3.4 所示。

图 3.4 example3_4.jsp 的运行结果

3. ＜jsp:forward＞动作标记

＜jsp:forward＞动作标记的作用是页面重定向，即跳转至 page 属性指定的页面，该页

面可以是一个 HTML 页面、JSP 页面，甚至是一个程序段，其语法格式有下面两种。

方式一：

```
<jsp:forward page = "跳转页面的URL|<% = 表达式 %>" />
```

方式二：

```
<jsp:forward page = "跳转页面的URL|<% = 表达式 %>">
    <jsp:param value = "参数值" name = "参数名" />
</jsp:forward>
```

<jsp:forward>只有一个 page 属性，用于设定跳转页面的 URL，也可以是一个表达式。如果要向跳转页面传递参数，可以使用第二种形式，利用<jsp:param>作为子标记传递参数。在下面的例子中，客户端请求 example3_5.jsp 页面后重定向到另一个 JSP 页面 forward.jsp。

example3_5.jsp

```
⋮
<body>
随机产生一个成绩(0～100)，判断其结果是否及格。
<%
request.setCharacterEncoding("utf-8");
int r = (int)(Math.random() * 100);
if(r >= 60)
{
 %>
 <jsp:forward page = "forward.jsp">
      <jsp:param value = "<% = r %>" name = "score" />
      <jsp:param value = "恭喜，你及格了！" name = "result" />
 </jsp:forward>
 <%
 }
 else
 {
 %>
  <jsp:forward page = "forward.jsp">
       <jsp:param value = "<% = r %>" name = "score" />
       <jsp:param value = "再接再励哦！" name = "result" />
  </jsp:forward>
  </jsp:forward>
  <%
  }
  out.println("本页面结束，但是看不到此行代码");
   %>
</body>
⋮
```

跳转页面 forward.jsp

```
⋮
<body>
你的成绩为：<% = request.getParameter("score") %>
</body>
⋮
```

该程序的某次运行结果如图 3.5 所示。

图 3.5 example3_5.jsp 的运行结果

在本例中，使用 Math 类的 random()方法产生一个 0～100 的随机数成绩，使用＜jsp:forward＞动作标记跳转到 forward.jsp 页面，并且根据成绩是否及格传递不同的参数。

注意：

在使用＜jsp:forward＞动作标记跳转页面时，其 URL 并不会随之改变为跳转后的页面地址，仍是跳转前的 URL。此外，一旦执行了＜jsp:forward＞动作标记，那么当前页面的后续代码将停止执行，例如 example3_5.jsp 页面中＜jsp:forward＞标记后的代码将不会被执行，并且当刷新页面的时候会导致重复提交。

4. ＜jsp:useBean＞动作标记

＜jsp:useBean＞、＜jsp:setProperty＞和＜jsp:getProperty＞这 3 个动作标记均与 JavaBean 有关。JavaBean 是一个可重复使用的软件组件，实际上 JavaBean 是一种特殊的类，通过封装属性和方法成为具有某种功能或者处理某个业务的对象。JavaBean 可以实现代码复用，具有易用、平台无关等特性，从而实现业务逻辑与表现层的分离。对于 JavaBean 将在本书第 5 章介绍，本小节仅介绍＜jsp:useBean＞动作标记。＜jsp:useBean＞的语法格式也有两种。

方式一：

```
<jsp:useBean id = "bean 的名字" class = "引用bean 的类" scope = "bean 的作用域">
</jsp:useBean>
```

方式二：

```
<jsp:useBean id = "bean 的名字" class = "引用bean 的类" scope = "bean 的作用域" />
```

其中各属性的含义如下。

(1) id：引用的 JavaBean 在所定义的作用域内的名称，在此作用域内使用该 id 就代表所引用的 JavaBean。注意 id 值的大小写，Java 是严格区分大小写的。

(2) class：所引用 JavaBean 的完整包路径，一般格式为“package.class”。

(3) scope：指定该 JavaBean 的作用域以及 id 变量名的有效范围，其取值可以为 page、

request、session、application，默认值为 page。

- page：其作用在当前页面有效，当用户离开此页面时 JavaBean 无效。不同用户访问同一个页面且作用域为 page 的 JavaBean 时，两个用户的 JavaBean 的取值是互不影响的，即一个用户改变自己的 JavaBean 属性不会影响其他用户。
- request：作用在用户的请求期间有效，用户在访问 Web 网站期间可能会请求多个页面，如果这些页面有取值范围为 request 的 JavaBean 引用，那么在每个页面分配的 JavaBean 也是互不影响的；当 Web 服务器对该请求做出响应之后，该 JavaBean 无效。
- session：其作用在用户的会话期间有效，即用户在多个页面间相互连接，每个页面都含有一个<jsp:useBean>动作标记，而且各个页面间的 id 值和 scope 值均相同，那么用户在这些页面得到的 JavaBean 实际上是同一个。如果用户在某个页面改变了 JavaBean 的属性，会影响到其他页面的 JavaBean。
- application：其作用范围是整个 Web 应用，对于同一 id 名称的 JavaBean，此时在 Web 应用的每一个页面都共享使用同一个 JavaBean，不同用户访问的也是同一个 JavaBean。当某个用户改变 JavaBean 的属性时也会影响其他用户对该 JavaBean 的使用。

5. <jsp:setProperty>动作标记

<jsp:setProperty>动作标记通常与<jsp:useBean>动作标记一起使用，使用<jsp:setProperty>动作标记设置 JavaBean 属性的值，其语法格式如下：

```
<jsp:setProperty name="useBean 标记中属性id 的值" property="* | JavaBean 的属性名" value
="JavaBean 属性值|<%=表达式 %>" />
```

其属性的含义如下。

- name：其值应为该页面中的<jsp:useBean>动作标记中引用 JavaBean 的 id 值。
- property：当值为“*”时，表示存储用户在 JSP 页面中输入的所有值，并自动匹配 JavaBean 的属性；当值为某一具体的属性时，表示 JavaBean 中的一个具体的属性名。
- value：为 JavaBean 中某一个具体的属性赋值，可以是一个字符串，或者是一个表达式。

6. <jsp:getProperty>动作标记

<jsp:getProperty>动作标记也与<jsp:useBean>动作标记一起使用，使用<jsp:getProperty>动作标记获取 JavaBean 中指定属性的值，其语法格式如下：

```
<jsp:getProperty name="useBean 标记中属性id 的值" property="JavaBean 的属性名" />
```

其属性的含义如下。

- name：其值应为该页面中的<jsp:useBean>动作标记中引用 JavaBean 的 id 值。

• property：指定要获取的 JavaBean 的某一具体属性名。

动手实践 3-1

现有 3 个页面，即 top.jsp、main.jsp 和 foot.jsp。其中 main.jsp 为主页面，在其顶部将 top.jsp 包含到该页面中，在其底部将 foot.jsp 包含到该页面中，根据前面介绍的 JSP 指令和动作标记应该如何实现？

3.2 JSP 内置对象

所谓 JSP 内置对象，就是不需要声明就可以在 JSP 页面中直接使用的对象。JSP 提供了 9 种内置对象，如表 3.2 所示。

表 3.2 JSP 的内置对象

内置对象	类型	作用域
request	javax.servlet.HttpServletRequest	request
response	javax.servlet.HttpServletResponse	page
page	java.lang.Object(相当于 this 关键字)	page
pageContext	javax.servlet.jsp.PageContext	page
session	javax.servlet.http.HttpSession	session
application	javax.servlet.ServletContext	application
out	javax.servlet.jsp.JspWriter	page
config	javax.servlet.ServletConfig	page
exception	java.lang.Throwable	page

从该表中可以看到，内置对象有 4 种作用域，分别是 page、request、session 和 application，且作用范围依次增大。

学习内置对象，关键在于掌握这些内置对象的方法和属性以及这些内置对象的作用。

- request：表示 HTTP 协议的请求，提供对请求数据的访问，JSP 页面可以在请求范围内共享数据。
- response：表示 HTTP 协议的响应，提供了访问响应报文的相关方法。
- page：代表 JSP 页面对应的 Servlet 实例。
- pageContext：表示 JSP 页面本身的上下文，它提供了一组方法用于管理具有不同作用域的属性。
- session：表示 HTTP 协议的会话，可以共享服务器与浏览器之间的会话数据，一旦关闭了浏览器，会话数据将自动销毁。
- application：代表应用程序上下文，允许 JSP 页面与同一应用程序中的 Web 组件共享数据。
- out：提供对输出流的访问。
- config：提供了一组方法访问 Web 应用程序的配置文件 web.xml。
- exception：表示异常对象，该对象含有特定 JSP 异常处理页面访问的异常信息。

3.3 request 对象

当客户端向 Web 服务器发送请求获取某种资源时，相当于向 Web 服务器发送了一个 HTTP 请求(request)。一个 HTTP 请求报文一般包括 4 部分，即请求行(Request Line)、请求首部(Header)、空行(Blank Line)和请求数据(Body)等。其中请求行由请求方法字段、URL 字段和 HTTP 协议版本字段 3 个字段组成，它们用空格分隔。请求方式可以是 GET、POST、PUT、TRACE、HEAD、OPTIONS 和 CONNECT。HTTP 请求首部是指客户端传递请求的附加信息和客户端把自己的附加信息给服务器的内容，一个 HTTP 请求首部可以包括 Accept、Accept-Language、Accept-Encoding、Authorization、From、Host、Range、Referer、User-Agent、TE 等。下面是一个请求方式为 GET 的 HTTP 请求报文。

```
GET / HTTP/1.1
Accept: */*
Accept-Language: zh-cn
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET
CLR 3.0.04506.648; .NET CLR 3.5.21022)
Host: www.henu.edu.cn
Connection: Keep-Alive
```

该 GET 请求报文的第一部分说明了该请求是一个 GET 请求，之后是一个斜线(/)，用来说明请求的是该域名的根目录，该行的最后说明使用的是 HTTP1.1 版本(也可选为 HTTP1.0)。

第二部分是请求的第一个首部，Host 指出请求的目的地为 www.henu.edu.cn。User-Agent 表示服务器端和客户端脚本都能访问它，它是浏览器类型检测逻辑的重要基础，该信息由客户端的浏览器来定义，并且在每个请求中自动发送。Connection 通常将浏览器操作设置为 Keep-Alive。

第三部分是空行，即使不存在请求主体此空行也是必需的。

request 对象主要用来获取客户端的请求信息，以获取通过 HTTP 协议传送给服务器端的数据，其中包括头信息(Header)、请求方式(get 或 post)以及客户端的其他信息。request 对象的主要方法如表 3.3 所示。

表 3.3 request 对象的主要方法

方　法	说　明
Object getAttribute(String name)	获得 name 的属性值，若不存在，则返回 null
Enumeration getAttributeNames()	返回一个枚举类型的包含 request 对象所有属性名称的集合
String getCharacterEncoding()	返回 request 请求体的字符编码
int getContentLength()	获得 HTTP 请求的长度
String getContentType()	获得客户端请求的 MIME 类型
String getContextPath()	获得上下文的路径，即当前 Web 应用的根目录
String getHeader(String name)	获得 HTTP 协议的文件头信息

续表

方 法	说 明
ServletInputStream getInputStream()	得到请求体中一行的二进制流
String getMethod()	获得客户端请求的方法类型,一般为 GET、POST 等
String getParameter(String name)	获得指定参数 name 的参数值
Enumeration getParameterNames()	返回一个枚举类型的所有参数名称的集合
String[] getParameterValues(String name)	返回包含参数 name 的所有值的数组
String getProtocol()	返回请求所使用的协议及其版本
String getQueryString()	获得查询字符串,该字符串在客户端以 GET 方式向服务器传送
BufferedReader getReader()	以字符码的形式返回请求体
String getRemoteAddr()	返回客户端的 IP 地址
String getRemoteHost()	返回客户端的主机名
String getScheme()	返回请求所用的协议名称,例如 HTTP、HTTPS、FTP 等
String getServerName()	获得服务器的名称,若没有设定服务器域名,则返回其 IP 地址
int getServerPort()	返回服务器的端口号
String getServletPath()	获得请求 JSP 页面的名称
boolean getSession()	返回和当前客户端请求相关联的 HttpSession 对象
boolean isSecure()	判断客户机是否以安全的访问方式访问服务器
void removeAttribute(String name)	删除名称为 name 的 request 参数
void setAttribute(String name,Object obj)	设置一个名称为 name 的参数,并且其值为 obj
void setCharacterEncoding(String enc)	设置请求信息的字符编码为 enc

下面的例子演示 request 内置对象的一些方法的使用,本例包含两个页面,即 example3_6.jsp 和 result.jsp。

第一步:创建 example3_6.jsp 页面,并添加一个表单,表单的 action 属性值为 result.jsp。

example3_6.jsp

```
<form action = "result.jsp" method = "post">
请输入内容:<input type = "text" name = "param" /><br>
<input type = "submit" value = "确定" /></form>
```

第二步:创建 result.jsp。

result.jsp

```
<body>
<%
    //设置请求报文的字符编码为 UTF-8,避免中文字符发生乱码
    request.setCharacterEncoding("utf-8");
%>
从 example3_6.jsp 页面中传过来的值为:
<% = request.getParameter("param") %><br>
客户端的 IP 地址为:<% = request.getRemoteAddr() %><br>
```

```
客户端的主机名为：<% = request.getRemoteHost() %><br>
客户端的端口号为：<% = request.getRemotePort() %><br>
服务器的名称为：<% = request.getServerName() %><br>
服务器的端口号为：<% = request.getServerPort() %><br>
客户请求使用的协议为：<% = request.getScheme() %><br>
客户端提交信息的页面为：<% = request.getServletPath() %><br>
客户端提交信息的长度为：<% = request.getContentLength() %><br>
采用的信息编码为：<% = request.getCharacterEncoding() %><br>
HTTP 文件头中的 User-Agent 值为：<% = request.getHeader("User-Agent") %><br>
HTTP 文件头中的 accept 值为：<% = request.getHeader("accept") %><br>
HTTP 文件头中的 Host 值为：<% = request.getHeader("Host") %><br>
Web 应用的目录为：<% = request.getContextPath() %>
</body>
```

运行本例的 example3_6.jsp 页面，结果如图 3.6 所示。

图 3.6　example3_6.jsp 的运行结果

假设在该页面中输入“Java Web 应用开发与实践”并单击“确定”按钮，程序将跳转到 result.jsp 页面，运行结果如图 3.7 所示。

图 3.7　result.jsp 的运行结果

注意：

使用 request 内置对象的诸多方法能够获得客户端及服务器的运行环境，还可以利用 getParameter()、setAttribute()、getAttribute()等方法实现在两个页面间传递数据。如果请求报文中含有中文字符会出现乱码的情况，原因在于请求报文的默认字符编码不支持中文。

动手实践 3-2

编写一个页面，尝试使用 request 对象的相关方法获取客户端和服务器的相关信息，例如 Web 应用的上下文、端口号、请求页面的名称等，并了解这些方法的使用和作用。

3.4 response 对象

当客户端向 Web 服务器发送请求后，Web 服务器接受请求并进行相应的响应(response)，一个 HTTP 响应报文包括状态行(Status Line)、响应头(Header)、空行(Blank Line)和可选实体内容(Body)。下面是一个 HTTP 响应报文的例子。

```
HTTP/1.1 200 OK
Date: Fri, 24 Jul 2015 08:07:21 GMT
Content - Type: text/html; charset = UTF - 8
< html >
      < head ></ head >
      < body >
            <! -- 网页主体内容,此处省略不再给出。 -- >
      </ body >
</ html >
```

在该响应报文中，HTTP 状态码为 200 表示找到资源且一切正常；Date 表示生成响应的日期和时间；Content-Type 指定了 MIME 类型为 text/html，编码类型是 UTF-8；最后为 HTML 源文件。

3.4.1 请求状态行

从 HTTP 响应报文可以看到，报文的第 1 行是状态行。状态行由协议版本、状态码和相关的文本短语组成，其中状态码由 3 位数字组成，状态码的第 1 位数字定义响应类表，后两位数字没有任何分类角色，第 1 位数字有 5 种取值，具体如下。

- 1XX：请求被接收到，继续处理；
- 2XX：被成功地接收；
- 3XX：重发，为了完成请求必须采取下一步动作；
- 4XX：客户端出错；
- 5XX：服务器端出错。

HTTP 响应状态码如表 3.4 所示。

表 3.4 HTTP 响应状态码

状 态 码	说 明	状 态 码	说 明
100	继续	206	部分内容
101	转换协议	300	多个选择
200	OK，成功	301	永久移动
201	已创建	302	发现
202	接受	303	见其他
203	非权威消息	304	没有被改变
204	无内容	305	使用代理
205	重置内容	400	坏请求

续表

状 态 码	说 明	状 态 码	说 明
401	未授权的	412	先决条件失败
402	必要的支付	413	请求实体太长
403	禁用	414	请求 URI 太大
404	资源未找到	415	不被支持的媒体类型
405	方式不被允许	500	服务器内部错误
406	不接受的	501	不能实现
407	需要代理验证	502	坏网关
408	请求超时	503	服务不能获得
409	冲突	504	网关超时
410	不存在	505	HTTP 版本不支持
411	长度必需		

3.4.2 response 内置对象的常用方法

response 内置对象的常用方法如表 3.5 所示。

表 3.5 response 内置对象的常用方法

方 法	说 明
void addCookie(Cookie cookie)	给客户端添加一个 Cookie 对象，以保存客户端的信息
void addDateHeader(String name,long value)	添加一个日期类型的 HTTP 首部信息，覆盖同名的 HTTP 首部
void addIntHeader(String name,int value)	添加一个整型的 HTTP 首部，并覆盖旧的 HTTP 首部
String encodeRedirectURL(String url)	对使用的 URL 进行编译
String encodeURL(String url)	封装 URL 并返回到客户端，实现 URL 重写
void flushBuffer()	清空缓冲区
int getBufferSize()	取得缓冲区的大小
String getCharacterEncoding()	取得字符编码类型
String getContentType()	取得 MIME 类型
Locale getLocale()	取得本地化信息
ServletOutputStream getOutputStream()	返回一个二进制输出字节流
PrintWriter getWriter()	返回一个输出字符流
void reset()	重设 response 对象
void resetBuffer()	重设缓冲区
void sendError(int sc)	向客户端发送 HTTP 状态码的出错信息
void sendRedirect()	重定向客户的请求到指定页面
void setBufferSize(int size)	设置缓冲区的大小为 size
void setCharacterEncoding(String encoding)	设置字符编码类型为 encoding
void setContentLength(int length)	设置响应数据的大小为 length
void setContentType(String type)	设置 MIME 类型
void setDateHeader(String s1,long l)	设置日期类型的 HTTP 首部信息
void setHeader(String s1,String s2)	设置 HTTP 首部信息
void setLocale(Locale locale)	设置本地化为 locale
void setStatus(int status)	设置状态码为 status

在上述方法中,使用 setHeader()方法可以设置页面自动刷新。

```
//设置页面每隔 1 秒自动刷新一次
response.setHeader("Refresh","1");
```

使用 response 对象的 setHeader()方法设置页面自动跳转。

```
//设置 10 秒后自动跳转到 anotherPage.jsp
response.setHeader("Refresh","10;URL = anotherPage.jsp");
```

使用 response 对象的 sendRedirect()方法可以实现页面直接跳转。

```
response.sendRedirect("anotherPage.jsp");
```

另外,还可以使用 response 对象禁用页面缓存。

```
//禁用页面缓存
response.setHeader("Cache - Control","no - cache");
response.setHeader("Pragma","no - cache");
response.setDateHeader ("Expires", 0);
```

下面是一个关于 response 内置对象设置 Cookie 的例子。

example3_7.jsp

```
<body>
  <%
        Cookie c1 = new Cookie("name","Java") ;
        Cookie c2 = new Cookie("password","123456") ;
        //最大保存时间为 180 秒
        c1.setMaxAge(180) ;
        c2.setMaxAge(180) ;
        //通过 response 对象将 Cookie 设置到客户端
        response.addCookie(c1) ;
        response.addCookie(c2) ;
   %>
</body>
```

cookie.jsp

```
<body>
        <%
          //通过 request 对象取得客户端设置的全部 Cookie
          //实际上客户端的 Cookie 是通过 HTTP 头信息发送到服务器端
          Cookie c[] = request.getCookies() ;
          for(int i = 0;i < c.length;i++)
          {
```

```
                Cookie temp = c[i] ;
        %>
                <% = temp.getName() %> : <% = temp.getValue() %><br>
        <%
          }
        %>
</body>
```

Cookie 是保存在客户端某个目录下的文本数据，由服务器生成该数据。当客户端下次请求该数据时，无须再从服务器端下载，而是从本地获取 Cookie 保存的信息（前提条件是浏览器启用 Cookie）。Cookie 是一组由 key 和 value 组成的键/值对。key 和 value 由开发者自定义，例如本例中定义了两组 Cookie，key 分别为 name 和 password，value 分别为 Java 和 123456。先后执行 example3_7.jsp 和 cookie.jsp 页面，客户端所保存的 Cookie 就会被 cookie.jsp 读取出来，具体如图 3.8 所示。

图 3.8　cookie.jsp 的运行结果

动手实践 3-3

新建一个 JSP 页面 from.jsp，在该页面中编写代码，使该页面跳转至 to.jsp。如果使 from.jsp 页面跳转至一个响应状态码为 404 的错误提示页面，将如何实现？

3.5　page 对象

page 对象代表当前正在运行的 JSP 页面，或者可以认为 page 代表的是 JSP 页面被编译后的 Servlet，相当于 Java 语言中的 Object 类。page 对象的主要方法如表 3.6 所示。

表 3.6　page 对象的主要方法

方　法	说　明
class getClass()	获取 page 对象的类
int hashCode()	获取 page 对象的 hash 码

example3_8.jsp

```
<body>
本页面对应的 Servlet 为：<% = page.getClass() %><br>
本页面对应的 Servlet hash 码为：<% = page.hashCode() %><br>
本页面使用的 JSP 引擎为：<% = ((HttpJspPage)page).getServletInfo() %><br>
</body>
```

运行结果如图 3.9 所示。

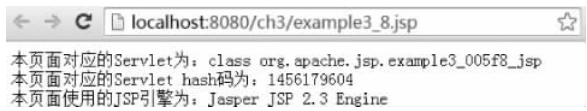

图 3.9 example3_8.jsp 的运行结果

3.6 pageContext 对象

pageContext 对象代表当前页面的上下文，即当前页面的所有属性和对象。pageContext 对象提供的方法可以获取当前页面的其他内置对象如 request、page、response 等。pageContext 对象的主要方法如表 3.7 所示。

表 3.7 pageContext 对象的主要方法

方 法	说 明
JspWriter getOut()	返回当前客户端响应使用的 JspWriter 流，即 out 对象
HttpSession getSession()	返回当前页中的 HttpSession 对象，即 session 对象
Object getPage()	返回当前页中的 Object 对象，即 page 对象
ServletRequest getRequest()	返回当前页中的 request 对象
ServletResponse getResponse()	返回当前页中的 response 对象
ServletConfig getServletConfig()	返回当前页中的 ServletConfig 对象
ServletContext getServletContext()	获取 ServletContext 对象，该对象在所有页面都是共享的
void setAttribute(String name,Object obj)	设置默认页面范围或特定对象范围内的属性 name，其值为 obj
void removeAttribute(String name)	删除默认页面范围或特定对象范围内的属性 name
Object getAttribute(String name)	获取默认页面范围或特定对象范围内的属性 name
void forward(String url)	将当前页面重定向到另一个页面或 Servlet 对象
Exception getException()	获取当前网页的异常对象
Object findAttribute(String name)	查找在所有范围内属性名称为 name 的属性

下面的例子通过使用 pageContext 对象保存并获取属性信息。

example3_9.jsp

```
<body>
<%
    //设置一个属性 name,其值为 PageContext,取值范围为 request
    pageContext.setAttribute("name","PageContext 对象"
                                    ,pageContext.REQUEST_SCOPE);
    pageContext.forward("TestPageContext.jsp");
 %>
```

pageContext.jsp

```
<body>
pageContent 对象的属性值为：<% =request.getAttribute("name") %>
</body>
```

运行 example3_9.jsp 页面，该页面设置了一个名称为 name、值为“PageContext 对象”的属性，并且使用 pageContext 对象的 forward()方法跳转到 pageContext.jsp 页面。程序的运行结果如图 3.10 所示。

图 3.10 example3_9.jsp 的运行结果

3.7 out 对象

out 对象的主要作用是向 JSP 页面输出各种类型的数据，并且管理 Web 服务器上的输出缓冲区。out 对象可以向 JSP 页面中输出文本内容，也可以输出 HTML 标签和 JavaScript 脚本。out 对象的主要方法如表 3.8 所示。

表 3.8 out 对象的主要方法

方 法	说 明
void print(DateType p)	向 JSP 页面中输出数据，但不结束当前行，下一个输出仍将在本行输出
void println(DateType p)	向 JSP 页面中输出数据，并且会结束当前行，下一个输出将在下一行输出
voidnewline()	换行
void close()	关闭输出流
void clear()	清空缓冲区的数据，但不把数据写到客户端
void clearBuffer()	清空缓冲区的数据，并将数据写到客户端
boolean isAutoFlush()	是否自动清空缓冲区，autoFlush 是通过 page 指令的 isAutoFlush 属性设置的
void flush()	清空缓冲区的数据
int getBufferSize()	返回缓冲区的大小，缓冲区的大小是通过 page 指令的 buffer 属性设置的
int getRemaining()	返回缓冲区的剩余空间大小

example3_10.jsp

```
<body>
<%
    //向 JSP 页面中输出文本
    out.print("明德新民,至于至善");
    //输出 HTML 标签,相当于换行
    out.print("<br>");
%>
缓冲区的大小为: <% =out.getBufferSize() %><br>
缓冲区的可用大小为: <% =out.getRemaining() %><br>
是否为自动清空缓冲区<% =out.isAutoFlush() %><br>
<%
    //输出 JavaScript 脚本
    out.println("<SCRIPT type=\"text/javascript\">alert(\"测试 out 对象的使用!\");</
SCRIPT>");
%>
</body>
```

运行结果如图 3.11 所示。

图 3.11 example3_10.jsp 的运行结果

注意：

out.print()方法和 System.out.print()方法的区别在于 out.print()方法是向 JSP 页面中输出数据，而 System.out.print()方法是向控制台(Console)输出数据。

3.8 session 对象

由于 HTTP 是一种无状态的协议，因此在 Web 应用开发中会话(session)是跟踪用户状态的一种重要手段。会话是指在一段时间内每个用户与 Web 应用的一连串相关的交互过程。在一个会话中，用户可以多次请求访问 Web 应用的页面。JSP 通过使用 session 对象保存每个用户的用户信息和会话状态。session 对象由 Web 容器自动创建，可以跟踪每个用户的操作状态。当用户首次登录 Web 应用时，Web 服务器自动给用户分配一个唯一的标识(即 session id)，以此来区分各个用户。session 对象采用 Map 类型保存数据，即每个用户都可以有若干个键/值对。

注意：

在 Web 应用中，Web 容器跟踪用户状态的方法通常有以下 4 种。

(1) 使用会话(session)；

(2) 在 HTML 表单中加入隐藏字段，它包含用于跟踪用户状态的数据；

(3) 重写 URL，使它包含用于跟踪用户状态的数据；

(4) 使用 Cookie 传送用于跟踪用户状态的数据。

session 对象的主要方法如表 3.9 所示。

表 3.9 session 对象的主要方法

方 法	说 明
long getCreationTime()	返回 session 的创建时间
String getId()	返回用户的 session id
long getLastAccessedTime()	返回 session 最后一次被操作的时间，单位为毫秒
Object getAttribute(String name)	返回会话属性名为 name 的值
Enumeration getAttributeNames()	返回一个枚举类型，即用户会话的所有属性
int getMaxInactiveInterval()	返回会话两次操作的最大时间间隔，超过此间隔该会话被取消
ServletContext getServletContext()	返回 Web 应用的上下文，即 Web 应用的路径
void invalidate()	取消会话

续表

方　法	说　明
boolean isNew()	返回服务器创建的一个会话，即客户端是否已经加入
void setAttribute(String name,Object obj)	设置指定名称为 name 的属性值为 obj，并存储在 session 对象中
void removeAttribute(String name)	移除会话中名称为 name 的属性
void setMaxInativeInterval(int time)	设置两次请求的最大时间间隔为 time，如超过 time 则 session 取消

example3_11.jsp 是一个模拟用户登录的程序，客户端通过输入用户名与密码登录，如登录成功，使用 session 对象的 setAttribute()和 getAttribute()方法存取用户名。

example3_11.jsp

```
<form action = "success.jsp" method = "post">
用户名:<input type = "text" name = "username" /><br>
密　码:<input type = "password" name = "password" /><br>
<input type = "submit" value = "登录" />
</form>
```

success.jsp

```
<body>
<%
    //取得输入的用户名和密码
    String name = request.getParameter("username");
    String pwd = request.getParameter("password");
    //判断用户名与密码是否正确,默认用户为 root,密码为 123456
    if(name.equals("root")&&pwd.equals("123456"))
    {
        //设置 session 属性 username,值为 name
        session.setAttribute("username",name);
        //通过 getAttribute()方法获取会话的 username 属性值,即 name 的值
        String user = (String)session.getAttribute("username");
        out.print(user + "欢迎你!");
    }
    else
    {
        out.print("<script type = \"text/javascript\">alert('对不起,用户名或密码错误!');
history.go(-1);</script>");
    }
%>
</body>
```

运行 example3_11.jsp，输入用户名和密码，如图 3.12 所示，登录成功后出现如图 3.13 所示的页面。

注意：

在使用 session 对象时要注意，即使是多个用户同时访问同一个 Web 应用的同一个页

面，各个用户的 session 的属性值也是相互独立的，各用户之间的 session 是不共享的。

图 3.12 example3_11.jsp 的运行结果

图 3.13 登录成功页面

动手实践 3-4

本节介绍了将 username 放入 session 的例子，那么如何实现用户注销呢？即如何清除 session 中相应的信息。请读者在上例的基础上实现用户注销功能。

3.9 application 对象

application 对象用于保存 Web 应用中的共享数据，与 session 对象不同的是，application 对象中的属性值是各个用户共享使用的，而各个用户之间的 session 对象没有任何必然联系；另外，application 对象的生存期要比 session 对象长，session 只在当前的会话期内有效，而 application 对象在 Web 服务器启动之后即产生，直至 Web 服务器关闭之前将一直存在。application 对象的主要方法如表 3.10 所示。

表 3.10 application 对象的主要方法

方 法	说 明
Object getAttribute(String name)	获得 application 对象的 name 属性值
Enumeration getAttributeNames()	获得所有属性的名称，返回类型为枚举类型
String getInitParameter(String name)	获得 name 属性的初始值
String getServerInfo()	获得当前 JSP 引擎名及版本信息等
String getRealPath(String path)	返回该 Web 应用的实际路径
ServletContext getContext(String path)	返回指定 Web 应用的 application 对象
int getMajorVersion()	返回服务器支持的 Servlet API 主版本号
int getMinorVersion()	返回服务器支持的 Servlet API 次版本号
String getMimeType(String file)	返回指定文件的 MIME 类型
URL getResource(String path)	返回指定资源(例如文件、目录)的 URL
Servlet getServlet(String name)	返回指定名称的 Servlet
Enumeration getServlets()	返回所有 Servlet 的枚举
Enumeration getServletNames()	返回所有 Servlet 名的枚举
void setAttribute(Stirng name,value k)	设置 application 范围内的属性 name 的值为 k

下面是一个使用 application 对象实现计数器功能的例子。

example3_12.jsp

```
<body>
<%
    Object number = application.getAttribute("count");
    int num = 0;
```

```
        if(number == null)
        {
            application.setAttribute("count",1);
        }
        else
        {
            num = (Integer)number;
            num++;
            application.setAttribute("count",num);
        }
        out.print("当前访问本页面的次数为: " + num);
    %>
    <br>
    以下是application对象信息<br>
    ----------------------------------------------------
    <br>
    支持Servlet API主版本号: <% =application.getMajorVersion() %><br>
    支持Servlet API次版本号: <% =application.getMinorVersion() %><br>
    Web应用的实际路径: <% =application.getRealPath("/") %><br>
    Web服务器的版本信息: <% =application.getServerInfo() %><br>
    </body>
```

运行结果如图 3.14 所示。

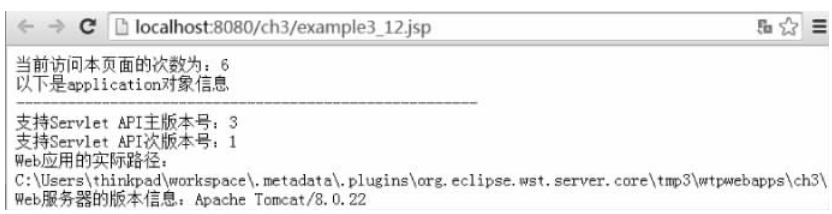

图 3.14　example3_12.jsp 的运行结果

动手实践 3-5

request、session 和 application 对象中都有 setAttribute()和 getAttribute()方法，其主要区别在于它们的作用域不同，请读者编写一个程序测试这些对象的相应方法的区别。

3.10　config 对象

config 对象主要用于读取 Web 应用的初始化参数，在 Java Web 应用中一般使用 web.xml 配置文件存储 Web 应用的配置信息。首先来认识 web.xml 配置文件和各元素的作用。

3.10.1　web.xml 配置文件

web.xml 配置文件是一个 XML 文件，保存在 Web 应用的 WEB-INF 文件夹下，主要作用是配置 Web 应用程序的欢迎页、Servlet、Filter 等。当 Web 应用没有使用到这些功能时，

web.xml 配置文件也是可以省略的。

在 web.xml 配置文件中定义了多种元素，下面是一个基本的 web.xml 配置文件：

```
<?xml version = "1.0" encoding = "UTF - 8"?>
< web - app xmlns:xsi = "http://www.w3.org/2001/XMLSchema - instance" xmlns = "http://xmlns.
jcp.org/xml/ns/javaee" xsi:schemaLocation = "http://xmlns.jcp.org/xml/ns/javaee http://
xmlns.jcp.org/xml/ns/javaee/web - app_3_1.xsd" id = "WebApp_ID" version = "3.1">
    < display - name > Project Name </display - name >
    < welcome - file - list >
        < welcome - file > index.jsp </welcome - file >
    </welcome - file - list >
</web - app >
```

web.xml 配置文件中的元素名称及其顺序有严格规定，但需要哪些元素因具体项目有所变化。每个 web.xml 配置文件的根元素＜web-app＞中定义的子元素并不是固定不变的，模式文件也是可以改变的，一般来说，随着 web.xml 文件的版本升级，其中定义的功能会越来越复杂，子元素的种类会越来越多。下面列出 web.xml 常用的元素以及这些元素的功能。

(1) 设置欢迎页面。也就是为 Web 应用设置首页，具体语法如下：

```
< welcome - file - list >
        < welcome - file > index.jsp </welcome - file >
        < welcome - file > index.htm </welcome - file >
</welcome - file - list >
```

＜welcome-file-list＞元素定义 Web 应用系统的首页，该元素下面又嵌套了＜welcome-file＞子元素并指定了两个欢迎页面，显示时按顺序首先从第一个查找，如果第一个页面存在，即设置第一个页面为首页面，后面的不起作用；否则继续查找第二个，以此类推。

(2) 命名与定制 URL。用户可以为 Servlet 和 JSP 文件命名并定制 URL，其中定制 URL 依赖命名，命名必须在定制 URL 前面定义。

第一步：为 Servlet 命名。

```
< servlet >
    < description >用户管理</description >
    < display - name > UserService </display - name >
    < servlet - name > UserAction </servlet - name >
    < servlet - class > com.action.UserServlet </servlet - class >
    < load - on - startup > 10 </load - on - startup >
</servlet >
```

其中，在＜servlet＞元素下又嵌套了几个子元素。

- ＜description＞：描述该 Servlet，为可选子元素；
- ＜display-name＞：设置显示该 Servlet 的名称，为可选子元素；
- ＜servlet-name＞：设置该 Servlet 的名称；
- ＜servlet-class＞：设置该 Servlet 具体的包路径；

- ＜load-on-startup＞：指定 Servlet 加载的次序，即启动装入优先权。数值越小，其优先级越高，需要说明的是这个数值是相对于其他 Servlet 而言的。

第二步：为 Servlet 定制 URL。

```
<servlet-mapping>
    <servlet-name>UserAction</servlet-name>
    <url-pattern>/admin/UserAction</url-pattern>
</servlet-mapping>
```

其中，＜servlet-mapping＞元素下也包含两个子元素。

- ＜servlet-name＞：设置 Servlet 的名称，必须与＜servlet＞元素下的＜servlet-name＞子元素的值相同；
- ＜url-pattern＞：设置该 Servlet 的 URL，如本例的 Web 应用名称为 ch3，那么该 Servlet 的 URL 为“http://localhost/ch3:8080/admin/UserAction”。

(3) 定制初始化参数。用户可以定制 Servlet、JSP、Context 的初始化参数，然后就可以使用 config 等内置对象在 Servlet、JSP 以及 Context 中获取这些参数值。

```
<servlet>
    <servlet-name>UserAction</servlet-name>
    <servlet-class>com.action.UserServlet</servlet-class>
    <init-param>
        <param-name>username</param-name>
        <param-value>Eric</param-value>
    </init-param>
    <init-param>
        <param-name>Email</param-name>
       <param-value>liangsbin@126.com</param-value>
    </init-param>
</servlet>
```

经过上面的配置，在 Servlet 中就可以使用 config 调用这些初始化参数了，例如使用 config.getInitParameter("Email")获得参数 Email 对应的值。

(4) 指定错误处理页面。用户可以通过“异常类型”或“状态码”指定错误处理页面。

```
<error-page>
    <error-code>404</error-code>
    <location>/error/NotFound.jsp</location>
</error-page>
<error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/exception.jsp</location>
</error-page>
```

其中，在＜error-page＞元素中可以有以下几种子元素。

- ＜error-code＞：指定 HTTP 状态码；
- ＜exception-type＞：指定异常类型；

• <location>：指定发生异常或者出错对应状态码时跳转到的错误处理页面。

(5) 设置过滤器。例如设置一个字符编码过滤器，在访问 Web 应用的所有资源时都要执行本过滤器。

```
<filter>
    <filter-name>CharSetFilter</filter-name>
    <filter-class>com.filter.CharSetFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>CharSetFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

(6) 设置监听器。

```
<listener>
    <listener-class>com.listener.MyLisenet</listener-class>
</listener>
```

(7) 设置会话过期时间。其中时间以分钟为计时单位，假如设置 20 分钟超时，超过规定时间后 session 失效。

```
<session-config>
    <session-timeout>20</session-timeout>
</session-config>
```

3.10.2 config 对象的主要方法

config 对象的主要方法如表 3.11 所示。

表 3.11 config 对象的主要方法

方 法	说 明
String getInitParameter(String name)	返回名称为 name 的初始化参数值
Enumeration getInitParameterNames()	返回所有初始化参数名称的枚举
ServletContext getServletContext()	返回当前 Servlet 的上下文
String getServletNames()	返回当前 Servlet 的名称

下面是一个使用 config 对象读取 web.xml 文件中初始化参数 parameter 的例子，以下是 web.xml 的内容。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID"
version="3.1">
```

```
    <display-name>ch3</display-name>
    <servlet>
    <servlet-name>config</servlet-name>
    <jsp-file>/example3_13.jsp</jsp-file>
    <!--定义初始化参数 data-->
    <init-param>
        <param-name>data</param-name>
        <param-value>123456</param-value>
    </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>config</servlet-name>
        <url-pattern>/example3_13.jsp</url-pattern>
    </servlet-mapping>
</web-app>
```

example3_13.jsp

```
<body>
读取 web.xml 中初始化参数 data 的值为：<%=config.getInitParameter("data") %>
</body>
```

运行结果如图 3.15 所示。

图 3.15　example3_13.jsp 的运行结果

动手实践 3-6

请读者在 web.xml 中添加一个名称为 number 的初始化参数，并设置其值为 20。然后在项目中编写一个 JSP 页面，在该页面中尝试使用 config 对象的相应方法读取 number 的数值。

3.11　exception 对象

exception 对象是 java.lang.Throwable 的实例对象，exception 对象主要用来处理 JSP 页面执行时产生的异常，从而提高 Web 应用的健壮性。需要注意的是，使用 exception 对象处理异常时需要指定错误处理页面，而错误处理页面通过使用 page 指令来设置。exception 对象的主要方法如表 3.12 所示。

表 3.12　exception 对象的主要方法

方　法	说　明
String getMessage()	返回异常信息
void printStackTrace()	以标准错误的形式输出错误及堆栈跟踪信息
String toString()	以字符串形式返回异常信息

下面是一个 exception 对象的例子。

第一步：首先创建异常处理页面。

example3_14.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" isErrorPage="true" %>
⋮
<body>
<%
out.println("error.jsp 发生了错误,具体原因如下:<br>");
out.println("--------------------------------<br>");
out.println(exception.getMessage());
%>
</body>
```

第二步：创建业务逻辑页面。

error.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8" errorPage="example3_14.jsp" %>
⋮
<body>
<%
int[] array = {1,2,3,4,5};
//下面的代码将导致数组越界
for(int i=0;i<6;i++)
{
    out.print("array[" + i + "] = " + array[i]);
    out.newLine();
}
%>
</body>
```

error.jsp 显然存在数据越界异常,由于 error.jsp 页面的 page 指令使用 errorPage 属性设定错误处理页面为 example3_14.jsp,并且 error.jsp 页面中没有异常处理程序,那么 error.jsp 页面执行产生异常后,它将交给 example3_14.jsp 页面处理这个异常,运行结果如图 3.16 所示。

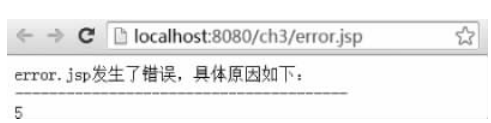

图 3.16 error.jsp 的运行结果

注意：

在使用 exception 内置对象时需要注意以下两点。

(1) 在异常处理页面中需要设定该页面 page 指令的 isErrorPage 属性值为 true。

(2) 如果在产生异常的页面中使用 try-catch-finally 进行异常捕获和处理,那么将不会

触发异常处理页面运行。

本章小结

本章介绍JSP的语法基础和内置对象。一个JSP页面由5个部分组成，即HTML元素、注释、脚本元素、动作和指令。脚本元素包括JSP页面变量和方法的声明、表达式和Java程序段。指令元素包括page指令、include指令和taglib指令等。动作标记包括<jsp:include>、<jsp:param>、<jsp:forward>、<jsp:useBean>、<jsp:setProperty>和<jsp:getProperty>等。

JSP提供了9种常用的内置对象，分别是request、response、session、application、config、pageContext、out、exception和page。内置对象属于某种接口或类，内置对象的优势在于使用这些内置对象时无须声明即可直接使用，从而简化了开发过程。