第3章

JSP技术基础

JSP(Java Server Pages)是一种动态页面技术,它的主要目的是将表示逻辑从 Servlet 中分离出来,实现表示逻辑。在 MVC 模式中,JSP 页面实现视图功能。

本章首先介绍 JSP 语法和生命周期、脚本元素、隐含变量、JSP 动作,然后介绍错误处理、作用域对象和 JavaBean 的使用,最后介绍 MVC 设计模式。

本章内容要点

- JSP 页面元素。
- JSP 生命周期。
- JSP 指令和动作。
- JSP 隐含变量。
- 错误处理。
- 作用域对象与 JavaBean。
- MVC 设计模式。

扫一扫

视频讲解

3.1 JSP 页面元素

在 JSP 页面中可以包含多种 JSP 元素,如声明变量和方法、JSP 表达式、指令和动作等,这些元素具有严格定义的语法。当 JSP 页面被访问时,Web 容器将 JSP 页面转换成 Servlet 类执行后将结果发送给客户。与其他的 Web 页面一样,JSP 页面也有一个唯一的 URL,客户可以通过它访问页面。一般来说,在 JSP 页面中可以包含的元素如表 3-1 所示。

表 3-1 在 JSP 页面中可以包含的元素

JSP 页面元素		简 要 说 明	标 签 语 法
指令		指定转换时向容器发出的指令	<%@ 指令 %>
动作		向容器提供请求时的指令	<jsp:动作名/>
EL 表达式		JSP 页面使用的数据访问语言	${EL 表达式}
脚本元素	JSP 声明	声明变量与定义方法	<%! Java 声明 %>
	JSP 小脚本	执行业务逻辑的 Java 代码	<% Java 代码 %>
	JSP 表达式	用于在 JSP 页面中输出表达式的值	<% = 表达式 %>
注释		用于文档注释	<% -- 任何文本 -- %>

在一个JSP页面中，除JSP元素外，其他内容称为**模板文本**（template text），也就是HTML标记和文本。清单3.1是一个简单的JSP页面，其中包含了多种JSP元素。

清单3.1　todayDate.jsp

```
<%@ page contentType="text/html;charset=UTF-8" %>
<%@ page import="java.time.LocalDate" %>
<%! LocalDate date=null; %>
<html>
<head><title>当前日期</title>
</head>
<body>
  <%
    date=LocalDate.now();
  %>
  今天的日期是:<%=date.toString()%>
</body>
</html>
```

该页面中包含JSP指令、JSP声明、JSP小脚本、JSP表达式和模板文本。当JSP页面被客户访问时，页面首先在服务器端被转换成一个Java源程序文件，然后该程序在服务器端编译和执行，最后向客户发送执行结果，通常是文本数据。这些数据由HTML标签包围起来，然后发送到客户端。由于嵌入在JSP页面中的Java代码是在服务器端处理的，客户并不了解这些代码。

3.1.1　JSP指令简介

JSP指令（directive）向容器提供关于JSP页面的总体信息。在JSP页面中，指令是以“<%@”开头，以“%>”结束的标签。指令有page指令、include指令和taglib指令3种类型。这3种指令的语法格式如下。

```
<%@ page attribute-list %>
<%@ include attribute-list %>
<%@ taglib attribute-list %>
```

在上面的指令标签中，attribute-list表示一个或多个针对指令的“属性-值”对，多个属性之间用空格分隔。

1. page指令

page指令通知容器关于JSP页面的总体特性。例如，下面的page指令通知容器页面输出的内容类型和使用的字符集。

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

2. include指令

include指令实现把一个文件（HTML、JSP等）的内容包含到当前页面中。下面是include指令的一个例子。

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

3. taglib指令

taglib指令用来指定在JSP页面中使用标准标签或自定义标签的前缀与标签库的URI。下面是taglib指令的例子。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

关于page指令的详细信息，请读者参考3.3.1节，在3.3.2节将详细讨论include指令，

在第 4 章将学习 taglib 指令的使用。

3.1.2　表达式语言

表达式语言(Expression Language,EL)是一种可以在 JSP 页面中使用的简洁的数据访问语言。它的语法格式如下。

```
${expression}
```

表达式语言以 $ 开头,后面是一对大括号,括号里面的 expression 是 EL 表达式,也可以是作用域变量、EL 隐含变量等。该结构可以出现在 JSP 页面的模板文本中,也可以出现在 JSP 标签的属性中。

清单 3.1 中对于日期的输出可以使用表达式语言改写如下。

```
今天的日期是:${LocalDate.now()}
```

由于 LocalDate 类的 now()方法是静态工厂方法,所以可以在 EL 中直接调用,但需要使用 page 指令将类导入。第 4 章将详细讨论表达式语言。

3.1.3　JSP 动作

JSP 动作(actions)是页面发给容器的命令,它指示容器在页面执行期间完成某种任务。动作的一般语法如下。

```
<prefix:actionName attribute-list />
```

动作是一种标签,在动作标签中,prefix 为前缀名,actionName 为动作名,attribute-list 表示针对该动作的一个或多个“属性-值”对。

在 JSP 页面中可以使用 3 种动作,即 JSP 标准动作、标准标签库(JSTL)中的动作和用户自定义动作。例如,下面的代码指示容器把 JSP 页面 copyright.jsp 的输出包含在当前 JSP 页面的输出中。

```
<jsp:include page="copyright.jsp"/>
```

3.1.4　JSP 脚本元素

脚本元素(scripting elements)是在 JSP 页面中使用的 Java 代码,主要用于实现业务逻辑,通常有 Java 声明、Java 小脚本和 Java 表达式 3 种脚本元素。

1. JSP 声明

JSP 声明(declaration)用来在 JSP 页面中声明变量和定义方法。声明是以“<%!”开头,以“%>”结束的标签,其中可以包含任意数量的合法的 Java 声明语句。下面是 JSP 声明的一个例子。

```
<%! LocalDate date = null; %>
```

上面的代码声明了一个名为 date 的变量并将其初始化为 null。声明的变量仅在页面第一次载入时由容器初始化一次,初始化后在后面的请求中一直保持该值。

注意:由于声明包含的是声明语句,所以每个变量的声明语句必须以分号结束。

下面的代码在一个标签中声明了一个变量 r 和一个方法 getArea()。

```
<%!
   double r = 0;                        //声明一个变量 r
   double getArea(double r) {           //声明求圆面积的方法
```

```
        return r * r * Math.PI;
    }
%>
```

2. JSP 小脚本

JSP 小脚本(scriptlets)是嵌入在 JSP 页面中的 Java 代码段。小脚本是以“<%”开头，以“%>”结束的标签。例如，在清单 3.1 中下面的代码就是 JSP 小脚本。

```
<%
   date = LocalDate.now();              //创建一个 LocalDate 对象
%>
```

小脚本在每次访问页面时都被执行，因此 date 变量在每次请求时会返回当前日期。由于小脚本可以包含任何 Java 代码，所以它通常用来在 JSP 页面中嵌入计算逻辑。另外，用户还可以使用小脚本打印 HTML 模板文本。

与其他元素不同，小脚本的起始标签“<%”后面没有任何特殊字符，在小脚本中的代码必须是合法的 Java 语言代码。例如，下面的代码是错误的，因为它没有使用分号结束。

```
<% out.print(count) %>
```

注意：不能在小脚本中声明方法，因为在 Java 语言中不能在方法中定义方法。

3. JSP 表达式

JSP 表达式(expression)是以“<%=”开头，以“%>”结束的标签，它作为 Java 语言中表达式的占位符。下面是 JSP 表达式的例子。

```
今天的日期是:<% = date.toString() %>
```

在页面每次被访问时都要计算表达式，然后将其值嵌入 HTML 的输出中。与变量的声明不同，表达式不能以分号结束，因此下面的代码是非法的。

```
<% = date.toString(); %>
```

使用表达式可以向输出流输出任何对象或任何基本数据类型(int、boolean、char 等)的值，也可以打印任何算术表达式、布尔表达式或方法调用返回的值。

提示：在 JSP 表达式的百分号和等号之间不能有空格。

3.1.5 JSP 注释

JSP 注释是以“<%--”开头，以“--%>”结束的标签。注释不影响 JSP 页面的输出，但它对用户理解代码很有帮助。JSP 注释的格式如下。

```
<% -- 这里是 JSP 注释内容 -- %>
```

Web 容器在输出 JSP 页面时去掉 JSP 注释内容，所以在调试 JSP 页面时可以将 JSP 页面中的一大块内容注释掉，包括嵌套的 HTML 和其他 JSP 标签，然而不能在 JSP 注释内嵌套另一个 JSP 注释。

3.2 JSP 生命周期

一个 JSP 页面在执行过程中要经历多个阶段，这些阶段称为**生命周期阶段**(life-cycle phases)。在讨论 JSP 页面的生命周期前需要了解 JSP 页面和它的实现类。

3.2.1　JSP 页面的实现类

JSP 页面从结构上看与 HTML 页面类似，但它实际上是作为 Servlet 运行的。当 JSP 页面第一次被访问时，Web 容器解析 JSP 文件并将其转换成相应的 Java 文件，该文件声明了一个 Servlet 类，该类称为页面实现类。接下来，Web 容器编译该类并将其装入内存，然后与其他 Servlet 一样执行并将其输出结果发送到客户端。

这里以清单 3.1 的 todayDate.jsp 页面为例，看一下 Web 容器将 JSP 页面转换后的 Java 文件代码。在页面转换阶段 Web 容器自动将该文件转换成一个名为 todayDate_jsp.java 的类文件，该文件是 JSP **页面实现类**。若 Web 项目部署到 Tomcat 服务器，该文件存放在安装路径的\work\Catalina\localhost\chapter03\org\apache\jsp 目录中。限于篇幅，这里不给出页面实现类的源代码，感兴趣的读者可以自己查看。

页面实现类继承了 HttpJspBase 类，同时实现了 JspPage 接口，该接口又继承了 Servlet 接口。JspPage 接口只声明了 jspInit()和 jspDestroy()两个方法，所有的 JSP 页面都应该实现这两个方法。在 HttpJspPage 接口中声明了一个_jspService()方法。这 3 个 JSP 方法的格式如下。

```
public void jspInit();

public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException, IOException;

public void jspDestroy();
```

这 3 个方法分别等价于 Servlet 的 init()、service()和 destroy()方法，称为 **JSP 页面的生命周期方法**。

每个容器销售商都提供了一个特定的类作为页面实现类的基类。在 Tomcat 中，JSP 页面转换的类继承了 HttpJspBase 类。

JSP 页面中的所有元素都转换成页面实现类的对应代码，page 指令的 import 属性转换成 import 语句，page 指令的 contentType 属性转换成 response.setContentType()调用，JSP 声明的变量转换为成员变量，小脚本转换成正常的 Java 语句，模板文本和 JSP 表达式都使用 out.write()方法打印输出，输出是用转换的_jspService()方法完成的。

3.2.2　JSP 执行过程

下面以 todayDate.jsp 页面为例说明 JSP 页面的生命周期阶段。当客户首次访问该页面时，Web 容器执行该 JSP 页面要经过 7 个阶段，如图 3-1 所示。其中，前 4 个阶段将 JSP 页面转换成一个 Servlet 类并装载和创建该类的实例，后 3 个阶段是初始化阶段、提供服务阶段和销毁阶段。

1. 转换阶段

Web 容器读取 JSP 页面对其解析，并将其转换成 Java 源代码。JSP 文件中的元素都转换成页面实现类的成员。在这个阶段，容器将检查 JSP 页面中标签的语法，如果发现错误将不能转换。例如，下面的指令就是非法的，因为在“Page”中使用了大写字母 P，这将在转换阶段被捕获。

```
<%@ Page import = "java.util.*" %>
```

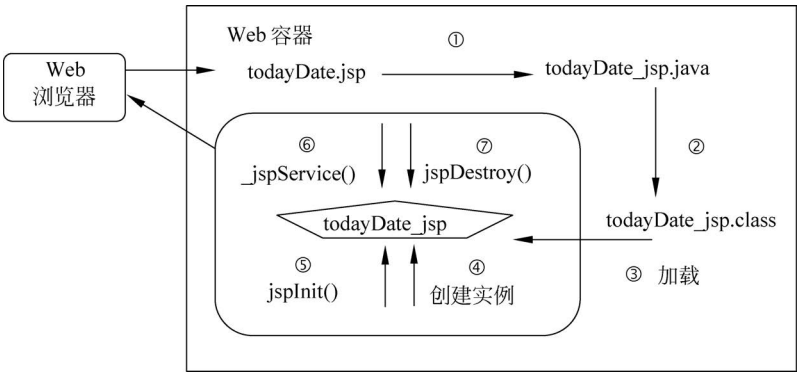

图 3-1 JSP 页面的生命周期阶段

除检查语法外，容器还将执行其他有效性检查，其中一些涉及验证。

- 指令中“属性-值”对对于标准动作的合法性。
- 同一个 JavaBean 名称在一个转换单元中没有被多次使用。
- 如果使用了自定义标签库，标签库是否合法、标签的用法是否合法。

一旦验证完成，Web 容器将 JSP 页面转换成页面实现类，它实际上是一个 Servlet，3.2.1 节中描述了页面实现类及其存放位置。

2. 编译阶段

在将 JSP 页面转换成 Java 文件后，Web 容器调用 Java 编译器编译该文件。在编译阶段，将检查在声明中、小脚本中及表达式中所写的全部 Java 代码。例如，下面的声明标签尽管能够通过转换阶段，但由于声明语句没有以分号结束，所以不是合法的 Java 声明语句，因此在编译阶段会被查出。

```
<%! LocalDate date = null %>
```

大家可能注意到，当 JSP 页面被首次访问时，服务器响应要比以后的访问慢一些，这是因为在 JSP 页面向客户提供服务之前必须要转换成 Servlet 类的实例。对于每个请求，容器要检查 JSP 页面源文件的时间戳及相应的 Servlet 类文件，以确定页面是否为新的或是否已经转换成类文件。因此，如果修改了 JSP 页面，将 JSP 页面转换成 Servlet 的整个过程要重新执行一遍。

3. 类的加载和实例化

在将页面实现类编译成类文件后，Web 容器调用类加载程序(class loader)将页面实现类加载到内存中，然后容器调用页面实现类的默认构造方法创建该类的一个实例。

4. 调用 jspInit()

Web 容器调用 jspInit()方法初始化 Servlet 实例。该方法是在任何其他方法调用之前调用的，并在页面生命周期内只调用一次。通常在该方法中完成初始化或只需一次的设置工作，如获得资源及初始化 JSP 页面中使用<%! … %>声明的实例变量。

5. 调用 _jspService()

对该页面的每次请求，容器都调用一次 _jspService()，并给它传递请求和响应对象。JSP 页面中所有的 HTML 元素、JSP 小脚本及 JSP 表达式在转换阶段都成为该方法的一部分。

6. 调用 jspDestroy()

当容器决定停止该实例提供服务时，它将调用 jspDestroy()，这是在 Servlet 实例上调用

的最后一个方法，它主要用来清理 jspInit()获得的资源。

一般不需要实现 jspInit()和 jspDestroy()，因为它们已经由基类实现，但可以根据需要使用 JSP 的声明标签<%! ... %>覆盖这两个方法。注意，不能覆盖_jspService()，因为该方法由 Web 容器自动产生。

扫一扫

视频讲解

3.3　JSP 指令

在 3.1.1 节中简要介绍了 JSP 指令，它用于向容器提供关于 JSP 页面的总体信息。在 JSP 页面中，指令是以“<%@”开头，以“%>”结束的标签。指令有 page 指令、include 指令和 taglib 指令 3 种类型，下面详细介绍这 3 种指令的使用。

3.3.1　page 指令

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。例如，下面的 page 指令通知容器页面输出的内容类型和使用的字符集。

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

表 3-2 列出了 page 指令的常用属性。

表 3-2　page 指令的常用属性

属　性　名	说　　明	默　认　值
import	导入在 JSP 页面中使用的 Java 类和接口，它们之间用逗号分隔	java.lang.*； jakarta.servlet.*； jakarta.servlet.jsp.*； jakarta.servlet.http.*；
contentType	指定输出的内容类型和字符集	text/html;charset=UTF-8
pageEncoding	指定 JSP 文件的字符编码	UTF-8
session	用布尔值指定 JSP 页面是否参加 HTTP 会话	true
errorPage	用相对 URL 指定另一个 JSP 页面用来处理当前页面的错误	null
isErrorPage	用布尔值指定当前 JSP 页面是否为错误处理页面	false
language	指定容器支持的脚本语言	java
extends	任何合法地实现了 jakarta.servlet.jsp.JspPage 接口的 Java 类	与实现有关
buffer	指定输出缓冲区的大小	与实现有关
autoFlush	指定是否当缓冲区满时自动刷新	true
info	关于 JSP 页面的任何文本信息	与实现有关
isThreadSafe	指定页面是否同时为多个请求服务	true
isELIgnored	指定是否忽略对 EL 表达式求值	false

大家应该了解 page 指令的所有属性及它们的取值，尤其是 contentType、pageEncoding、import、session、errorPage 和 isErrorPage 属性。

1. contentType 和 pageEncoding 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 UTF-8。MIME 类型和字符集之间用分号分隔，例如：

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

上述代码与 Servlet 中的下面一行等价。

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

对于包含中文的 JSP 页面，字符编码指定为 UTF-8 或 GB18030，如果页面仅包含英文字符，字符编码可以指定为 ISO-8859-1，例如：

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

pageEncoding 属性指定 JSP 页面的字符编码，它的默认值为 UTF-8。如果设置了该属性，JSP 页面使用该属性设置的字符集编码；如果没有设置该属性，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 或 GB18030，例如：

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

2. import 属性

import 属性的功能类似于 Java 程序中的 import 语句，它是将 import 属性值指定的类导入页面中。在转换阶段，容器将使用 import 属性声明的每个包都转换成页面实现类的一个 import 语句。在一个 import 属性中可以导入多个包，包名用逗号分开，例如：

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

为了增强代码的可读性也可以使用多个 page 指令，如上面的 page 指令也可以写成：

```
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
<%@ page import="com.boda.xy.*" %>
```

由于在 Java 程序中 import 语句的顺序是没有关系的，所以这里 import 属性的顺序也没有关系。另外，容器总是导入 java.lang.*、jakarta.servlet.*、jakarta.servlet.http.* 和 jakarta.servlet.jsp.* 包，所以不必明确地导入它们。

3. 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(将在 3.4 节学习更多的隐含变量)。如果不希望页面参加会话，可以明确地加入下面一行：

```
<%@ page session = "false" %>
```

4. errorPage 和 isErrorpage 属性

在页面执行的过程中，嵌入在页面中的 Java 代码可能抛出异常。与一般的 Java 程序一样，在 JSP 页面中也可以使用 try-catch 块处理异常。JSP 规范定义了一种更好的方法，它可以使错误处理代码与主页面代码分离，从而提高异常处理机制的可重用性。在该方法中，JSP 页面使用 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。在清单 3.2 创建的 helloUser.jsp 页面中，errorHandler.jsp 被指定为错误处理页面。

清单 3.2　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。");
    }
```

```
    %>
    你好!<% = request.getParameter("name") %>
</body>
</html>
```

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 errorHandler.jsp 页面。

errorPage 属性的值不必一定是 JSP 页面,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。

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。如在上例使用的 errorHandler.jsp 页面中该属性必须明确地设置为 true,见清单 3.3。

清单 3.3　errorHandler.jsp

```
<%@ page contentType = "text/html;charset = UTF - 8" %>
<%@ page isErrorPage = "true" %>
<html>
<body>
  <table>
    <tr><td><img src = "images/error.png" width = 150 height = 100/></td>
    <td>页面发生了下面的错误:<% = exception.getMessage() %><br>
    请重试!</td></tr>
  </table></body>
</html>
```

在这种情况下,容器在页面实现类中声明一个名为 exception 的隐含变量。

注意:该页面仅从异常对象中检索信息并产生适当的错误消息。因为该页面没有实现任何业务逻辑,所以可以被不同的 JSP 页面重用。

一般来说,为所有的 JSP 页面指定一个错误页面是一个良好的编程习惯,这可以防止在客户端显示不希望显示的错误消息。

3.3.2　include 指令

代码的可重用性是软件开发的一个重要原则。使用可重用的组件可以提高应用程序的生产率和可维护性。JSP 规范定义了一些允许重用 Web 组件的机制,其中包括在 JSP 页面中包含另一个 Web 组件的内容或输出。这可以通过静态包含或动态包含方式实现。

include 指令用于把另一个文件(HTML、JSP 等)的**内容**包含到当前页面中,这种包含称为静态包含。**静态包含**是在 JSP 页面转换阶段将另一个文件的内容包含到当前 JSP 页面中。使用 JSP 的 include 指令完成这一功能,它的语法如下。

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

file 是 include 指令唯一的属性,它指被包含的文件。文件使用相对路径指定,相对路径或者以斜杠(/)开头,是相对于 Web 应用程序文档根目录的路径;或者不以斜杠开头,是相对于当前 JSP 文件的路径。

下面是 include 指令的一个例子。

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

由于被包含文件的内容成为主页面代码的一部分，所以每个页面都可以访问在另一个页面中定义的变量，它们共享所有的隐含变量。清单 3.4 创建的 hello.jsp 页面中包含了 response.jsp 页面。

清单 3.4　hello.jsp

```
<%@ page contentType="text/html;charset=UTF-8" %>
<html>
<head><title>Hello</title></head>
<%! String usernname="Duke"; %>
<body>
    <img src="images/duke.gif">
    我的名字叫 Duke,你的名字叫什么?
    <form action="" method="post">
      <input type="text" name="username" size="25">
      <input type="submit" value="提交">
      <input type="reset" value="重置">
    </form>
    <%@ include file="response.jsp" %>
</body>
</html>
```

清单 3.5 创建被包含页面 response.jsp。

清单 3.5　response.jsp

```
<%@ page contentType="text/html;charset=UTF-8" %>
<% username=request.getParameter("username"); %>
<h4 style="color:blue">你好,<%=username%>!</h4>
```

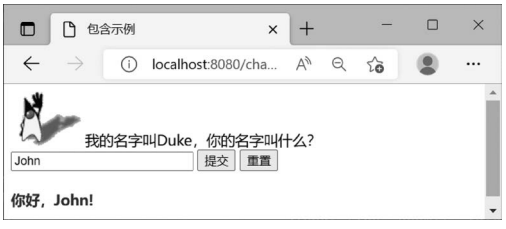

图 3-2　hello.jsp 的运行结果

在 hello.jsp 页面中声明了一个变量 username，并使用 include 指令包含了 response.jsp 页面。在 response.jsp 页面中使用了 hello.jsp 页面中声明的变量 username。程序的运行结果如图 3-2 所示。

在使用 include 指令包含一个文件时需要遵循下列几个规则。

（1）在转换阶段不进行任何处理，这意味着 file 属性值不能是请求时属性表达式，因此下面的使用是非法的。

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

（2）不能通过 file 属性值向被包含的页面传递任何参数，因为请求参数是请求的一个属性，它在转换阶段没有任何意义。下面例子中的 file 属性值是非法的。

```
<%@ include file="other.jsp?name=Hacker" %>
```

（3）被包含的页面可能不能单独编译。清单 3.5 的文件就不能单独编译，因为它没有定义 username 变量。一般来说，最好避免这种依赖性，而使用隐含变量 pageContext 共享对象，通过使用 pageContext 的 setAttribute()和 getAttribute()实现。

3.3.3　taglib 指令

taglib 指令用来指定在 JSP 页面中使用标准标签或自定义标签的前缀与标签库的 URI，下面是 taglib 指令的例子。

```
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
```

对于指令的使用需要注意下面几个问题：

- 标签名、属性名及属性值都是大小写敏感的。
- 属性值必须使用一对单引号或双引号引起来。
- 在等号(=)与值之间不能有空格。

扫一扫

视频讲解

3.4　JSP 隐含变量

在 JSP 页面的转换阶段，容器在页面实现类的_jspService()方法中声明并初始化一些变量，可以在 JSP 页面的小脚本或表达式中直接使用这些变量。这些变量是由容器隐含声明的，所以一般被称为**隐含变量**或**隐含对象**(implicit objects)。表 3-3 给出了页面实现类中声明的 9 个隐含变量。

表 3-3　JSP 隐含变量

隐 含 变 量	类 或 接 口	说　　明
request	jakarta. servlet. http. HttpServletRequest 接口	引用页面的当前请求对象
response	jakarta. servlet. http. HttpServletResponse 接口	用来向客户发送一个响应
out	jakarta. servlet. jsp. JspWriter 类	引用页面输出流
page	java. lang. Object 类	引用页面的 Servlet 实例
application	jakarta. servlet. ServletContext 接口	引用 Web 应用程序上下文
session	jakarta. servlet. http. HttpSession 接口	引用用户会话
pageContext	jakarta. servlet. jsp. PageContext 类	引用页面上下文
config	jakarta. servlet. ServletConfig 接口	引用 Servlet 的配置对象
exception	java. lang. Throwable 类	引用异常对象，用来处理错误

下面详细介绍几个最常用的隐含变量的使用。

注意：这些隐含变量只能在 JSP 页面的 JSP 小脚本和 JSP 表达式中使用。

3.4.1　request 与 response 变量

request 和 response 分别是 HttpServletRequest 和 HttpServletResponse 类型的隐含变量，当页面实现类向客户提供服务时，它们作为参数传递给_jspService()方法。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，例如：

```
<%
   String uri = request.getRequestURI();
   response.setContentType("text/html;charset = UTF - 8");
%>

请求方法为:<% = request.getMethod() %><br>
请求 URI 为:<% = uri %><br>
协议为:<% = request.getProtocol() %>
```

用户可以在 JSP 小脚本中使用隐含变量，也可以在 JSP 表达式中使用隐含变量。

3.4.2　out 变量

out 是 jakarta. servlet. jsp. JspWriter 类型的隐含变量，JspWriter 类扩展了 java. io. Writer 并继承了所有重载的 write()方法，在此基础上还增加了自己的一组 print()和 println()

来打印输出所有的基本数据类型、字符串及用户定义的对象。用户可以在小脚本中直接使用out变量，也可以在表达式中间接地使用它产生HTML代码。

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

上面两行代码，在页面实现类中都使用out.print()语句输出。

3.4.3 application变量

application是jakarta.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（在第2章中曾讨论了ServletContext接口）。在Servlet中可以使用如下代码访问ServletContext对象，将today存储到ServletContext对象中。

```
LocalDate today = LocalDate.now();
ServletContext context = getServletContext();
context.setAttribute("today",today);
```

在JSP页面中，使用下面application对象的getAttribute()方法检索存储在上下文中的数据。

```
<% = application.getAttribute("today") %>
```

3.4.4 session变量

session是jakarta.servlet.http.HttpSession类型的隐含变量，它在JSP页面中表示HTTP会话对象。如果要使用会话对象，必须要求JSP页面参加会话，即要求将JSP页面的page指令的session属性值设置为true。

在默认情况下，session属性的值为true，所以即使没有指定page指令，该变量也会被声明并可以使用。下面的代码可以在页面中输出当前会话的ID。

```
会话 ID = <% = session.getId() %>
```

然而，如果明确地将page指令的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，例如：

```
<% @ page session = "false" %>
```

3.4.5 exception变量

如果一个页面是错误处理页面，即页面中包含下面的page指令。

```
<% @ page isErrorPage = "true" %>
```

则页面实现类中将声明一个exception隐含变量，它是java.lang.Throwable类型的隐含变量，被用来作为其他页面的错误处理器。为了使页面能够使用exception变量，必须在page指令中将isErrorPage的属性值设置为true，例如：

```
<% @ page isErrorPage = "true" %>
<html><body>
    页面发生了下面的错误:<br>
    <% = exception.toString() %>
</body></html>
```

在上述代码中，将page指令的isErrorPage属性设置为true，容器明确地定义了exception变量。该变量指向使用该页面作为错误处理器的页面抛出的未捕获的异常对象。

如果去掉第一行，容器将不会明确地定义 exception 变量，因为 isErrorPage 属性的默认值为 false，此时使用 exception 变量将产生错误。

3.4.6　config 变量

config 是 jakarta.servlet.ServletConfig 类型的隐含变量。在第 2 章曾介绍过，用户可以通过 DD 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。

类似地，用户也可以为 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。如果要实现这一点，应该首先在 DD 文件 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，这样 Servlet 的所有初始化参数就可以在 JSP 页面中通过 config 隐含变量使用。例如：

```
<servlet>
    <servlet-name>initTestServlet</servlet-name>
    <jsp-file>/initTest.jsp</jsp-file>
    <init-param>
        <param-name>company</param-name>
        <param-value>Beijing New Techonology CO.,LTD</param-value>
    </init-param>
    <init-param>
        <param-name>email</param-name>
        <param-value>smith@yahoo.com.cn</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>initTestServlet</servlet-name>
    <url-pattern>/initTest.jsp</url-pattern>
</servlet-mapping>
```

以上代码声明了一个名为 initTestServlet 的 Servlet 并将它映射到/initTest.jsp 文件，同时为该 Servlet 指定了 company 和 email 初始化参数，该参数可以在 initTest.jsp 文件中使用隐含变量 config 检索到。例如：

```
公司名称:<%=config.getInitParameter("company")%><br>
邮箱地址:<%=config.getInitParameter("email")%>
```

3.4.7　pageContext 变量

pageContext 是 jakarta.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)，它有下面 3 个作用。

(1) 存储隐含对象的引用。pageContext 对象管理在 JSP 页面中使用的所有其他对象，包括用户定义的对象和隐含对象，并且提供了一个访问方法来检索它们。如果用户查看 JSP 页面生成的 Servlet 代码，会看到 session、application、config 与 out 这些隐含变量是调用 pageContext 对象的相应方法得到的。

(2) 提供了在不同作用域内返回或设置属性的方便的方法。本书 3.7 节将对相关内容进行详细说明。

(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，它们的格式如下。

- 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()的功能相同。
- public void forward(String relativeURL)：将请求转发到参数指定的资源，与 RequestDispatcher 接口的 forward()的功能相同。

例如，从 Servlet 中将请求转发到另一个资源，需要写下面两行代码。

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

在 JSP 页面中，使用 pageContext 变量仅需要一行就可以完成上述功能。

```
<%
   pageContext.forward("other.jsp");
%>
```

3.5 JSP 动作

JSP 动作(action)是页面发给容器的命令，它指示容器在页面执行期间完成某种任务。动作的一般语法如下。

```
<prefix:actionName attribute-list/>
```

动作的表示类似于 HTML 的标签，在动作标签中需要指定前缀名(prefix)、动作名(actionName)和属性列表(attribute-list)，它是针对该动作的一个或多个属性-值对。

在 JSP 页面中可以使用 3 种动作，即 JSP 标准动作、标准标签库(JSTL)中的动作和用户自定义动作。

下面是常用的 JSP 标准动作。

- <jsp:include.../>：在当前页面中包含另一个页面的输出。
- <jsp:forward.../>：将请求转发到指定的页面。
- <jsp:useBean.../>：查找或创建一个 JavaBean 对象。
- <jsp:setProperty.../>：设置 JavaBean 对象的属性值。
- <jsp:getProperty.../>：返回 JavaBean 对象的属性值。

下面介绍<jsp:include>和<jsp:forward>两个动作，3.9 节将介绍有关 JavaBean 使用的 3 个动作。

3.5.1 <jsp:include>动作

在 3.3.2 节讨论了使用 include 指令实现静态包含，本节讨论使用 JSP 的<jsp:include>动作实现动态包含。**动态包含**是在请求时将另一个页面的**输出**包含到主页面的输出中。该动作的格式如下。

```
<jsp:include page="relativeURL" flush="true|false"/>
```

这里 page 属性是必需的，其值必须是一个相对 URL，它指向任何静态或动态 Web 组件，包括 JSP 页面、Servlet 等。可选的 flush 属性用于指定在将控制转向被包含页面之前是否刷新主页面。如果当前 JSP 页面被缓冲，那么在把输出流传递给被包含组件之前应该刷新缓冲区。flush 属性的默认值为 false。

例如，下面的动作指示容器把另一个 JSP 页面 copyright.jsp 的输出包含在当前 JSP 页面

的输出中。

```
<jsp:include page = "copyright.jsp" flush = "true"/>
```

在功能上<jsp:include>动作的语义与 RequestDispatcher 接口的 include()方法的语义相同,因此在 Servlet 中使用下面的代码实现包含。

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

在 JSP 页面还可以使用下面的结构实现动态包含,就是在脚本中使用 pageContext.include()方法包含一个 JSP 页面的输出。

```
<%
    pageContext.include("other.jsp");
 %>
```

<jsp:include>动作的 page 属性的值可以是请求时属性表达式,例如:

```
<%! String pageURL = "other.jsp"; %>
 <jsp:include page = "<% = pageURL %>"/>
```

1. 使用<jsp:param>传递参数

在<jsp:include>动作中可以使用<jsp:param/>向被包含的页面传递参数。下面的代码向 somePage.jsp 页面传递两个参数:

```
<jsp:include page = "somePage.jsp">
    <jsp:param name = "name1" value = "value1"/>
    <jsp:param name = "name2" value = "value2"/>
</jsp:include>
```

在<jsp:include>元素中可以嵌入任意多个<jsp:param>元素。value 属性的值也可以像下面这样使用请求时属性表达式来指定。

```
<jsp:include page = "somePage.jsp">
    <jsp:param name = "name1" value = "<% = someExpr1 %>"/>
    <jsp:param name = "name2" value = "<% = someExpr2 %>"/>
</jsp:include>
```

通过<jsp:param>动作传递的"名-值"对保存在 request 对象中并且只能由被包含的组件使用,在被包含的页面中使用 request 隐含对象的 getParameter()获得传递来的参数。这些参数的作用域是被包含的页面,在被包含的组件完成处理后,容器将从 request 对象中清除这些参数。

上面的例子使用的是<jsp:include>动作,这里的讨论也适用于<jsp:forward>动作。

2. 与动态包含的组件共享对象

被包含的页面是单独执行的,因此它们不能共享在主页面中定义的变量和方法;它们处理的请求对象是相同的,因此可以共享属于请求作用域的对象。下面看清单 3.6 和清单 3.7 两个程序。

清单 3.6　hello2.jsp

```
<%@ page contentType = "text/html;charset = UTF - 8" %>
<html>
<head><title>Hello</title></head>
<body>
    <img src = "images/duke.gif">
    我的名字叫 Duke,你的名字叫什么?
    <form action = "" method = "post">
      <input type = "text" name = "username" size = "25">
      <input type = "submit" value = "提交">
```

```
        <input type="reset" value="重置">
    </form>
    <% String userName=request.getParameter("username");
       request.setAttribute("username",userName);
    %>
    <jsp:include page="response2.jsp"/>
</body>
</html>
```

清单 3.6 产生的输出结果与清单 3.4 产生的输出结果相同，但它使用了动态包含。主页面 hello2.jsp 通过调用 request.setAttribute()把 username 对象添加到请求作用域中，然后被包含的页面 response2.jsp 通过调用 request.getAttribute()检索该对象并使用表达式输出。

清单 3.7　response2.jsp

```
<%@ page contentType="text/html;charset=UTF-8" %>
<% String username=(String)request.getAttribute("username"); %>
<h4 style="color:blue">你好,<%=username%>!</h4>
```

这里，hello2.jsp 文件中的隐含变量 request 与 response2.jsp 文件中的隐含变量 request 是请求作用域内的同一个对象。对<jsp:forward>动作可以使用相同的机制。

除 request 对象外，用户还可以使用隐含变量 session 和 application 在被包含的页面中共享对象，但它们的作用域不同。例如，如果使用 application 代替 request，那么 username 对象就可以被多个客户使用。

3.5.2　<jsp:forward>动作

使用<jsp:forward>动作把请求转发到其他组件，然后由转发到的组件把响应发送给客户，该动作的格式如下。

```
<jsp:forward page="relativeURL"/>
```

page 属性的值为转发到的组件的相对 URL，它可以使用请求时属性表达式。<jsp:forward>动作与<jsp:include>动作的不同之处在于，当转发到的页面处理完输出后，并不将控制转回主页面。使用<jsp:forward>动作，主页面也不能包含任何输出。

下面的<jsp:forward>动作将控制转发到 other.jsp 页面。

```
<jsp:forward page="other.jsp"/>
```

在功能上<jsp:forward>的语义与 RequestDispatcher 接口的 forward()的语义相同，因此它的功能与在 Servlet 中实现请求转发的功能等价。

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

在 JSP 页面中使用<jsp:forward>标准动作实现的实际上是控制逻辑的转移。在 MVC 设计模式中，控制逻辑应该由控制器(Servlet)实现而不应该由视图(JSP 页面)实现，因此尽可能不要在 JSP 页面中使用<jsp:forward>动作转发请求。

扫一扫

视频讲解

3.6　案例学习：使用包含设计页面布局

Web 应用程序页面应该具有统一的视觉效果，或者说所有的页面都有同样的整体布局。一种比较典型的布局通常包含标题部分、脚注部分、菜单、广告区和主体实际内容部分。在设计这些页面时，如果在所有的页面中复制相同的代码，不仅不符合模块化设计原则，将来修改

布局也非常麻烦。使用 JSP 技术提供的 include 指令(<%@ include .../>)包含静态文件和使用 include 动作(<jsp:include .../>)包含动态资源就可以实现一致的页面布局。

清单 3.8 的 home.jsp 页面使用 标签和 include 指令实现页面布局。

清单 3.8　home.jsp

```
<%@ page contentType = "text/html; charset = UTF - 8" %>
<html>
<head>
<title>百斯特电子商城</title>
<link href = "css/style.css" rel = "stylesheet" type = "text/css"/>
</head>
<body>
    <div class = "container">
        <div class = "header">
            <%@ include file = "/WEB - INF/jsp/header.jsp" %>
        </div>
        <div class = "topmenu">
            <%@ include file = "/WEB - INF/jsp/topmenu.jsp" %></div>
        <div class = "mainContent" class = "clearfix">
            <div class = "leftmenu">
              <%@ include file = "/WEB - INF/jsp/leftmenu.jsp" %></div>
            <div class = "content">
              <%@ include file = "/WEB - INF/jsp/content.jsp" %></div>
        </div>
        <div class = "footer">
              <%@ include file = "/WEB - INF/jsp/footer.jsp" %></div>
    </div>
</body>
</html>
```

被包含的 JSP 文件存放在 WEB-INF/jsp 目录中,WEB-INF 中的资源只能被服务器访问,这样可以防止这些 JSP 页面被客户直接访问。访问 index.jsp 页面,输出结果如图 3-3 所示。

图 3-3　index.jsp 页面的运行结果

该页面使用了 CSS 样式进行布局,样式表文件 style.css 的代码见清单 3.9。

清单 3.9　style.css

```
@CHARSET "UTF - 8";
body,div,ul{
```

```
    margin:0;
    padding:0;
}
p{text-align:center;}
a:link{
    color:blue;
    text-decoration:none;
}
a:hover{
    background-color:gray;
}
.container {
    width:1004px;
    margin:0 auto;
}
.header {
    margin-bottom:5px;
}
.topmenu {
    margin-bottom:5px;
}
.mainContent {
    margin:0 0 5px 0;
}
.leftmenu {
    float:left; width:120px;
    padding:5px 0 5px 30px;
}
.leftmenu ul{
    list-style:none;
}
.leftmenu p{
    margin:0 0 10px 0;
}
.content {
    float:left; width:700px;
}
.footer {
   clear:both;
   height:60px;
}
```

清单 3.10～清单 3.14 分别是标题页面 header.jsp、顶部菜单页面 topmenu.jsp、左侧菜单页面 leftmenu.jsp、主体内容页面 content.jsp 和页脚页面 footer.jsp 的代码。

清单 3.10　header.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java">
<script language="JavaScript" type="text/javascript">
   function register(){
      open("/helloweb/register.jsp","register");
   }
</script>
<p><img src="images/head.jpg" /></p>
```

清单 3.11　topmenu.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java">
<table border='0'>
<tr>
    <td><a href="/helloweb/index.jsp">【首页】</a></td>
```

```
    <td>
      <form action="login.do" method="post" name="login">
        用户名<input type="text" name="username" size="13"/>
        密　码 <input type="password" name="password" size="13"/>
          <input type="submit" name="submit" value="登　录">
        <input type="button" name="register" value="注　册"
                    onclick="check();">
      </form>
    </td>
    <td><a href="showOrder">我的订单</a>|</td>
    <td><a href="showCart">查看购物车</a></td>
</tr>
</table>
```

清单 3.12　leftmenu.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java">
<p><b>商品分类</b></p>
<ul>
   <li><a href="showProduct?category=101">手机数码</a></li>
   <li><a href="showProduct?category=102">家用电器</a></li>
   <li><a href="showProduct?category=103">汽车用品</a></li>
   <li><a href="showProduct?category=104">服饰鞋帽</a></li>
   <li><a href="showProduct?category=105">运动健康</a></li>
</ul>
```

清单 3.13　content.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java">
<table border="0">
   <tr><td colspan="2">
      <b><i>${sessionScope.message}</i></b></td>
   </tr>
   <tr>
      <td colspan="4">百斯特 11.11!手机价格真正低,买华为手机送苹果 13!</td>
   </tr>
   <tr>
      <td width=20%><img src="images/phone.jpg" width=100></td>
      <td><p style="text-indent:2em">HUAWEI Mate 40 Pro 麒麟 9000 5G SoC 芯片
       超感知徕卡电影影像 8GB+256GB 秘银色 5G 全网通 特价:5288 元</p>
      <img src="images/gw.jpg">
      </td>

      <td width=20%><img src="images/comp.jpg" width=120></td>
      <td><p style="text-indent:2em">联想(Lenovo)G460AL-ITH 14.0 英寸
        笔记本电脑(i8-370M 2G 500G 512 独显 DVD 刻录 摄像头 Win7)特价:3199 元!</p>
      <img src="images/gw.jpg">
      </td>
   </tr>
</table>
```

清单 3.14　footer.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java">
<hr/>
<p>关于我们|联系我们|人才招聘|友情链接</p>
<p>版权所有 &copy; 2024 百斯特电子商城公司,电话:8899123.</p>
```

在上面这些被包含的文件中没有使用<html>和<body>等标签。实际上,它们不是完整的页面,而是页面片段,因此文件名也可以不使用.jsp 作为扩展名,而是可以使用任何的扩展名,如.htmlf、.jspf 等。

由于被包含的文件是由服务器访问的，所以可以将被包含的文件存放到 Web 应用程序的 WEB-INF 目录中，这样可以防止用户直接访问被包含的文件。

3.7 错误处理

与任何 Java 程序一样，Web 应用程序在执行过程中可能发生各种错误。例如，网络问题可能引发 SQLException 异常，在读取文件时可能因为文件损坏发生 IOException 异常。如果这些异常没有被适当处理，Web 容器将产生一个 Internal Server Error 页面，给用户显示一个长长的栈跟踪。这在产品环境下通常是不可以接受的。

3.7.1 声明式错误处理

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 文件中配置错误页面需要使用< error-page >元素，它有以下 3 个子元素。

- < error-code >：指定一个 HTTP 错误代码，如 404。
- < exception-type >：指定一种 Java 异常类型(使用完全限定名)。
- < location >：指定要被显示的资源位置。该元素值必须以“/”开头。

下面的代码为 HTTP 的状态码 404 配置了一个错误处理页面。

```
< error - page >
    < error - code > 404 </error - code >
    < location >/errors/notFoundError.html </location >
</error - page >
```

下面的代码声明了一个处理 SQLException 异常的错误页面。

```
< error - page >
    < exception - type > java.sql.SQLException </exception - type >
    < location >/errors/sqlError.html </location >
</error - page >
```

另外，还可以像下面这样声明一个更通用的处理页面。

```
< error - page >
    < exception - type > java.lang.Throwable </exception - type >
    < location >/errors/errorPage.html </location >
</error - page >
```

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

注意：在< error-page >元素中，< error-code >和< exception-type >不能同时使用。< location >元素的值必须以斜杠(/)开头，它是相对于 Web 应用程序的上下文根目录。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。

3.7.2 使用 Servlet 和 JSP 页面处理错误

在前面的例子中使用 HTML 页面作为异常处理页面，HTML 是静态页面，不能为用户提供有关异常的信息。可以使用 Servlet 或 JSP 作为异常处理页面，下面的代码使用 Servlet 处

理 403 错误码，使用 JSP 页面处理 SQLException 异常。

```
<error-page>
    <error-code>403</error-code>
    <location>/errorHandler-servlet</location>
</error-page>

<error-page>
    <exception-type>java.sql.SQLException</exception-type>
    <location>/errors/sqlError.jsp</location>
</error-page>
```

为了在异常处理的 Servlet 或 JSP 页面中分析异常原因并产生详细的响应信息，Web 容器将控制转发到错误页面前在请求对象(request)中定义了若干属性。

- jakarta.servlet.error.status_code：类型为 java.lang.Integer，该属性包含错误的状态码值。
- jakarta.servlet.error.exception_type：类型为 java.lang.Class，该属性包含未捕获的异常的 Class 对象。
- jakarta.servlet.error.message：类型为 java.lang.String，该属性包含在 sendError()方法中指定的消息，或包含在未捕获的异常对象中的消息。
- jakarta.servlet.error.exception：类型为 java.lang.Throwable，该属性包含未捕获的异常对象。
- jakarta.servlet.error.request_uri：类型为 java.lang.String，该属性包含当前请求的 URI。
- jakarta.servlet.error.servlet_name：类型为 java.lang.String，该属性包含引起错误的 Servlet 名。

下面的代码显示了 MyErrorHandlerServlet 的 doGet()方法，它使用这些属性在产生的错误页面中包含有用的错误信息。

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                      throws ServletException,IOException{
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out=response.getWriter();
    out.println("<html>");
    out.println("<head><title>Error Demo</title></head>");
    out.println("<boy>");
    String code="" +
            request.getAttribute("jakarta.servlet.error.status_code");
    if("403".equals(code){
      out.println("<h3>对不起,您无权访问该页面!</h3>");
      out.println("<h3>请登录系统!</h3>");
     }else{
      out.println("<h3>对不起,我们无法处理您的请求!</h3>");
      out.println("请将该错误报告给管理员 admin@xyz.com! " +
            request.getAttribute("jakatar.servlet.error.request_uri"));
    }
    out.println("</body>");
    out.println("</html>");
}
```

上述 Servlet 根据错误码产生一个自定义的 HTML 页面。

除了标准的异常，Web 应用程序还可能定义自己的业务逻辑异常。例如，在银行应用中

可能定义 InsufficientFundsException 表示资金不足异常和 InvalidTransactionException 表示非法交易异常，它们用来表示通常的错误条件。同样需要解析这些异常中的消息，并把这些消息展示给用户。

3.8 作用域对象

扫一扫

视频讲解

在 Web 应用中经常需要将数据存储到某个作用域中，然后在 JSP 页面中使用它们。**作用域**(scope)有应用作用域、会话作用域、请求作用域和页面作用域 4 种，它们的类型分别是 ServletContext、HttpSession、HttpServletRequest 和 PageContext，这 4 种作用域如表 3-4 所示。

表 3-4 JSP 作用域

作 用 域 名	对应的对象	存在性和可访问性
应用作用域	application	在整个 Web 应用程序中有效
会话作用域	session	在一个用户会话范围内有效
请求作用域	request	在用户的请求和转发的请求内有效
页面作用域	pageContext	只在当前的页面(转换单元)内有效

在 JSP 页面中，用户定义的对象存储在这 4 种作用域的一种之中，这些作用域定义了对象的存在性和在 JSP 页面中的可访问性。在作用域中，应用作用域的应用范围最大，页面作用域的应用范围最小。

3.8.1 应用作用域

存储在应用作用域的对象可以被 Web 应用程序的所有组件共享，并且在应用程序的生命周期内都可以访问。这些对象是通过 ServletContext 实例作为“属性-值”对维护的。在 JSP 页面中，该实例可以通过隐含对象 application 访问。因此，如果要在应用程序级共享对象，可以使用 ServletContext 接口的 setAttribute()和 getAttribute()方法。例如，在 Servlet 中使用下面的代码将对象存储在应用作用域中。

```
User user = new User("张大海","12345");
ServletContext context = getServletContext();
context.setAttribute("user",user);
```

在 JSP 页面中可以使用脚本(不推荐)访问应用作用域中的数据，也可以使用 EL 表达式访问应用作用域中的数据，例如：

```
${applicationScope.user}
```

3.8.2 会话作用域

存储在会话作用域的对象可以被属于一个用户会话的所有请求共享，并且只能在会话有效时才可以被访问。这些对象是通过 HttpSession 类的一个实例作为“属性-值”对维护的。在 JSP 页面中，该实例可以通过隐含对象 session 访问。因此，如果要在会话级共享对象，可以使用 HttpSession 接口的 setAttribute()和 getAttribute()方法。

在购物车应用中，用户的购物车对象就应该存放在会话作用域中，它在整个用户会话中共享。

```
HttpSession session = request.getSession(true);
```

```
//从会话对象中检索购物车
ShoppingCart cart = (ShoppingCart)session.getAttribute("cart");
if (cart == null) {
    cart = new ShoppingCart();
    //将购物车存储到会话对象中
    session.setAttribute("cart",cart);
}
```

存储在会话作用域中的数据在 JSP 页面中通过 session 隐含对象访问，例如：

```
<%
  //从会话作用域中取出购物车对象 cart
  ShoppingCart cart = (ShoppingCart) session.getAttribute("cart");
  //从购物车中取出每件商品并存储在 ArrayList 中
  ArrayList<GoodsItem> items = new ArrayList<GoodsItem>(cart.getItems());
%>
```

3.8.3 请求作用域

存储在请求作用域的对象可以被处理同一个请求的所有组件共享，并且仅在该请求被服务期间可以被访问。这些对象是由 HttpServletRequest 对象作为“属性-值”对维护的。在 JSP 页面中，该实例是通过隐含对象 request 的形式被使用的。通常，在 Servlet 中使用请求对象的 setAttribute()将一个对象存储到请求作用域中，然后将请求转发到 JSP 页面，在 JSP 页面中通过脚本或 EL 取出作用域中的对象。

例如，下面的代码在 Servlet 中创建一个 User 对象并存储在请求作用域中，然后将请求转发到 valid-servlet 去验证。

```
User user = new User();
user.setName(request.getParameter("name"));
user.setPassword(request.getParameter("password"));
request.setAttribute("user",user);
RequestDispatcher rd = request.getRequestDispatcher("/valid-servlet");
rd.forward(request,response);
```

在 ValidateServlet 中使用下面的代码验证用户的合法性。

```
User user = (User) request.getAttribute("user");
if (isValid(user)){            //验证用户是否合法
    request.removeAttribute("user");
    session.setAttribute("user",user);
    dispatchUrl = "account.jsp";
}else{
    dispatchUrl = "loginError.jsp";
}
RequestDispatcher rd = request.getRequestDispatcher(dispatchUrl);
rd.forward(request,response);
```

这里用 isValid()方法验证用户是否合法，若合法，将用户对象存储在会话作用域中并将请求转发给 account.jsp 页面，否则将请求转发到 loginError.jsp。

3.8.4 页面作用域

存储在页面作用域的对象只能在它们所定义的转换单元中被访问。它们不能存在于一个转换单元的单个请求处理之外。这些对象是由 PageContext 抽象类的一个具体子类的一个实例作为“属性-值”对维护的。在 JSP 页面中，该实例可以通过隐含对象 pageContext 访问。

为了在页面作用域中共享对象，可以使用 jakarta.servlet.jsp.PageContext 定义的

setAttribute()和 getAttribute()方法。下面的代码设置一个页面作用域的属性。

```
<% Float one = new Float(42.5); %>
<% pageContext.setAttribute("number",one); %>
```

下面的代码获得一个页面作用域的属性。

```
<% = pageContext.getAttribute("number") %>
```

在 PageContext 类中还定义了几个常量和其他属性处理方法，使用它们可以方便地处理不同作用域的属性。

扫一扫

视频讲解

3.9 JavaBean

JavaBean 是 Java 平台的组件技术，在 Java Web 开发中常用 JavaBean 来存放数据、封装业务逻辑等，从而很好地实现业务逻辑和表示逻辑的分离，使系统具有更好的健壮性和灵活性。对程序员来说，JavaBean 最大的好处是可以实现代码的重用，另外对程序的易维护性等也有很大的意义。

本节首先介绍 JavaBean 规范，然后重点介绍如何在 JSP 页面中使用 JavaBean 的 3 个动作，分别是<jsp:useBean>动作、<jsp:setProperty>动作和<jsp:getProperty>动作。

3.9.1 JavaBean 规范

JavaBean 是用 Java 语言定义的类，这种类的设计需要遵循 JavaBean 规范的有关约定。任何满足下面 3 个要求的 Java 类都可以作为 JavaBean 使用。

(1) JavaBean 应该是 public 类，且实现 java.io.Serializable 接口。该类应该提供无参数的 public 构造方法，通过定义不带参数的构造方法或使用默认的构造方法均可以满足这个要求。

(2) JavaBean 类的成员变量一般称为**属性**(property)。每个属性的访问权限一般定义为 private，而不是 public。注意，属性名必须以小写字母开头。

(3) 每个属性通常定义两个 public 方法，一个是访问方法，另一个是修改方法，使用它们访问和修改 JavaBean 的属性值。

除了访问方法和修改方法，在 JavaBean 类中还可以定义其他方法实现某种业务逻辑，也可以只为某个属性定义访问方法，这样的属性就是只读属性。

清单 3.15 中的 Customer 类是一个 JavaBean，它使用 3 个 private 属性封装了客户信息，并提供了访问和修改这些信息的方法。

清单 3.15 Customer.java

```
package com.boda.domain;

import java.io.Serializable;

public class Customer implements Serializable{
    private String name;
    private String email;
    private String phone;

    public Customer(){}

    public Customer(String name,String email,String phone){
        this.name = name;
```

```
        this.email = email;
        this.phone = phone;
    }

    public String getName(){
        return this.name;
    }

    public String getEmail(){
        return this.email;
    }

    public String getPhone(){
        return this.phone;
    }

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

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

    public void setPhone(String phone){
        this.phone = phone;
    }
}
```

该类定义了 3 个属性，为其定义了无参数构造方法和带参数构造方法，还为该类的每个属性定义了 getter 方法和 setter 方法。

3.9.2 使用 Lombok 库

在清单 3.15 定义的 Customer 类中不仅定义了属性，还定义了构造方法和每个属性的访问方法及修改方法，这就需要程序员编写很多行模板代码。使用 Lombok 库的注解标注类可以避免编写这些模板代码，需要在 pom.xml 文件中添加下面的依赖。

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.24</version>
</dependency>
```

在 JavaBean 类的定义中就可以使用@Data 等注解标注类，之后在运行时系统将自动添加访问方法和修改方法。另外，也可以通过注解添加无参数构造方法和带参数构造方法。

下面使用 Lombok 库的注解标注 Customer 类。

```
package com.boda.domain;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.io.Serializable;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Customer implements Serializable{
    private String name;
```

```
    private String email;
    private String phone;
}
```

在上述代码中@Data注解用于生成属性的访问方法和修改方法，@NoArgsConstructor注解用于生成无参数构造方法，@AllArgsConstructor注解用于生成带所有参数的构造方法。

注意：在IntelliJ IDEA中使用Lombok库需要安装Lombok插件。

3.9.3 <jsp:useBean>动作

<jsp:useBean>动作用来在指定的作用域中查找或创建一个bean实例，一般格式如下。

```
<jsp:useBean id = "beanName" class = "package.class"
             scope = "page|request|session|application"
    其他元素
</jsp:useBean>
```

id属性用来唯一标识一个bean实例。在JSP页面的实现类中，id的值被作为Java语言的变量，因此可以在JSP页面的表达式和小脚本中使用该变量。

class属性指定创建bean实例的Java类。如果在指定的作用域中找不到一个现存的bean实例，将使用class属性指定的类创建一个bean实例。如果该类属于某个包，则必须指定类的全名，如com.boda.xy.Customer。

scope属性指定bean实例的作用域，该属性的取值可以是page、request、session或application，它们分别表示页面作用域、请求作用域、会话作用域和应用作用域。该属性是可选的，默认值为page作用域。如果将page指令的session属性设置为false，则bean不能在JSP页面中使用会话作用域。

下面的动作使用id、class和scope属性声明一个JavaBean。

```
<jsp:useBean id = "customer" class = "com.boda.xy.Customer" scope = "session"/>
```

当JSP页面执行到该动作时，容器在会话(session)作用域中查找或创建bean实例，并用customer引用指向它。这个过程与下面的代码等价。

```
Customer customer = (Customer)session.getAttribute("customer");
if (customer == null){
    customer = new Customer();
    session.setAttribute("customer",customer);
}
```

3.9.4 <jsp:setProperty>动作

<jsp:setProperty>动作用来给bean实例的属性赋值，它的格式如下。

```
<jsp:setProperty name = "beanName" property = "propertyName"
                 value = "{string|<% = expression %>}"/>
```

name属性用来标识一个bean实例，该实例必须是使用<jsp:useBean>动作声明的，并且name属性值必须与<jsp:useBean>动作中指定的id属性值相同。该属性是必需的。

property属性指定要设置值的bean实例的属性，容器将根据指定的bean的属性调用适当的setXxx()，因此该属性也是必需的。value属性指定属性值。

另外，还可以使用param属性指定请求参数名，如果请求中包含指定的参数，那么使用该参数值来设置bean的属性值。

提示：在MVC设计模式中不建议在页面中使用<jsp:setProperty>动作设置JavaBean

的属性值，而应该在 Servlet 或控制器中设置属性值，在页面中只显示属性值。

3.9.5　＜jsp:getProperty＞动作

＜jsp:getProperty＞动作检索并向输出流中打印 bean 的属性值，它的语法如下，非常简单。

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

该动作只有 name 和 property 两个属性，并且都是必需的。name 属性指定 bean 实例名，property 属性指定要输出的属性名。

下面的动作指示容器打印 customer 的 email 和 phone 属性值。

```
<jsp:getProperty name="customer" property="email"/>
<jsp:getProperty name="customer" property="phone"/>
```

下面通过清单 3.16 说明这几个动作的使用。如果要在 JSP 页面中使用 JavaBean，可以通过 JSP 标准动作＜jsp:useBean＞创建类的一个实例，JavaBean 类的实例一般称为一个 **bean**。

input-customer.jsp 页面接收用户输入的信息，然后将控制转到 CustomerServlet，最后将请求转发到 display-customer.jsp 页面。

清单 3.16　input-customer.jsp

```
<%@ page contentType="text/html;charset=UTF-8" %>
<html>
<head><title>输入客户信息</title></head>
<body>
<h4>输入客户信息</h4>
<form action="customer-servlet" method="post">
 <table>
  <tr><td>客户名:</td><td><input type="text" name="name"></td></tr>
  <tr><td>邮箱地址:</td><td><input type="text" name="email"></td></tr>
  <tr><td>电话:</td><td><input type="text" name="phone"></td></tr>
  <tr><td><input type="submit" value="确定"></td>
      <td><input type="reset" value="重置"></td>
  </tr>
</table>
</form>
</body>
</html>
```

清单 3.17 给出了如何在 Servlet 代码中创建 JavaBean 类的实例，以及如何使用作用域对象共享它们，可以直接在 Servlet 中使用 JavaBean，并且可以在 JSP 页面和 Servlet 中共享 bean 实例。

清单 3.17　CustomerServlet.java

```
package com.boda.xy;

import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;
import com.boda.domain.Customer;

@WebServlet(name="customerServlet",value={"/customer-servlet"})
public class CustomerServlet extends HttpServlet {
   public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
            throws java.io.IOException,ServletException {
      String name=request.getParameter("name");
      String email=request.getParameter("email");
      String phone=request.getParameter("phone");
```

```
        Customer customer = new Customer(name,email,phone);

        HttpSession session = request.getSession();
        session.setAttribute("customer",customer);
        response.sendRedirect("display - customer.jsp");
    }
}
```

这个例子说明在 Servlet 中可以把 JavaBean 实例存储到作用域对象中。

清单 3.18 创建的页面在会话作用域内查找 customer 的一个实例，并用表格的形式打印出它的属性值。

清单 3.18　display-customer.jsp

```
<%@ page contentType = "text/html;charset = UTF - 8" language = "java" %>
<jsp:useBean id = "customer"
             class = "com.boda.domain.Customer" scope = "session"/>
<html>
<head><title>显示客户信息</title></head>
<body>
<h4>客户信息如下</h4>
<table border = "1">
<tr>
   <td>客户名:</td>
   <td><jsp:getProperty name = "customer" property = "name"/></td>
</tr>
<tr>
   <td>邮箱地址:</td>
   <td><jsp:getProperty name = "customer" property = "email"/></td>
</tr>
<tr>
   <td>电话:</td>
   <td><jsp:getProperty name = "customer" property = "phone"/></td>
</tr>
</table>
</body></html>
```

该页面首先在会话作用域内查找名为 customer 的 bean 实例，如果找到，将输出 bean 实例的各属性值。

在 JSP 页面中使用 JavaBean，早期的方法是使用 3 个 JSP 标准动作实现的，它们分别是<jsp:useBean>动作、<jsp:setProperty>动作和<jsp:getProperty>动作。在 JSP 2.0 中不推荐用这种方法使用 JavaBean，而是使用 EL 表达式访问 JavaBean 对象，例如：

```
${sessionScope.customer.email}
${sessionScope.customer.phone}
```

表达式语言将在第 4 章中详细讨论。

3.10　MVC 设计模式

Sun 公司在推出 JSP 技术以后提出了建立 Web 应用程序的两种体系结构方法，这两种方法分别称为模型 1 和模型 2，二者的差别在于处理请求的方式不同。

3.10.1　模型 1 介绍

在模型 1 体系结构中没有一个核心组件控制应用程序的工作流程，所有的业务处理都使用 JavaBean 实现。每个请求的目标都是 JSP 页面。JSP 页面负责完成请求需要的所有任务，

其中包括验证用户、使用 JavaBean 访问数据库及管理用户状态等。最后，响应结果通过 JSP 页面发送给用户。

该结构具有严重的缺点。首先，它需要将实现业务逻辑的大量 Java 代码嵌入 JSP 页面中，这对不熟悉服务器端编程的 Web 页面设计人员不友好；其次，这种方法并不具有代码可重用性。例如，为一个 JSP 页面编写的用户验证代码无法在其他 JSP 页面中重用。

3.10.2　模型 2 介绍

模型 2 体系结构如图 3-4 所示，这种体系结构又称为 MVC(Model-View-Controller)设计模式。在这种结构中，将 Web 组件分为模型(Model)、视图(View)和控制器(Controller)，每种组件完成各自的任务。在这种结构中，所有请求的目标都是 Servlet 或 Filter，它充当应用程序的控制器。Servlet 分析请求并将响应所需要的数据收集到 JavaBean 对象或 POJO 对象中，该对象作为应用程序的模型。数据可以从数据库中检索，也可以存储到数据库中。最后，Servlet 控制器将请求转发到 JSP 页面。这些页面使用存储在 JavaBean 中的数据产生响应。JSP 页面构成了应用程序的视图。

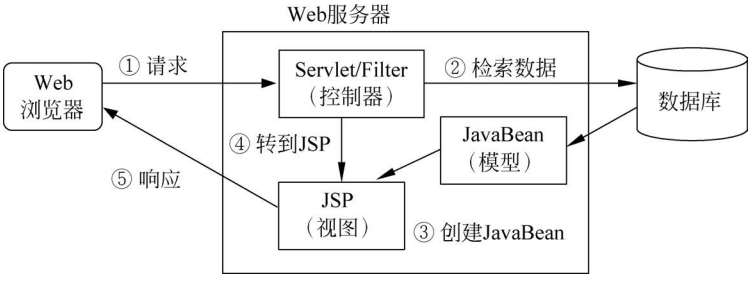

图 3-4　模型 2 体系结构

该模型的最大优点是将业务逻辑和数据访问从表示层分离出来。控制器提供了应用程序的单一入口点，提供了较清晰地实现安全性和状态管理的方法，并且这些组件可以根据需要实现重用。然后，根据客户的请求，控制器将请求转发给合适的表示组件，由该组件来响应客户。这使得 Web 页面开发人员可以只关注数据的表示，因为 JSP 页面不需要任何复杂的业务逻辑。

在 Web 应用系统开发中被广泛应用的 Spring MVC 框架就是基于模型 2 体系结构的。Spring MVC 对系统中各部分要完成的功能和职责有一个明确的划分，采用 Spring MVC 开发 Web 应用程序可以节省开发时间和费用，同时开发出来的系统易于维护。现在越来越多的 Web 应用系统开始采用 Spring MVC 框架开发。

3.10.3　实现 MVC 设计模式的一般步骤

使用 MVC 设计模式开发 Web 应用程序一般使用下面的步骤。

1. 定义 JavaBean 存储数据

在 Web 应用中通常使用 JavaBean 对象或实体类存放数据，在 JSP 页面作用域中取出数据，因此首先应该根据应用处理的实体设计合适的 JavaBean。例如，在订单应用中可能需要设计 Product、Customer、Orders、OrderItem 等 JavaBean 类。

2. 使用 Servlet 处理请求

在 MVC 模式中，使用 Servlet 或 Filter 充当控制器，从请求中读取请求信息(如表单数据)、创建 JavaBean 对象、执行业务逻辑，最后将请求转发到视图组件(JSP 页面)。Servlet 通

常并不直接向客户输出数据。在控制器创建 JavaBean 对象后需要填写该对象的值，可以通过请求参数值或访问数据库得到有关数据。

3. 将结果存储到作用域中

在创建了与请求相关的数据并将数据存储到 JavaBean 对象中以后，接下来应该将这些对象存储到 JSP 页面能够访问的地方。在 Web 中主要可以在 3 个位置存储 JSP 页面所需的数据，它们是 HttpServletRequest 对象、HttpSession 对象和 ServletContext 对象。下面的代码创建 Customer 类对象并将其存储到会话作用域中。

```
Customer customer = new Customer(name,email,phone);
HttpSession session = request.getSession();
session.setAttribute("customer",customer);
```

4. 转发请求到 JSP 页面

在使用请求作用域共享数据时，应该使用 RequestDispatcher 对象的 forward()方法将请求转发到 JSP 页面。在使用 ServletContext 对象或请求对象的 getRequestDispatcher()方法获得 RequestDispatcher 对象以后，调用它的 forward()方法将控制转发到指定的组件。

在使用会话作用域共享数据时，使用响应对象的 sendRedirect()方法重定向可能更合适。

5. 从模型中提取数据

当请求到达 JSP 页面以后，使用表达式语言和 JSTL 标准标签库等从模型中取出数据，在 JSP 页面中显示。例如，下面的代码显示客户信息。

```
客户名：${customer.name}<br>
客户地址：${customer.address}
```

提示：建议使用 EL 和 JSTL 输出数据，不建议使用<jsp:getProperty>输出模型数据，因为 EL 更简单、方便。第 4 章学习表达式语言和 JSTL。

本章小结

本章讨论了在 JSP 页面中使用指令、声明、小脚本、表达式、动作及注释等语法元素。JSP 页面在其生命周期中要经历 7 个阶段，前 4 个阶段是页面转换阶段、编译阶段、加载和实例化阶段，后 3 个阶段是初始化阶段、提供服务阶段和销毁阶段。

在 JSP 页面中可以使用的指令有 3 种，即 page 指令、include 指令和 taglib 指令。在 Java Web 开发中可以有多种方式重用 Web 组件。在 JSP 页面中包含组件的内容或输出实现 Web 组件的重用有两种方式，即使用 include 指令的静态包含和使用<jsp:include>动作的动态包含。

JavaBean 是遵循一定规范的 Java 类，它在 JSP 页面中主要用来表示数据。MVC 设计模式是 Web 应用开发中最常使用的设计模式，它将系统中的组件分为模型、视图和控制器，实现了业务逻辑和表示逻辑的分离。

练习与实践

扫一扫 习题

扫一扫 自测题