Java Web 编程技术

第3章 JSP技术基础

主要内容

1. JSP语法概述
2. JSP页面生命周期
3. JSP脚本元素
4. JSP隐含变量
5. page指令属性
6. JSP组件包含
7. 作用域变量
8. JavaBeans
9. MVC设计模式
10. 错误处理

3.1 JSP语法概述

JSP语法

- JSP（JavaServer Pages）是一种在HTML页面中嵌入JSP元素的动态Web页面，它的主要用来实现表示逻辑。
- 在JSP页面中可以包含多种JSP元素，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该页面。

JSP语法

JSP页面元素	简要说明	标签语法
声明	声明变量与定义方法	<%! Java 声明 %>
小脚本	执行业务逻辑的Java代码	<% Java 代码 %>
表达式	出表达式的值	<%= 表达式 %>
指令	指定转换时向容器发出的指令	<%@ 指令 %>
动作（标签）	向容器提供请求时的指令	<jsp:标签名 />
EL表达式	JSP 2.0引进的表达式语言	${applicationScope.email}
注释	用于文档注释	<%-- 任何文本 --%>

JSP脚本元素

JSP脚本元素

- JSP声明。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。

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

- 在一个标签中声明了一个变量r和一个getArea()方法。

```
<%!
    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;
    }
%>
```

JSP脚本元素

- JSP小脚本。以“<%”开头，以“%>”结束的标签，可以包含任意数量Java语句。

```
<%
   date = LocalDate.now();
   out.println(date);
%>
```

JSP脚本元素

- JSP表达式。以“<%=”开头，以“%>”结束的标签，它作为Java语言表达式的占位符。

  今天的日期是：<%=date.toString() %>

JSP指令

- 指令（directive）向容器提供关于JSP页面的总体信息。指令是以“<%@”开头，以“%>”结束的标签。
- 指令有三种：page指令、include指令和taglib指令。
- 三种指令的语法格式如下：

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

JSP动作

- 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。动作的一般语法为：

  <prefix:actionName attribute-list />

- 在JSP页面中可以使用三种动作：JSP标准动作，标准标签库（JSTL）中的动作和用户自定义动作。
- JSP标准include动作：

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

表达式语言

- 表达式语言（Expression Language，EL）是JSP 2.0新增加的特性，它是一种可以在JSP页面中使用的简洁的数据访问语言。格式为：

  ${expression}

- 下面EL显示客户地址：

  ${pageContext.request.remoteAddr}

JSP注释

- JSP注释以“<%--”开头，以“--%>”结束的标签。注释不影响JSP页面的输出，它对用户理解代码很有帮助。格式为：

  <%-- 这里是JSP注释内容 --%>

  <% // 这里是Java 注释 %>

- 在JSP页面中可以用HTML风格注释。

  <!-- 这里是HTML 注释 -->

3.2 JSP页面生命周期

JSP页面实现类

- 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**。
- 以todayDate.jsp页面为例，在页面转换阶段Web容器自动将该文件转换成名为**todayDate_jsp.java**类文件，该文件是JSP页面实现类。
- 存放在安装目录的\work\Catalina\localhost\chapter03\org\apache\jsp目录中。

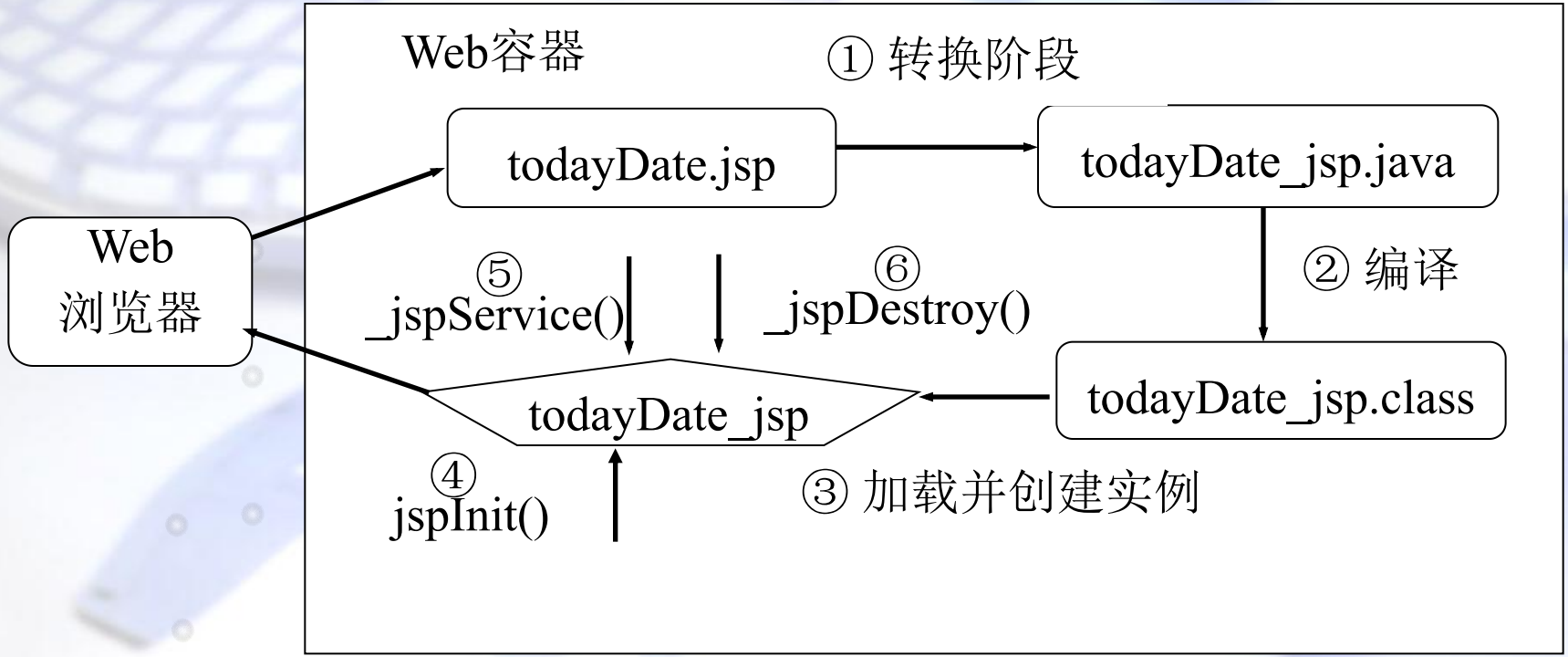
JSP页面实现类

- JSP页面中的所有元素都转换成页面实现类的对应代码。
- 如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Java语句。
- 模板文本和JSP表达式都使用out.write()方法打印输出。

JSP页面执行过程

- 客户首次访问页面时，Web容器执行该JSP页面要经过6个阶段。
- 前三个阶段将JSP页面转换成一个Servlet类并装载和创建该类实例，后三个阶段是初始化、提供服务和销毁阶段。

JSP页面执行过程


Web容器
① 转换阶段
todayDate.jsp
todayDate_jsp.java
Web
浏览器
② 编译
⑤
_jspService()
⑥
_jspDestroy()
todayDate_jsp
todayDate_jsp.class
④
jspInit()
③ 加载并创建实例

3.3 JSP脚本元素

脚本元素和隐含变量

- 在JSP页面中可以使用三种脚本元素：JSP声明、JSP小脚本和JSP表达式。
- 在页面实现类jspService()方法中声明一些变量，称为隐含变量，可以在小脚本和表达式中使用。

变量的声明及顺序

- 在JSP声明中定义的变量和方法都转换成页面实现类的成员，因此它们在页面中出现的顺序无关紧要。
- 程序3.5 area.jsp

```
半径为<%= r %>的圆的面积为： <%=area(r)%>
<%!
   double r = 0;                  // 声明一个变量r
   double area(double r) { // 声明求圆面积的方法
       return (int)(r * r * Math.PI*100)/100.0;
   }
%>
```

小脚本顺序

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

```
<% String s = s1 + s2; %>
<%! String s1 = "hello"; %>
<% String s2 = "world"; %>
<% out.print(s); %>
```

变量初始化

- 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。

```
<%! int a; %>        // a是成员变量，默认值0
<% int b; %>         // b是局部变量，使用前需赋值
a = <%= ++a %><br>
b = <%= ++b %><br>     // 该行代码不能被编译
```

使用条件和循环

- 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。

```
<%        …
if(isLoggedIn){
   out.print("<h3>欢迎您访问该页面！ </h3>");
}else{
   out.println("您还没有登录！ ");
   out.println("<a href='login.html'>登录</a>");
}
%>
```

使用条件和循环

- squareRoot.jsp。

```
<table border=1>
 <%
    for(int n = 10; n<=100;n+=10){
 %>
   <tr><td><%=n %></td>
   <td><%=Math.sqrt(n) %></td></tr>
<%
  }
%> </table>
```

10	3.162277660168379
20	4.47213595499958
30	5.477225575051661
40	6.324555320336759
50	7.071067811865475
60	7.745966692414834
70	8.366600265340756
80	8.94427190999916
90	9.486832980505138
100	10.0

请求时属性表达式

- JSP表达式并不总是输出到页面中，它们的也可以传递给JSP动作的属性。

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

请求时属性表达式

- 注意，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，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 %>" %>
```

3.4 JSP隐含变量

JSP隐含变量

- 在JSP页面的转换阶段，Web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，所以被称为隐含对象（implicit objects），这些对象可象变量一样使用，也被叫做隐含变量（implicit variables）。

request和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。

```
<%
   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()%>
```

out变量

- out是javax.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()。
- 可以在小脚本中直接使用它，也可以在表达式中间接使用它产生HTML代码。

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

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

application变量

- application是javax.servlet.ServletContext类型的隐含变量，是JSP页面所在的Web应用程序的上下文的引用。在Servlet中，可用如下代码访问ServletContext对象。

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

application变量

- 在JSP页面中，使用下面代码完成检索today对象。

```
<%! LocalDate now; %>
<%
  now = (LocalDate)(application.getAttribute("today"));
%>
今天日期是：<%=now%>
```

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（默认值）。

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

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。
- 有下面三个作用：

①存储隐含对象的引用。

②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()。

pageContext变量

- 例如，在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，通过使用pageContext变量仅需一行就可以完成上述功能。

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

3.5 page指令属性

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。

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

import属性

- import属性的功能类似于Java程序的import语句，它是将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，中间用逗号分隔。

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

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.*" %>
```

- 这里import属性的顺序也没有关系。另外，容器总是导入：

java.lang.*

javax.servlet.*

javax.servlet.http.*

javax.servlet.jsp.*

contentType和pageEncodingt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。MIME类型和字符集之间用分号分隔。

```
<%@ 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");
```

contentType和pageEncodingt属性

- 如果页面需要显示中文，字符集应该指定为UTF-8或GB18030，如下所示。

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

contentType和pageEncodingt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，JSP页面使用设置的字符集编码；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或GB18030，如下所示：

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

session属性

- session属性指示JSP页面是否参加HTTP会话，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，可以明确地加入下面一行：

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

errorPage和isErrorPage属性

- 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。使用isErrorPage属性指定当前页面是错误处理页面。
- helloUser.jsp

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

- errorHandler.jsp

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

3.6 JSP组件包含

静态包含：include指令

- JSP规范提供重用Web组件的机制，允许在JSP页面中包含另一个Web组件的内容或输出。通过两种方式实现：
  - 静态包含:include指令
  - 动态包含:include动作

静态包含：include指令

- 静态包含是将另一个文件的内容包含到当前JSP页面中。用include指令完成，语法为：

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

- file是include指令唯一的属性，它是指被包含的文件。文件路径以斜杠（/）开头，是相对于Web应用程序文档根目录的路径，路径不以斜杠开头，相对于当前JSP文件的路径。

动态包含：include动作

- 动态包含是通过JSP标准动作<jsp:include>实现的。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属性是必须的，其值必须是相对URL，并指向任何静态或动态Web组件，包括JSP页面、Servlet等。

动态包含：include动作

- 下面是include动作的使用：

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

- page属性的值可以是请求时表达式，例如：

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

使用包含设计页面布局

- Web应用程序界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。
- 下面的index.jsp页面使用 标签和include指令实现页面布局。

3.7 作用域对象

作用域对象

- 在Web应用中可以使用四个作用域对象，如表3-5所示。

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

应用作用域

- 存储在应用作用域的对象可被Web应用程序的所有组件共享并在应用程序生命期内都可以访问。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。
- 要在应用程序级共享对象，可以使用ServletContext接口的setAttribute()和getAttribute()。

应用作用域

- 在Servlet中可以使用ServletContext接口的setAttribute()方法将对象存储在应用作用域中。

```
String username = request.getParameter("username");
ServletContext context = getServletContext();
context.setAttribute("name", username);
```

- 在JSP页面中就可使用下面代码访问context中数据：

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

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只能在会话有效时才可被访问。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。
- 在会话级共享对象，使用HttpSession接口的setAttribute()。

会话作用域

- 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，它在整个的用户会话中共享。

```
HttpSession session = request.getSession(true);
ShoppingCart cart =
            (ShoppingCart)session.getAttribute("cart");
if (cart == null) {
    cart = new ShoppingCart();
    session.setAttribute("cart", cart);
}
```

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仅在该请求被服务期间可被访问。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。

请求作用域

- 下面代码在Servlet中创建一个User对象并存储在请求作用域中。

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

请求作用域

- 下面是valid.jsp文件。

```
<%
    User user = (User) request.getAttribute("user");
    if (isValid(user)){
        request.removeAttribute("user");
        session.setAttribute("user",user);
        pageContext.forward("account.jsp");
    }else{
        pageContext.forward("loginError.jsp");
    }
%>
```

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通过属性/值对维护的。
- 在JSP页面中，该实例可以通过隐含对象pageContext访问。

3.8 Java Beans

JavaBeans概述

- JavaBeans是用Java语言定义的类，是Java平台的组件技术，在Java Web开发中常用JavaBeans来存放数据。
- JavaBeans类非常简单，所以有时也称为POJO（Plain Old Java Object），普通的Java对象。在数据库应用中也叫实体类。

JavaBeans规范

- 定义JavaBeans类，需要遵循有关约定。

① JavaBeans应该是public类，且具有无参数的public构造方法。也可以定义带参数构造方法。

② 类的成员变量一般称为属性（property）。对每个属性访问权限一般定义为private。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（getter），一个是修改方法（setter），使用它们访问和修改JavaBeans的属性值。

JavaBeans规范

- 可以用JSP标准动作<jsp:useBean>创建JavaBeans类的一个实例，也可以在Servlet中创建类的实例。JavaBeans类的实例一般称为**bean**。
- 在JSP页面中使用JavaBeans是通过三个JSP标准动作实现，分别是：

<jsp:useBean>动作

<jsp:setProperty>动作

<jsp:getProperty>动作

使用<jsp:useBean>动作

- <jsp:useBean>动作一般格式如下：

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

- 功能：在指定的作用域中查找名为beanName的JavaBean，找到用id指向它，找不到用class属性指定的类创建一个bean实例。

使用<jsp:useBean>动作

- <jsp:useBean>动作示例:

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

- 等价代码:

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

<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=""“
           | property = "propertyName“
                          [param="paramName"]
           | property = “* “
           }
/>
```

<jsp:getProperty>动作

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

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

- 示例代码：

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

JavaBeans应用实例

- 下面示例定义一个名为Customer的JavaBeans存储客户，编写inputCustomer.jsp输入客户信息，然后将控制转到CustomerServlet，最后将请求转发到displayCustomer.jsp页面显示客户信息。

3.9 MVC设计模式

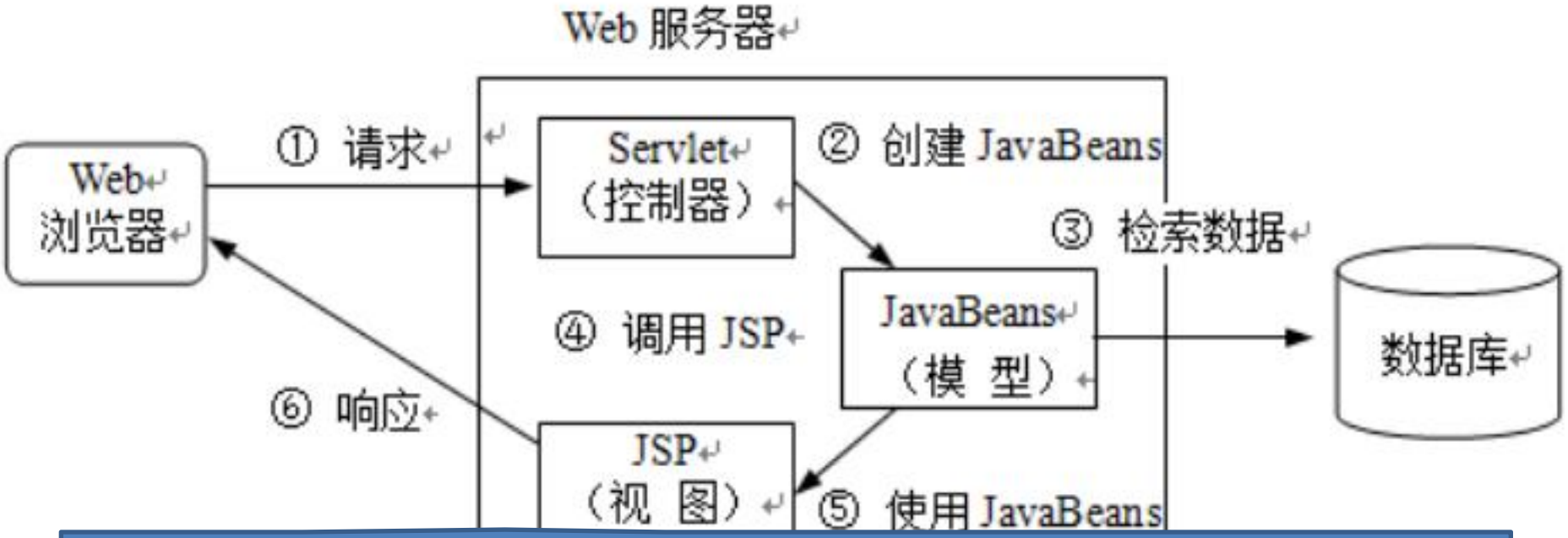
MVC设计模式

- Sun公司推出两种体系结构方法，这两种方法分别称为模型1和模型2。
- 模型1体系结构中，每个请求的目标都是JSP页面。JSP页面负责完成请求所需要的所有任务。
- 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代码嵌入到JSP页面中，这对不熟悉服务器端编程的Web页面设计人员将产生困难。第二，这种方法并不具有代码可重用性。

MVC设计模式

- 模型2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。
- 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。

MVC设计模式



目前流行的Spring MVC框架就是基于MVC体系结构。

实现MVC模式的一般步骤

1. 定义JavaBeans存储数据
2. 编写Servlet处理请求
3. 数据结果存储在作用域中
4. 转发请求到JSP页面
5. 从JavaBeans中提取数据

定义JavaBeans存储数据

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

编写Servlet处理请求

- 使用Servlet或Filter充当控制器功能，执行业务逻辑，它从请求中读取请求信息（如表单数据）、创建JavaBeans对象、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。控制器创建JavaBeans对象后需要填写对象的值。可以通过请求参数值或访问数据库得到有关数据。

数据结果存储在作用域中

- 创建了与请求相关的数据并将数据存储到JavaBeans对象中后，接下来应该将这些对象存储在JSP页面能够访问的地方。
- 在Web中主要可以在三个作用域对象存储JSP页面所需的数据，它们是HttpServletRequest对象、HttpSession对象和ServletContext对象。
- 这些存储位置对应<jsp:useBean>动作scope属性的三个非默认值：request、session和application。

数据结果存储在作用域中

- 下面代码创建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);
```

转发请求到JSP页面

- 使用请求作用域共享数据，应该用RequestDispatcher对象的forward()方法将请求转发到JSP页面。
- 获得RequestDispatcher对象可用ServletContext对象或请求对象的getRequestDispatcher()方法。
- 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。

从JavaBeans对象中提取数据

- 控制转到JSP页面后，使用<jsp:useBean>和<jsp:getProperty>动作提取JavaBeans数据，也可以使用表达式语言提取数据。
- 为保证JSP页面不会创建对象，应该使用动作：

```
<jsp:useBean id="customer" type="com.demo.Customer" />
```

- 而不应该使用动作：

```
<jsp:useBean id="customer" class="com.demo.Customer" />
```

3.10 错误处理

错误处理

- Web应用程序在执行过程中可能发生各种错误。例如，在读取文件时可能因为文件损坏发生IOException异常，网络问题可能引发SQLException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。
- 有多种错误处理方法：声明式错误处理和编程式错误处理。

声明式错误处理

- 前面章节我们介绍了使用page指令的errorPage属性指定一个错误处理页面，通过page指令的isErrorPage属性指定页面是错误处理页面。。
- 还可以在**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。

声明式错误处理

- 在web.xml文件中配置错误页面需要使用<error-page>元素，它有三个子元素：
  - <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。

声明式错误处理

- 下面代码为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>
```

声明式错误处理

- 下面代码声明了一个处理SQLException）的错误页面。

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

- 注意：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。

使用Servlet或JSP处理错误

- 下面代码使用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>/MyErrorHandlerServlet.do</location>
</error-page>
<error-page>
    <exception-type>java.sql.SQLException</exception-type>
    <location>/errorpages/sqlError.jsp</location>
</error-page>
```

编程式错误处理

- 处理Servlet中产生的异常最简单的方法是将代码包含在try-catch块中，在异常发生时通过catch块将错误消息发送给浏览器。

```
try{
....
}catch(SQLException e){
    response.sendError(
HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
"产生数据库连接错误，请联系管理员！"
    );    }
```

编程式错误处理

- HttpServletResponse接口定义了两个重载的sendError()方法，如下所示。

```
public void sendError (int sc)

public void sendError (int sc, String msg)
```

编程式错误处理

- 通过编程方式处理业务逻辑异常。例如，在银行应用中，可能定义
  - 表示资金不足异常InsufficientFundsException
  - 表示非法交易异常InvalidTransactionException

编程式错误处理

```
private double withdraw(String accounId, double amount)
                    throws InsufficientFundsException{
        double currentBalance = getBalance(accounted);
        if(currentBalance < amount)
            throw new
    InsufficientFundsException(currentBalance,amount);
        else{
     setNewBalance(accounted, currentBalance - amount);
     return currentBalance – amount;
}       }
```