Chapter 3 第3章 深入 JSP

3.1 JavaBean

3.1.1 编写 JavaBean

JavaBean 是用 Java 编写的组件，每个 JavaBean 实现了一个特定的功能，其他开发者可以通过 JSP 页面、Servlet 来使用这些对象。JavaBean 的优点在于提高代码的重用性，可以一次性编写，任何地方重用。

JavaBean 通常具有如下特点。

(1) JavaBean 类必是一个 public 类。

(2) JavaBean 的属性必须为 private 类型。

(3) 对 JavaBean 属性的操作通常由两个 public 类型的方法 getXxx()和 setXxx()完成。其中，getXxx()方法用于获取属性 xxx 的值，setXxx()方法用于设置属性 xxx 的值。

因为 JavaBean 本质上就是一个 Java 类，因此可以在 JavaBean 中定义各种方法。但在实际的 JSP 程序设计时，通常只在 JavaBean 中包括构造函数和属性的 get/set 方法。

代码 3-1 声明了一个 JavaBean 类 note.java。

代码 3-1　实现 JavaBean

```
//note.java 文件源代码
package notes.model;
public class note {
    String title;
    String msgPerson;
    String content;

    public String getTitle() {
        try {
            byte b[]=title.getBytes("ISO-8859-1");
            title=new String(b);
            return title;
        } catch (Exception e) {
            return title;
        }
    }
```

```
    public void setTitle(String title) {
        this.title=title;
    }
    public String getMsgPerson() {
        try {
            byte b[]=msgPerson.getBytes("ISO-8859-1");
            msgPerson=new String(b);
            return msgPerson;
        } catch (Exception e) {
            return msgPerson;
        }
    }
    public void setMsgPerson(String msgPerson) {
        this.msgPerson=msgPerson;
    }
    public String getContent() {
        try {
            byte b[]=content.getBytes("ISO-8859-1");
            content=new String(b);
            return content;
        } catch (Exception e) {
            return content;
        }
    }
    public void setContent(String content) {
        this.content=content;
    }
}
```

3.1.2　使用 JavaBean

要在 JSP 页面中使用编写好的 JavaBean，需要先在 JSP 页面中创建一个 JavaBean 的实例，然后才能通过该实例调用 JavaBean 定义的方法。使用 JavaBean 的方法有两种：一种是利用 JSP 定义的＜jsp：useBean＞动作、＜jsp：setProperty＞动作和＜jsp：getProperty＞动作；另一种是 new 操作显式地创建 JavaBean 实例。下面逐一介绍。

1. 标准的 JavaBean 操作动作

(1) ＜jsp:useBean＞动作

＜jsp:useBean＞用于在 JSP 页面中创建一个 JavaBean 实例，语法格式如下：

```
<jsp:useBean id="bean 名字" scope="page|request|session|application"
             class="类名">
```

其中：

① id 属性用于定义要加载的 JavaBean 实例的名称。使用该名称可以引用 JavaBean 的属性和方法。

② scope 属性定义 JavaBean 的生命周期，默认值为 page，表示该 JavaBean 只在当前

JSP 页面中可用。程序员可以根据需要将 scope 设置为其他值。

③ class 属性指定要加载的 JavaBean 的类文件路径。

(2) <jsp:setProperty>动作

<jsp:setProperty>动作和<jsp:useBean>动作配合使用,用于设置 JavaBean 的属性值,语法格式如下:

```
<jsp:setProperty name="beanname" property="*">
```

或

```
<jsp:setProperty name="beanname" property="属性名" [param="参数名"]>
```

或

```
<jsp:setProperty name="beanname" property="属性名" value="属性值">
```

其中:

① name 属性是通过<jsp:useBean>动作引入的 JavaBean 实例的名字。

② property 属性用于匹配 JavaBean 定义的属性。当取值为"*"时,JSP 容器会自动将 request 中各参数的值赋值给 JavaBean 实例中的同名属性;如果 request 中的属性名与 JavaBean 中的属性名字不相同,必须利用 param 指定 request 中的参数名。

③ 格式 3 利用 value 的值来设置 JavaBean 的属性值。

(3) <jsp:getProperty>动作

<jsp:getProperty>动作也必须和<jsp:useBean>动作配合使用,用于获取 JavaBean 中属性的值,所得到的值会自动转换成字符串类型,并通过输出流输出到 JSP 页面。格式如下:

```
<jsp:getProperty name="beanname" property="属性名">
```

各属性值的含义和<jsp:setProperty>动作中属性的含义相同。

代码 3-2 演示了如何利用标准的 JavaBean 操作动作访问代码 3-1 中定义的 JavaBean。

代码 3-2 使用 JavaBean

```
<!--post.jsp 文件源代码-->
<%@page language="java" import="java.util.*" pageEncoding="GB2312"%>
<html>
    <head>
        <title>留言板程序</title>
    </head>
    <body>
        <form action="showPost.jsp" mrthod="post">
          标题:<input type="text" name="title" size="80"/><br/>
          留言人:<input type="text" name="msgPerson" size="80"><br/>
          内容:
        <textarea rows="20" cols="70" name="content"></textarea><br/>
```

```
            <input type="submit" value="留言"/>
            <input type="reset" value="重写"/>
        </form>
    </body>
</html>

<!--showPost.jsp 文件源代码-->
<%@page language="java" contentType="text/html; charset=GB2312"%>
<html>
    <head>
        <title>显示留言</title>
    </head>
    <body>
    <jsp:useBean id="note" scope="session" class="notes.model.note"/>
    <jsp:setProperty property="*" name="note"/>
    <center><font color="red">您好,以下是您输入的留言信息:</font>
        <hr/>
        留言人:<jsp:getProperty property="msgPerson" name="note"/><br/>
        题目:<jsp:getProperty property="title" name="note"/><br/>
        内容:<jsp:getProperty property="content" name="note"/><br/>
    </center></body>
</html>
```

在 JSP 程序开发中,如果通过表单提交的数据中存在中文,则获取该数据后输出到页面时会显示乱码。因此,在输出获取的表单数据之前,必须进行转码操作。前面曾经利用 request. setCharacterEncoding("utf-8")进行过转码,但是当使用<jsp:setProperty>动作接收数据时,这种方法不会发挥作用,这时可以将该转码操作在 JavaBean 中实现,如代码 3-1 所示。

2. 利用 new 操作实例化 JavaBean

JavaBean 类可以像普通类一样通过 import 语句引入 JSP 文件,再利用 new 操作显式创建实例。

在代码 3-3 中,给出了显式实例化 JavaBean 的方法。这里重写了 note. java 类,更名为 note2. java。note2. java 不需要使用 getBytes("ISO-8859-1")进行转码。

代码 3-3　显式实例化 JavaBean

```
//note2.java 文件源代码
package notes.model;
public class note2 {
    String title;
    String msgPerson;
    String content;
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title=title;
```

```
    }
    public String getMsgPerson() {
        return msgPerson;
    }
    public void setMsgPerson(String msgPerson) {
        this.msgPerson=msgPerson;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content=content;
    }
}

<!-showPost2.jsp 文件源代码-->
<%@page language="java" import="notes.model.note2" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>显示留言</title>
    </head>
    <body>
      <%
          request.setCharacterEncoding("UTF-8");
          note2 n=new note2();
          n.setTitle(request.getParameter("title"));
          n.setMsgPerson(request.getParameter("msgPerson"));
          n.setContent(request.getParameter("content"));

          String title=n.getTitle();
          String msgPerson=n.getMsgPerson();
          String content=n.getContent();
      %>
      <center><font color="red">您好,以下是您输入的留言信息:</font>
      <hr/>
          留言人:<%=msgPerson %><br/>
          题目:<%=title%><br/>
          内容:<%=content%><br/>
      </center>
    </body>
</html>
```

3.2 Servlet

3.2.1 Servlet 概念

Servlet 是运行在 Web 服务器端的 Java 小程序,可以生成动态的网页。在基于

MVC 的 Web 程序开发中，通常利用 Servlet 接受客户请求，调用业务逻辑处理的方法，最后通过 HttpResponse 对象跳转到相应的页面。

Servlet 与 JSP 的功能相似，都是用于处理客户请求并返回响应。事实上，对于每一个 JSP 页面，在第一次被客户访问时，服务器都会将其转换成 Servlet。例如，名为 notes 的 Web 工程的根目录下有个 post.jsp 页面(见代码 3-2)，其对应的 Servlet 位于 Tomcat 主目录下的 work\Catalina\localhost\notes\org\apache\jsp 子目录中，名为 post_jsp.java，翻译后的内容如图 3-1 所示。

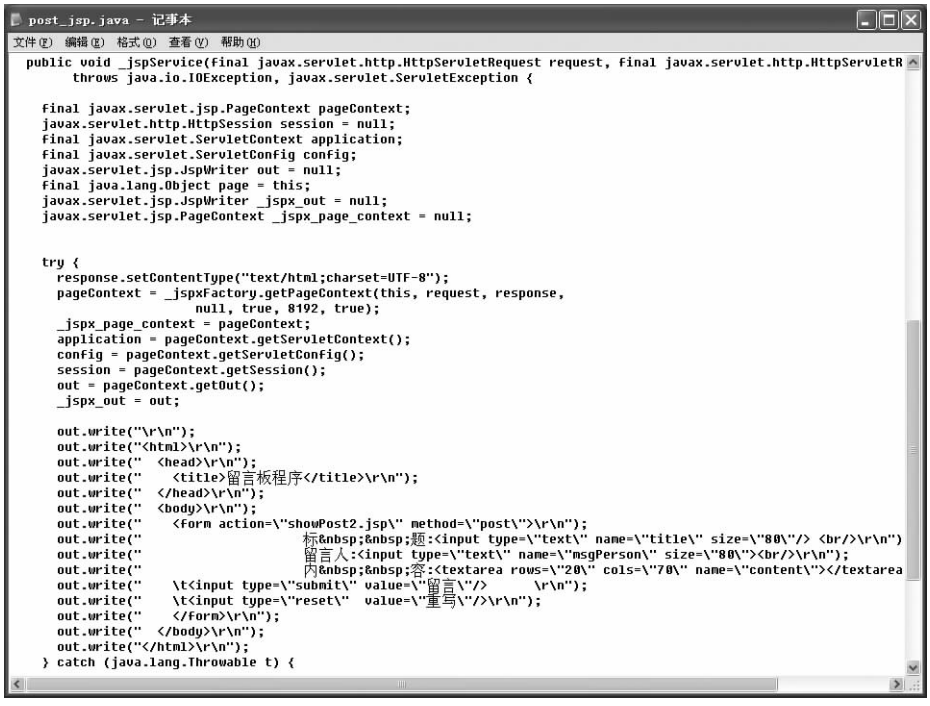

图 3-1　post.jsp 转换得到的 post_jsp.java 代码

Servlet 和 JSP 两者最大的区别就是：Servlet 以 Java 程序为主，输出 HTML 代码时需要使用 out.println 函数，也就是说，Java 中内嵌 HTML；而 JSP 以 HTML 布局为主，需要写 Java 代码时，在页面中直接插入 Java 代码，即 HTML 中内嵌 Java。

与 Servlet 相比，JSP 最大的优点是能够很方便地实现页面布局，显示复杂页面。

与 JSP 相比，Servlet 最大的优点在于后台处理，在业务处理和数据运算方面更稳健。另外，像动态生成验证码图像、显示邮箱的利用率等功能，通常都可以利用 Servlet 实现。

因此，在实际的项目开发时，可以利用 JSP 技术实现页面表现，利用 Servlet 技术完成逻辑处理，将二者有机地结合在一起使用。

3.2.2　Servlet 生命周期

Servlet 的生命周期主要由 4 个过程组成。

1. 装载和创建 Servlet 实例

Servlet 容器负责将 Servlet 类加载到 Java 虚拟机中并初始化。加载和实例化可以发生在 Web 服务程序启动时，也可以在客户端首次提出对 Servlet 服务请求时。

2. 初始化

当容器装载 Servlet 时，它会运行 Servlet 的 init()方法。在 Servlet 生命周期中，init()方法只被执行一次。为了提高系统性能，可以在 init()方法中缓存一些静态的数据或完成一些只需要执行一次的、耗时的操作，例如初始化数据库连接、获取配置信息等。在初始化不成功时，Servlet 实例将抛出 ServletException 异常或 UnavailableException 异常来通知 Web 容器。ServletException 异常用于指明一般的初始化失败，例如没有找到初始化参数；UnavailableException 异常用于通知容器该 Servlet 实例不可用。

3. 执行

初始化完毕，Servlet 就可以通过 service()等方法为客户提供服务请求。service()方法是 Servlet 的核心，能够获得与服务请求对象有关的信息，对请求进行处理，访问系统资源，然后将生成的响应封装在响应对象中，并回传给客户端。service()在执行的过程中可能会调用 doPost()、doGet()或程序员自定义的方法。在 service()方法执行期间，如果发生错误，Servlet 实例将抛出 ServletException 异常或 UnavailableException 异常。当发生 UnavailableException 异常时，客户将接收到 HTTP 404(请求的资源不可用)响应或者 HTTP 503(服务器暂时忙，不能处理请求)响应。

4. 服务结束

当容器停止且卸载 Servlet 时，将执行 destroy()方法。通常情况下，不需要覆盖 destroy()方法，但是当需要完成如关闭数据库连接等资源回收工作时，可以覆盖它。在 destroy()方法调用之后，该 Servlet 实例将会被容器释放并被 Java 的垃圾收集器回收。当再次需要请求该 Servlet 时，容器会创建一个新的实例。

3.2.3 Servlet 编程接口

Servlet 规范中规定，所有的 Servlet 类必须实现 javax. servlet. Servlet 接口。Servlet 规范提供了该接口的两个通用实现类：javax. servlet. GenericServlet 和 javax. servlet. http. HttpServlet。因此，程序员在编写 Servlet 时，既可以直接实现 Servlet 接口，也可以扩展 GenericServlet 或 HttpServlet。由于目前 Servlet 容器都是基于 HTTP 协议的，所以继承 HttpServlet 是最方便、最快捷的开发方式。

HttpServlet 类中包含了 init()、destroy()和 service()方法，同时增加了一些附加的方法，这些方法可以供 service()自动调用。其中最重要的是 doGet()方法和 doPost()方法。

1. doGet()方法

doGet()方法用于处理 HTTP 的 GET 请求。当客户通过 HTML 表单发出一个 HTTP GET 请求或者直接请求一个 URL 时，doGet()方法被自动调用。与 GET 请求相

关的参数添加到 URL 的后面，并与这个请求一起发送。当不会修改服务器端的数据时，应该使用 doGet()方法。

doGet()的语法如下：

```
protected void doGet(
    HttpServletRequest request,  //指定 HttpServletRequest 对象,包含客户端请求
    HttpServletResponse response    //指定 HttpServletResponse 对象,包含 Servlet 响应
)throws ServletException, IOException
```

2. doPost()方法

doPost()方法用于处理 HTTP 的 POST 请求。当客户通过 HTML 表单发出一个 HTTP POST 请求时，doPost()方法被自动调用。与 POST 请求相关的参数作为一个单独的 HTTP 请求从客户端发送到服务器。当需要修改服务器端的数据时，应该使用 doPost()方法。

doPost()的语法如下：

```
protected void doPost(
    HttpServletRequest request,  //指定 HttpServletRequest 对象,包含客户端请求
    HttpServletResponse response    //指定 HttpServletResponse 对象,包含 Servlet 响应
)throws ServletException, IOException
```

3.2.4　编写和部署 Servlet

下面通过一个简单的小例子说明如何编写和部署 Servlet。

1. 编写 Servlet

代码 3-4 给出的 servlet.LoginServlet 用于登录时的身份验证。

代码 3-4　LoginServlet.jsp

```
package servlet;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.*;
public class LoginServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {
        //由于本程序中 doGet()的功能和 doPost()的功能一致,将直接调用 doPost()
        doPost(request,response);
    }
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                       throws ServletException, IOException {
        //设置 request 字符编码
        request.setCharacterEncoding("utf-8");
        String path=request.getContextPath();
        String basePath=request.getScheme()+"://";
```

```
            basePath+=request.getServerName();
            basePath+=":";
            basePath+=request.getServerPort();
            basePath+=path+"/";

            //获取客户请求参数
            String username=request.getParameter("username");
            String password=request.getParameter("password");
            if (username.equals("张三") && password.equals("123")){
                //将用户名保存在 session 中
                HttpSession session=request.getSession();
                session.setAttribute("username",username);
                //重定向到 index.jsp 页面,注意: 由于 index.jsp 位于网站根目录中,为了防止
                //出现路径错误,最好使用绝对路径
                response.sendRedirect(basePath+"index.jsp");
            }
            else
                response.sendRedirect(basePath+"login.jsp");
        }
    }
```

为验证上述程序，将前面给出的 login. jsp 文件的 action 值修改为/servlet/LoginServlet，然后发布到 Web 服务器上进行查看。

基于 Web 的服务程序在与客户的交互过程中完成了特定的系统功能。为了实现交互过程，Servlet 中必须能够获取客户的请求参数，能够完成服务响应，能够读取和修改 session，能够实现数据流输出。在 JSP 文件中，可以通过内置的 request、response 和 session 等对象来获取这些信息。尽管在 Servlet 中不能直接使用 JSP 提供的内置对象，但是在 service()、doGet()和 doPost()等方法中封装了类型为 HttpServletRequest 的对象 request 和类型为 HttpServletResponse 的对象 response。利用 request 对象可以获得与客户请求相关的信息，利用 response 对象可以完成服务响应处理。如果要完成与 session 有关的处理，必须获得封装在 request 或 response 中的 HttpSession 的实例。

2. 部署 Servlet

在编写完 Servlet 之后，必须在 web. xml 中对它进行正确的配置之后才能访问。代码 3-5 给出配置了 LoginServlet 的 web. xml。

代码 3-5　在 web. xml 中配置 Servlet

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet><!-部署 servlet.LoginServlet-->
        <description>Authenticate users</description>
```

```
        <display-name>loginServlet</display-name>
        <servlet-name>LoginServlet</servlet-name>
        <servlet-class>servlet.LoginServlet</servlet-class>
    </servlet>
    <!-为 Servlet 作 URL 映射 -->
    <servlet-mapping>
        <servlet-name>LoginServlet</servlet-name>
        <url-pattern>/servlet/LoginServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

其中，＜url-pattern＞/servlet/LoginServlet＜/url-pattern＞标签给出了访问该 Servlet 所要用到的 URL，也就是为什么要将 login.jsp 文件的 action 值修改为/servlet/LoginServlet 的原因。

MyEclipse 等 IDE 集成开发环境能够自动生成 Servlet 的结构框架和部署过程，程序员在开发时可以直接采用，提高编程效率。

3.2.5　Servlet 过滤器

1. 什么是过滤器

Servlet 过滤器(Filter)作为一种特殊的 Servlet，位于客户端和 Web 应用程序之间。利用 Servlet 过滤器可以在客户请求到达 Servlet 和 JSP 文件之前，或在服务响应到达客户端前完成一些附加的操作。多个过滤器形成一个过滤器链，过滤器链中不同过滤器的先后顺序由部署文件 web.xml 中过滤器映射＜filter-mapping＞的顺序决定。图 3-2 演示了过滤器链的工作机制。

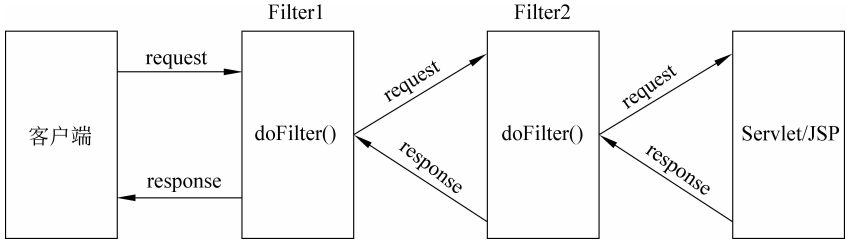

图 3-2　Servlet 过滤器链

利用 Servlet 过滤器可以完成以下工作。

(1) 字符编码转换：利用 Servlet 过滤器解决中文乱码问题。

(2) 权限验证：将 Web 服务程序的身份验证功能放到一个过滤器中，可以避免在每一个 Servlet 类和 JSP 文件中都要检查用户是否登录，身份是否合法等。

(3) 日志记录。

2. 过滤器的编程接口

所有的 Servlet 过滤器类都必须实现 javax.servlet.Filter 接口。这个接口含有过滤器类必须实现的 3 个方法。