

户访问网站时可能发生的各种情况,比如用户登录网站后在 session 的有效期外进行相应操作,用户会看到一张错误页面。这样的现象是不允许发生的。为了避免这种情况的发生,在开发系统时应该对 session 的有效性进行判断。

在 session 对象中提供了设置会话生命周期的方法,分别介绍如下。

- ☑ `getLastAccessedTime()`: 返回客户端最后一次与会话相关联的请求时间。
- ☑ `getMaxInactiveInterval()`: 以秒为单位返回一个会话内两个请求最大时间间隔。
- ☑ `setMaxInactiveInterval()`: 以秒为单位设置 session 的有效时间。

例如,通过 `setMaxInactiveInterval()` 方法设置 session 的有效期为 10000 秒,超出这个范围 session 将失效。

```
session.setMaxInactiveInterval(10000);
```

3.4.5 session 对象的应用

session 是较常用的内置对象之一,与 request 对象相比其作用范围更大。下面通过实例介绍 session 对象的应用。

例 3.08 在 index.jsp 页面中,提供用户输入用户名文本框;在 session.jsp 页面中,将用户输入的用户名保存在 session 对象中,用户在该页面中可以添加最喜欢去的地方;在 result.jsp 页面中,将用户输入的用户名与最想去的地方在页面中显示。(实例位置:光盘\TM\Instances\3.08)

(1) index.jsp 页面的代码如下:

```
<form id="form1" name="form1" method="post" action="session.jsp">
  <div align="center">
    <table width="23%" border="0">
      <tr>
        <td width="36%"><div align="center">您的名字是: </div></td>
        <td width="64%">
          <label>
            <div align="center">
              <input type="text" name="name" />
            </div>
          </label>
        </td>
      </tr>
      <tr>
        <td colspan="2">
          <label>
            <div align="center">
              <input type="submit" name="Submit" value="提交" />
            </div>
          </label>
        </td>
      </tr>
    </table>
```



```
</div>
</form>
```

该页面运行结果如图 3.11 所示。

图 3.11 index.jsp 页面运行结果

(2) 在 session.jsp 页面中, 将用户在 index.jsp 页面中输入的用户名保存在 session 对象中, 并为用户提供用于添加最想去的地址的文本框。代码如下:

```
<%
    String name = request.getParameter("name");           //获取用户填写的用户名
    session.setAttribute("name",name);                   //将用户名保存在 session 对象中
%>
<div align="center">
<form id="form1" name="form1" method="post" action="result.jsp">
<table width="28%" border="0">
<tr>
<td>您的名字是: </td>
<td><%=name%></td>
</tr>
<tr>
<td>您最喜欢去的地方是: </td>
<td><label>
<input type="text" name="address" />
</label></td>
</tr>
<tr>
<td colspan="2"><label>
<div align="center">
<input type="submit" name="Submit" value="提交" />
</div>
</label></td>
</tr>
</table>
</form>
```

session.jsp 页面运行结果如图 3.12 所示。

图 3.12 session.jsp 页面运行结果

(3) 在 result.jsp 页面中, 实现将用户输入的用户名、最喜欢去的地方在页面中显示。代码如下:

```
<%
    String name = (String)session.getAttribute("name"); //获取保存在 session 范围内的对象
    String solution = request.getParameter("address"); //获取用户输入的最喜欢去的地方
```

```

%>
<form id="form1" name="form1" method="post" action="">
  <table width="28%" border="0">
    <tr>
      <td colspan="2"><div align="center"><strong>显示答案</strong></div></td>
    </tr>
    <tr>
      <td width="49%"><div align="left">您的名字是: </div></td>
      <td width="51%"><label>
        <div align="left"><%=name%></div>                                <!-- 将用户输入的用户名在页面中显示 -->
      </label></td>
    </tr>
    <tr>
      <td><label>
        <div align="left">您最喜欢去的地方是: </div>
      </label></td>
      <td><div align="left"><%=solution%></div></td>                                <!-- 将用户输入的最喜欢去的地方在页面中显示 -->
    </tr>
  </table>
</form>

```

result.jsp 页面的运行结果如图 3.13 所示。

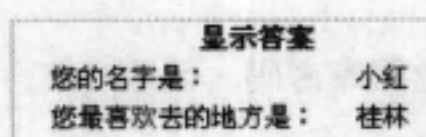


图 3.13 result.jsp 页面的运行结果

3.5 application 对象

视频讲解：光盘\TM\Video\3\application 对象.exe

application 对象可将信息保存在服务器中，直到服务器关闭，否则 application 对象中保存的信息会在整个应用中都有效。与 session 对象相比，application 对象的生命周期更长，类似于系统的“全局变量”。application 对象的常用方法如表 3.4 所示。

表 3.4 application 对象的常用方法

方 法	返 回 值	说 明
getAttribute(String name)	Object	通过关键字返回保存在 application 对象中的信息
getAttributeNames()	Enumeration	获取所有 application 对象使用的属性名
setAttribute(String key,Object obj)	void	通过指定的名称将一个对象保存在 application 对象中
getMajorVersion()	int	获取服务器支持的 Servlet 版本号
getServerInfo()	String	返回 JSP 引擎的相关信息
removeAttribute(String name)	void	删除 application 对象中指定名称的属性
getRealPath()	String	返回虚拟路径的真实路径
getInitParameter(String name)	String	获取指定 name 的 application 对象属性的初始值

3.5.1 访问应用程序初始化参数

application 提供了对应用程序环境属性访问的方法。例如,通过初始化信息为程序提供连接数据库的 URL、用户名、密码,每个 Servlet 程序客户和 JSP 页面都可以使用它获取连接数据库的信息。为了实现该目的,Tomcat 使用了 web.xml 文件。

application 对象访问应用程序初始化参数的方法分别介绍如下:

- ☑ getInitParameter(String name): 返回一个已命名的参数值。
- ☑ getAttributeNames(): 返回所有已定义的应用程序初始化名称的枚举。

例 3.09 访问应用程序初始化参数。(实例位置: 光盘\TM\Instances\3.09)

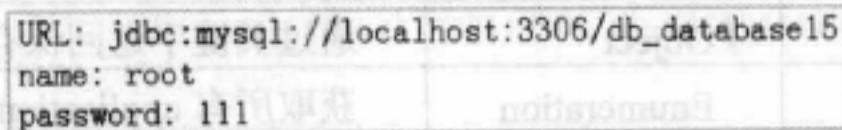
(1) 在 web.xml 文件中通过配置<context-param>元素初始化参数。程序代码如下:

```
<context-param>                <!-- 定义连接数据库 URL -->
    <param-name>url</param-name>
    <param-value>jdbc:mysql://localhost:3306/db_database15</param-value>
</context-param>
<context-param>                <!-- 定义连接数据库用户名 -->
    <param-name>name</param-name>
    <param-value>root</param-value>
</context-param>
<context-param>                <!-- 定义连接数据库密码 -->
    <param-name>password</param-name>
    <param-value>111</param-value>
</context-param>
```

(2) 在 index.jsp 页面中,访问 web.xml 文件获取初始化参数。代码如下:

```
<%
    String url = application.getInitParameter("url");        //获取初始化参数,与 web.xml 文件中的内容相对应
    String name = application.getInitParameter("name");
    String password = application.getInitParameter("password");
    out.println("URL: "+url+"<br>");                          //将信息在页面中显示
    out.println("name: "+name+"<br>");
    out.println("password: "+password+"<br>");
%>
```

index.jsp 页面运行结果如图 3.14 所示。



```
URL: jdbc:mysql://localhost:3306/db_database15
name: root
password: 111
```

图 3.14 index.jsp 页面运行结果

3.5.2 管理应用程序环境属性

与 session 对象相同,也可以在 application 对象中设置属性。与 session 对象不同的是,session 只

是在当前客户的会话范围内有效，当超过保存时间，session 对象就被收回；而 application 对象在整个应用区域中都有效。application 对象管理应用程序环境属性的方法分别介绍如下。

- ☑ getAttributeNames(): 获得所有 application 对象使用的属性名。
- ☑ getAttribute(String name): 从 application 对象中获取指定对象名。
- ☑ setAttribute(String key, Object obj): 使用指定名称和指定对象在 application 对象中进行关联。
- ☑ removeAttribute(String name): 从 application 对象中去掉指定名称的属性。

3.6 out 对象

 视频讲解：光盘\TM\Video\3\out 对象.exe

out 对象用于在 Web 浏览器内输出信息，并且管理应用服务器上的输出缓冲区。在使用 out 对象输出数据时，可以对数据缓冲区进行操作，及时清除缓冲区中的残余数据，为其他的输出让出缓冲空间。待数据输出完毕后，要及时关闭输出流。

3.6.1 管理响应缓冲

在使用 out 输出数据时，需要使用 clear() 方法清除缓冲区内容，以便重新开始操作。此外，也可以通过 clearBuffer() 方法清除缓冲区的内容。两者的区别是：通过 clear() 方法清除缓冲区，如果相应内容已经提交，则会报出 IOException 异常；而使用 clearBuffer() 方法，即使内容已经提交给客户端，也能够访问该方法。out 对象管理缓冲区的方法如表 3.5 所示。

表 3.5 out 对象管理缓冲区的常用方法

方 法	返 回 值	说 明
clear()	void	清除缓冲区中尚存的内容
clearBuffer()	void	清除当前缓冲区中尚存的内容
flush()	void	刷新流
isAutoFlush()	boolean	检查当前缓冲区是自动清空，还是满了就抛出异常
getBufferSize()	int	获取缓冲区的大小

3.6.2 向客户端输出数据

out 对象的另一个重要功能是向客户端写入数据。这一功能在 JSP 开发过程中使用最为频繁，然而使用起来也最为简单。out 对象提供了两个向页面中输出信息的方法，分别介绍如下。

- ☑ print(): 在页面中打印字符串信息，不换行。
- ☑ println(): 在页面中打印字符串信息，并且换行。

例如，通过 out 对象实现向客户端输出数据。

```
<%out.print("编程词典 程序员的黄金搭档");%>
```


3.7 其他内置对象

除以上常用的内置对象外，JSP 中还包括 `pageContext`、`config`、`page` 及 `exception` 等对象。下面对这些对象分别进行介绍。

3.7.1 获取会话范围的 `pageContext` 对象

`pageContext` 对象的作用是取得任何范围的参数，通过它可以获取 JSP 页面的 `out`、`request`、`response`、`session`、`application` 等对象。`pageContext` 对象的创建和初始化都是由容器来完成的，在 JSP 页面中可以直接使用 `pageContext` 对象。`pageContext` 对象的常用方法如表 3.6 所示。

表 3.6 `pageContext` 对象的常用方法

方 法	返 回 值	说 明
<code>forward(String path)</code>	<code>void</code>	将 JSP 页面重新定向至另一个页面
<code>getAttribute(String name)</code>	<code>Object</code>	获取参数值
<code>getAttributeNamesInScope(int scope)</code>	<code>Enumeration</code>	获取某范围的参数名称集合
<code>getRequest()</code>	<code>ServletRequest</code>	获取 <code>request</code> 对象
<code>getResponse()</code>	<code>ServletResponse</code>	获取 <code>response</code> 对象
<code>getOut()</code>	<code>JspWriter</code>	获取 <code>out</code> 对象
<code>getSession()</code>	<code>HttpSession</code>	获取 <code>session</code> 对象
<code>getPage()</code>	<code>Object</code>	获取 <code>page</code> 对象
<code>setAttribute(String name, Object value)</code>	<code>void</code>	设置指定参数属性

3.7.2 读取 `web.xml` 配置信息的 `config` 对象

`config` 对象的主要作用是取得服务器的配置信息。通过 `pageContext` 对象的 `getServletConfig()` 方法可以获取一个 `config` 对象。当一个 `Servlet` 初始化时，容器把某些信息通过 `config` 对象传递给这个 `Servlet`。开发者可以在 `web.xml` 文件中为应用程序环境中的 `Servlet` 程序和 JSP 页面提供初始化参数。`config` 对象的常用方法如表 3.7 所示。

表 3.7 `config` 对象的常用方法

方 法	返 回 值	说 明
<code>getInitParameter(String name)</code>	<code>String</code>	获取服务器指定参数的初始值
<code>getInitParameterNames()</code>	<code>Enumeration</code>	获取服务器所有初始参数名称
<code>getServletContext()</code>	<code>ServletContext</code>	获取 <code>Servlet</code> 上下文
<code>getServletName()</code>	<code>String</code>	获取 <code>Servlet</code> 服务器名

3.7.3 应答或请求的 page 对象

page 对象代表 JSP 本身，只有在 JSP 页面内才是合法的。page 隐含对象本质上包含当前 Servlet 接口引用的变量，类似于 Java 编程中的 this 指针。page 对象的常用方法如表 3.8 所示。

表 3.8 page 对象的常用方法

方 法	返 回 值	说 明
getClass()	Object	返回当前 Object 的类
hashCode()	Object	返回此 Object 的哈希代码
toString()	String	将此 Object 类转换成字符串对象
equals(Object obj)	boolean	比较此对象与指定的对象是否相等

3.7.4 获取异常信息的 exception 对象

exception 对象的作用是显示异常信息，只有在包含 isErrorPage="true" 的页面中才可以被使用，在一般的 JSP 页面中使用该对象将无法编译 JSP 文件。exception 对象和 Java 的所有对象一样，都具有系统的继承结构。exception 对象几乎定义了所有异常情况。在 Java 程序中，可以使用 try/catch 关键字来处理异常情况；如果在 JSP 页面中出现没有捕捉到的异常，就会生成 exception 对象，并把 exception 对象传送到在 page 指令中设定的错误页面中，然后在错误页面中处理相应的 exception 对象。exception 对象的常用方法如表 3.9 所示。

表 3.9 exception 对象的常用方法

方 法	说 明
getMessage()	返回 exception 对象的异常信息字符串
getLocalizedMessage()	返回本地化的异常错误
toString()	返回关于异常错误的简单信息描述
fillInStackTrace()	重写异常错误的栈执行轨迹

例 3.10 获取异常信息的 exception 对象。（实例位置：光盘\TM\Instances\3.10）

(1) 创建 index.jsp 页面，在该页面中编写代码，并通过 errorPage 属性指定有异常信息，系统将转发至 error.jsp 页面。

```
<%@ page language="java" import="java.util.*" pageEncoding="gbk" errorPage="error.jsp"%>
<%
    int apple = Integer.parseInt("ad");
    out.println("苹果每斤"+apple+"元");
%>
```

(2) 编写 error.jsp 页面，用于接收传递过来的异常信息。关键代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="gbk" isErrorPage="true"%>
<body>
```



```
错误提示为: <%=exception.getMessage() %>
</body>
```

由于将字符串“ad”转换为 int 型变量会发生异常,因此系统将转发至 error.jsp 页面,如图 3.15 所示。如果将 index.jsp 页面中的代码 `int apple = Integer.parseInt("ad");` 替换为 `int apple = Integer.parseInt("3");`,则不会有异常发生,不会转发至 error.jsp 页面,运行结果如图 3.16 所示。


错误提示为: For input string: "ad"

图 3.15 本实例的运行结果

苹果每斤3元

图 3.16 没有异常发生

3.8 实 战

 视频讲解: 光盘\TM\Video\3\实战.exe

3.8.1 application 实现网页计数器

由于 application 保存的信息在整个应用中都有效,因此可以将当前访问网站的数量保存在 application 对象中,在每次访问网页时,实现将保存在 application 对象中的值加 1,从而实现网页计数器。

例 3.11 应用 application 对象实现网页计数器。(实例位置: 光盘\TM\Instances\3.11)

(1) 在 index.jsp 页面中,实现将信息保存在 application 对象中。具体代码如下:

```
<h4>application 对象实现网页计数器</h4>
<%
    out.println("设置数值");      //页面显示信息
    Integer intcount;              //定义用于网页计数变量
    if(application.getAttribute("count")==null){      //如果保存在 application 对象中的内容为空
        intcount = 1;
    }
    else{
        intcount = (Integer.parseInt(
            application.getAttribute("count").toString())); //获取保存在 application 对象中的内容
    }
    application.setAttribute("name","cdd");          //将信息保存在 application 对象内
    application.setAttribute("count",intcount);
    out.print("set name = cdd ");
    out.print("<br>set counter = "+intcount+"<br>");

    %>
    <a href="gateppatter.jsp">计数器页面</a>          <!-- 转发至 gateppatter.jsp 页面 -->
```

index.jsp 页面运行结果如图 3.17 所示。

(2) 在 gateppatter.jsp 页面中,实现计数器统计。代码如下:

```
<br>获取用户名: <%=application.getAttribute("name")%>
<br>计数器:
```



```

<%
    int mycount = Integer.valueOf(
        application.getAttribute("count").toString()).intValue(); //获取保存在 application 对象中的信息
    out.println(mycount);
    application.setAttribute("count",Integer.toString(mycount+1)); //将该信息做加 1 处理
%>

```

gateppatter.jsp 页面运行结果如图 3.18 所示。

application 对象实现网页计数器
设置数值 set name = cdd
set counter = 1
计数器页面

图 3.17 index.jsp 页面运行结果

获取用户名: cdd
计数器: 3

图 3.18 gateppatter.jsp 页面运行结果

3.8.2 在提交表单时加入验证码

几乎所有的登录系统都会有验证码, 验证码的实现有多种, 本节将向读者介绍的验证码是由 4 位数字的图片组成。在本实例的 num 文件夹中保存有 0~9 十张图片, 每次刷新页面时, 从中随机取出 4 张图片作为验证码, 并在 check.jsp 页面中验证用户名与密码是否正确。

例 3.12 在提交表单时加入验证码。(实例位置: 光盘\TM\Instances\3.12)

(1) 在 index.jsp 页面中, 为用户设置“用户名”、“密码”、“验证码”文本框。代码如下:

```

<form name="form1" method="POST" action="check.jsp"><!-- 定义 form 表单 -->
    <table width="364" height="145" border="0" align="center"
        cellpadding="0" cellspacing="0">
        <tr>
            <td height="2" colspan="2"></td>
        </tr>
        <tr>
            <td height="2" colspan="2" valign="top"></td>
        </tr>
        <tr>
            <td width="54" height="22" valign="bottom">
                <span class="STYLE15">用户名: </span>
            </td>
            <td width="310" valign="bottom"><input name="UserName" type="text" class="input2"
                onKeyDown="if(event.keyCode==13) {form1.PWD.focus();}"
                onMouseOver="this.style.background='#F0DAF3';"
                onMouseOut="this.style.background='#FFFFFF'"><!-- 设置用户名文本框, 并设置了鼠标经过时样式 -->
            </td>
        </tr>
        <tr>
            <td height="23" colspan="2" valign="bottom"></td>
        </tr>
    </table>

```


(2) 在 index.jsp 页面中编写 JavaScript 脚本, 通过定义 mycheck()方法来验证用户输入的用户名、密码以及验证码是否正确。代码如下:

```
<script language="javascript">
function mycheck(){
if (form1.UserName.value=="")
{alert("请输入用户名！");form1.UserName.focus();return;}
if(form1.PWD.value=="")
```



```

{alert("请输入密码! ");form1.PWD.focus();return;}
if(form1.yanzheng.value=="")
{alert("请输入验证码!");form1.yanzheng.focus();return;}
if(form1.yanzheng.value != form1.verifycode2.value)
{alert("请输入正确的验证码!!");form1.yanzheng.focus();return;}
form1.submit();
}
</script>

```

(3) 验证用户输入的用户名、密码、验证码是否合法。代码如下:

```

<body>
<%
String name = request.getParameter("UserName");    //获取用户名参数
String password = request.getParameter("PWD");      //获取用户输入的密码参数
String message ;
if(name.equals("mr")&&(password.equals("mrsoft"))){ //判断用户名与密码是否合法
    message ="可以登录系统";
}
else{
    message ="用户名或密码错误";
}
%>
<script language="javascript">
alert("<%=message%>!!")
window.location.href='index.jsp';
</script>
</body>

```

本实例的运行结果如图 3.19 所示。

图 3.19 实例的运行结果

3.9 本章小结


本章向读者介绍了 JSP 的 9 个内置对象的基本应用和常用方法, 其中详细地介绍了 request 对象、response 对象、out 对象, 对 pageContext 对象、config 对象、page 对象、exception 对象中的方法作了简要的介绍。JSP 内置对象在开发中应用较为广泛, 希望通过本章的学习, 读者可在开发中灵活地使用这些内置对象。

3.10 实战练习

1. 应用 JSP 内置对象，实现用户注册。（答案位置：光盘\TM\Instances\3.13）
2. 应用 JSP 内置对象，实现简单的留言本，写入留言，提交后显示留言内容。（答案位置：光盘\TM\Instances\3.14）
3. 应用 cookie 对象，实现自动登录。（答案位置：光盘\TM\Instances\3.15）

第4章

Servlet 技术

( 视频讲解：72 分钟)

Servlet 是 Java 语言应用到 Web 服务器端的扩展技术，它的产生为 Java Web 开发奠定了基础。随着 Web 开发技术的不断发展，Servlet 也在不断发展与完善，并凭借其安全性、跨平台等诸多优点，深受广大 Java 编程人员的青睐。在本章中，将以理论与实践相结合的方式系统讲解 Servlet 技术。

通过阅读本章，您可以：

- » 理解 Servlet 技术原理
- » 了解 Servlet 在 Servlet 容器中的生命周期
- » 掌握 Servlet 的创建与配置方法
- » 掌握 Servlet API 的主要接口与类
- » 理解 Servlet 过滤器的实现原理
- » 掌握 Filter API 的常用接口
- » 掌握 Servlet 过滤器的创建与配置
- » 掌握 Servlet 过滤器的典型应用

4.1 Servlet 基础

Servlet 是使用 Java Servlet 接口 (API) 运行在 Web 应用服务器上的 Java 程序。与普通 Java 程序不同,它是位于 Web 服务器内部的服务器端的 Java 应用程序,可以对 Web 浏览器或其他 HTTP 客户端程序发送的请求进行处理。

4.1.1 Servlet 与 Servlet 容器

Servlet 对象与普通的 Java 对象不同,它可以处理 Web 浏览器或其他 HTTP 客户端程序发送的 HTTP 请求,但前提条件是把 Servlet 对象布置到 Servlet 容器之中,也就是说,其运行需要 Servlet 容器的支持。

通常情况下,Servlet 容器也就是指 Web 容器,如 Tomcat、Jboss、Resin、WebLogic 等,它们对 Servlet 进行控制。当一个客户端发送 HTTP 请求时,由容器加载 Servlet 对其进行处理并作出响应。在 Web 容器中,Servlet 主要经历 4 个阶段,如图 4.1 所示。

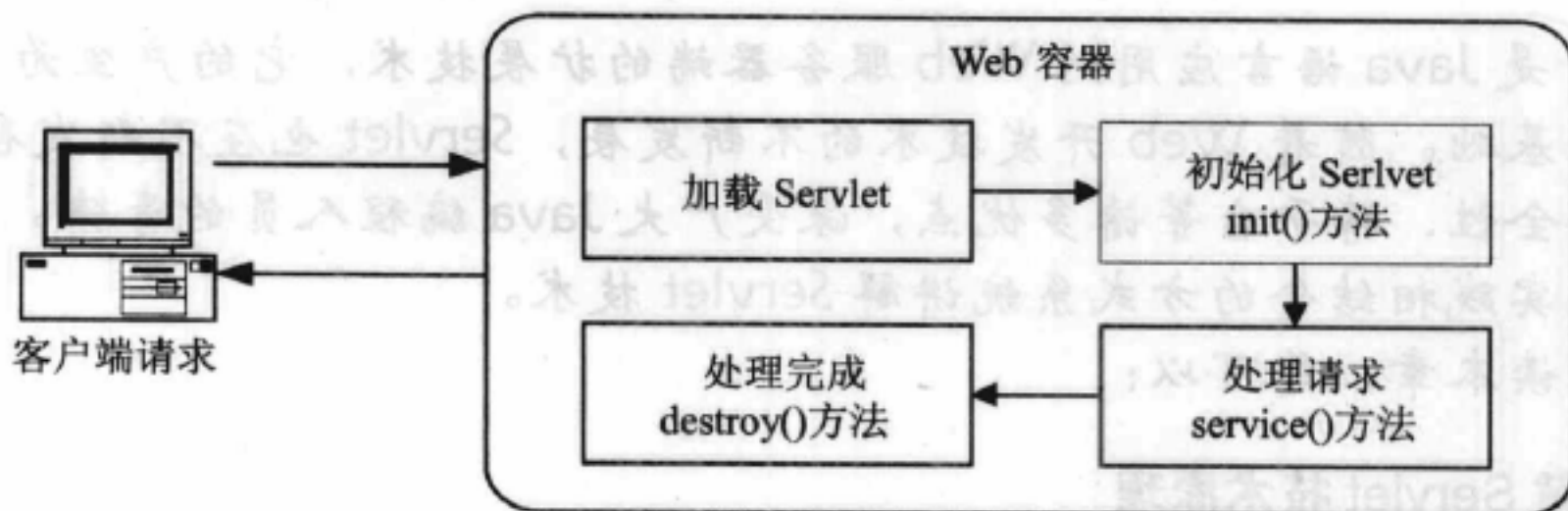


图 4.1 Servlet 与容器

Servlet 与 Web 容器的关系是非常密切的,在 Web 容器中 Servlet 主要经历了 4 个阶段,这 4 个阶段实质是 Servlet 的生命周期,由容器进行管理。

(1) 在 Web 容器启动或客户机第一次请求服务时,容器将加载 Servlet 类并将其放入到 Servlet 实例池。

(2) 当 Servlet 实例化后,容器将调用 Servlet 对象的 init()方法完成 Servlet 的初始化操作,主要是为了让 Servlet 在处理请求之前做一些初始化工作。

(3) 容器通过 Servlet 的 service()方法处理客户端请求。在 Service()方法中,Servlet 实例根据不同的 HTTP 请求类型作出不同处理,并在处理之后作出相应的响应。

(4) 在 Web 容器关闭时,容器调用 Servlet 对象的 desdroy()方法对资源进行释放。在调用此方法后,Servlet 对象将被垃圾回收器回收。

4.1.2 Servlet 技术特点

Servlet 采用 Java 语言编写,继承了 Java 语言中的诸多优点,同时还对 Java 的 Web 应用进行了扩

展。Servlet 具有以下特点:

☒ 方便、实用的 API 方法

Servlet 对象对 Web 应用进行了封装, 针对 HTTP 请求提供了丰富的 API 方法, 它可以处理表单提交数据、会话跟踪、读取和设置 HTTP 头信息等, 对 HTTP 请求数据的处理非常方便, 只需要调用相应的 API 方法即可。

☒ 高效的处理方式

Servlet 的一个实例对象可以处理多个线程的请求。当多个客户端请求一个 Servlet 对象时, Servlet 为每一个请求分配一个线程, 而提供服务的 Servlet 对象只有一个, 因此我们说 Servlet 的多线程处理方式是**非常高效的**。

☒ 跨平台

Servlet 采用 Java 语言编写, 因此它继承了 Java 的跨平台性, 对于已编写好的 Servlet 对象, 可运行在多种平台之中。

☒ 更加灵活、扩展

Servlet 与 Java 平台的关系密切, 它可以访问 Java 平台丰富的类库; 同时由于它采用 Java 语言编写, 支持封装、继承等面向对象的优点, 使其更具应用的灵活性; 此外, 在编写过程中, 它还对 API 接口进行了适当扩展。

☒ 安全性

Servlet 采用了 Java 的安全框架, 同时 Servlet 容器还为 Servlet 提供了额外的功能, 其安全性是非常高的。

4.1.3 Servlet 技术功能

Servlet 是位于 Web 服务器内部的服务器端的 Java 应用程序, 它对 Java Web 的应用进行了扩展, 可以对 HTTP 请求进行处理及响应, 功能十分强大。

- ☒ Servlet 与普通 Java 应用程序不同, 它可以处理 HTTP 请求以获取 HTTP 头信息, 通过 `HttpServletRequest` 接口与 `HttpServletResponse` 接口对请求进行处理及回应。
- ☒ Servlet 可以在处理业务逻辑之后, 将动态的内容通过返回并输出到 HTML 页面中, 与用户请求进行交互。
- ☒ Servlet 提供了强大的过滤器功能, 可针对请求类型进行过滤设置, 为 Web 开发提供灵活性与扩展性。
- ☒ Servlet 可与其他服务器资源进行通信。

4.1.4 Servlet 与 JSP 的区别

Servlet 是一种运行在服务器端的 Java 应用程序, 先于 JSP 的产生。在 Servlet 的早期版本中, 业务逻辑代码与网页代码写在一起, 给 Web 程序的开发带来了很多不便。如网页设计的美工人员, 需要学习 Servlet 技术进行页面设计; 而在程序设计中, 其代码又过于复杂, Servlet 所产生的动态网页需要在代码中编写大量输出 HTML 标签的语句。针对早期版本 Servlet 的不足, Sun 提出了 JSP (Java Server

Page) 技术。

JSP 是一种在 Servlet 规范之上的动态网页技术,通过 JSP 页面中嵌入的 Java 代码,可以产生动态网页。也可以将其理解为是 Servlet 技术的扩展,在 JSP 文件被第一次请求时,它会被编译成 Servlet 文件,再通过容器调用 Servlet 进行处理。由此可以看出,JSP 与 Servlet 技术的关系是十分紧密的。

JSP 虽是在 Servlet 的基础上产生的,但与 Servlet 也存在一定的区别。

- ☑ Servlet 承担客户请求与业务处理的中间角色,需要调用固定的方法,将动态内容混合到静态之中产生 HTML;而在 JSP 页面中,可直接使用 HTML 标签进行输出,要比 Servlet 更具显示层的意义。
- ☑ Servlet 中需要调用 Servlet API 接口处理 HTTP 请求,而在 JSP 页面中,则直接提供了内置对象进行处理。
- ☑ Servlet 的使用需要进行一定的配置,而 JSP 文件通过“.jsp”扩展名部署在容器之中,容器对其自动识别,直接编译成 Servlet 进行处理。

4.1.5 Servlet 代码结构

在 Java 中,通常所说的 Servlet 是指 HttpServlet 对象,在声明一个对象为 Servlet 时,需要继承 HttpServlet 类。HttpServlet 类是 Servlet 接口的一个实现类,继承此类后,可以重写 HttpServlet 类中的方法对 HTTP 请求进行处理。其代码结构如下:

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestServlet extends HttpServlet {
    //初始化方法
    public void init() throws ServletException {
    }
    //处理 HTTP Get 请求
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    }
    //处理 HTTP Post 请求
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    }
    //处理 HTTP Put 请求
    public void doPut(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    }
    //处理 HTTP Delete 请求
    public void doDelete(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    }
}
```



```

    }
    //销毁方法
    public void destroy() {
        super.destroy();
    }
}

```

上述代码显示了一个 Servlet 对象的代码结构，TestServlet 类通过继承 HttpServlet 类被声明为一个 Servlet 对象。此类中包含 6 个方法，其中 init() 方法与 destroy() 方法为 Servlet 初始化与生命周期结束所调用的方法，其余的 4 个方法为 Servlet 针对处理不同的 HTTP 请求类型所提供的方法，其作用如注释中所示。

在一个 Servlet 对象中，最常用的方法是 doGet() 与 doPost() 方法，这两个方法分别用于处理 HTTP 的 Get 与 Post 请求。例如，<form> 表单对象所声明的 method 属性为 “post”，提交到 Servlet 对象处理时，Servlet 将调用 doPost() 方法进行处理。

4.1.6 简单的 Servlet 程序

在编写 Servlet 时，不必重写 Servlet 对象中的所有方法，只需重写请求所使用方法即可。例如，处理 get 请求需要重写 doGet() 方法，在此方法中编写业务逻辑代码。

例 4.01 简单的 Servlet 程序。（实例位置：光盘\TM\Instances\4.01）

```

public class SimpleServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("This is a Servlet.");
    }
}

```

注意：本实例演示了简单的 Servlet 程序，其涉及到的 Servlet 配置方法将在 4.2.2 节中进行详细讲解。

SimpleServlet 类是一个 Servlet 对象，它继承了 HttpServlet 类。在此类的 doGet() 方法中，通过 PrintWriter 对象向页面中打印了一句话，通过浏览器可查看此 Servlet 运行效果，如图 4.2 所示。

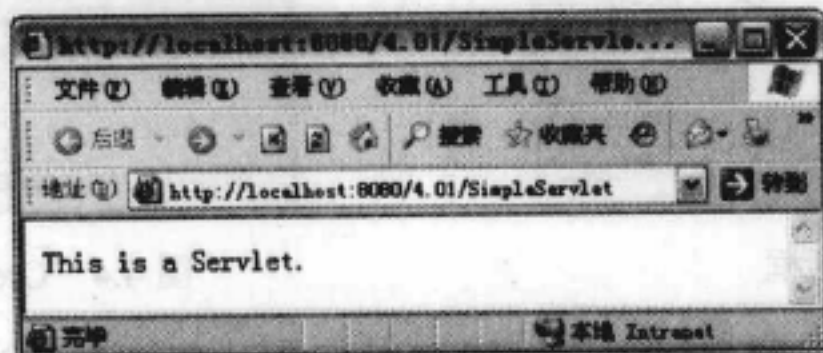



图 4.2 一个简单的 Servlet 程序

4.2 Servlet 开发

 视频讲解：光盘\TM\Video\4\Servlet 开发.exe

在 Java 的 Web 开发中，Servlet 具有重要的地位，程序中的业务逻辑可以由 Servlet 进行处理；它也可以通过 `HttpServletResponse` 对象对请求作出响应，功能十分强大。本节将对 Servlet 的创建及配置进行详细的讲解。

4.2.1 Servlet 的创建

Servlet 的创建十分简单，主要有两种创建方法。第一种方法为创建一个普通的 Java 类，使这个类继承 `HttpServlet` 类，再通过手动配置 `web.xml` 文件注册 Servlet 对象。此方法操作比较繁琐，在快速开发中通常不被采纳，而是使用第二种方法——直接通过 IDE 集成开发工具进行创建。

使用集成开发工具创建 Servlet 非常方便，下面以 MyEclipse 为例介绍 Servlet 的创建过程，其他开发工具大同小异。

(1) 在指定的项目中打开 MyEclipse 的新建向导，并在“向导”文本框中输入 `servlet`，MyEclipse 将自动导航到 Servlet，如图 4.3 所示。

(2) 选择 Servlet 单击“下一步”按钮，打开 Create a new Servlet 对话框，按提示创建 Servlet 对象，如图 4.4 所示。在其中要输入 Servlet 对象的包名及类名，同时 MyEclipse 还提供了 Servlet 对象的方法，如 `doPost()`、`doGet()` 等，可根据实际需要进行选择。



图 4.3 “新建”对话框

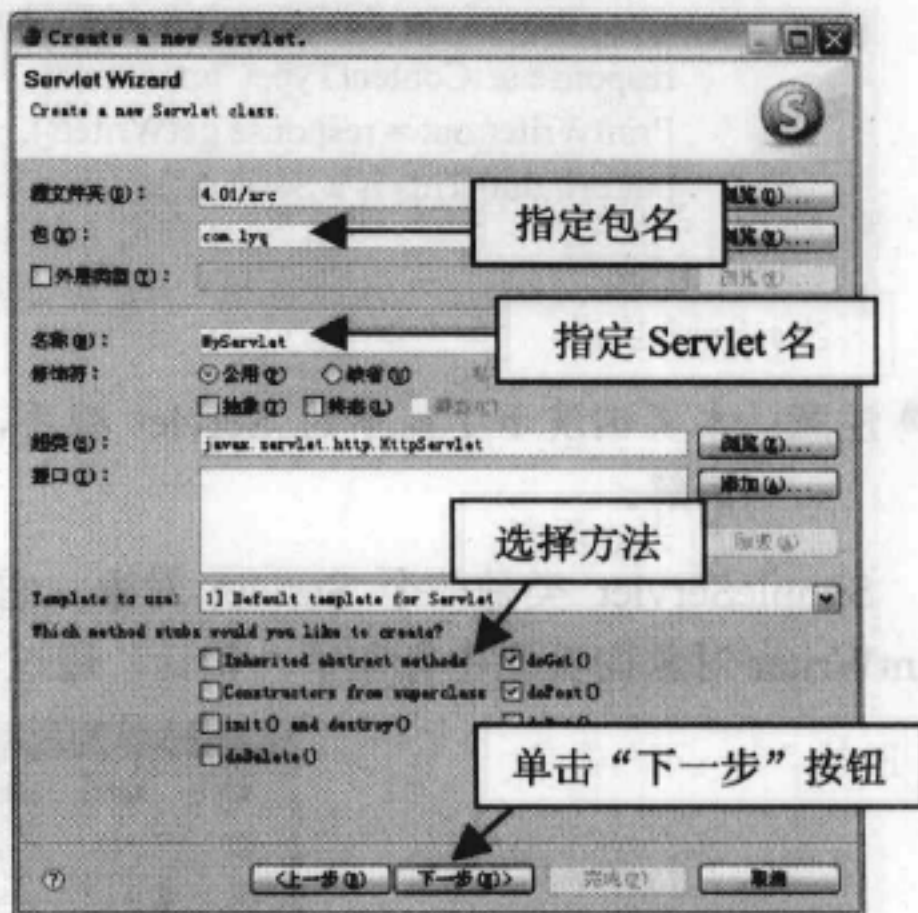


图 4.4 Create a new Servlet 对话框

(3) 单击“下一步”按钮，打开 Servlet 配置对话框，保持默认设置不变，单击“完成”按钮，完成 Servlet 的创建。

4.2.2 Servlet 配置的相关元素

要使 Servlet 对象正常地运行,需要进行适当的配置,以告知 Web 容器哪一个请求调用哪一个 Servlet 对象处理,对 Servlet 起到一个注册的作用。Servlet 的配置包含在 web.xml 文件中,主要通过以下两步进行设置。

(1) 声明 Servlet 对象

在 web.xml 文件中,通过<servlet>标签声明一个 Servlet 对象。在此标签下包含两个主要子元素,分别为<servlet-name>与<servlet-class>。其中,<servlet-name>元素用于指定 Servlet 的名称,此名称可以为自定义的名称;<servlet-class>元素用于指定 Servlet 对象的完整位置,包含 Servlet 对象的包名与类名。其声明语句如下:

```
<servlet>
  <servlet-name>SimpleServlet</servlet-name>
  <servlet-class>com.lyq.SimpleServlet</servlet-class>
</servlet>
```

(2) 映射 Servlet

在 web.xml 文件中声明了 Servlet 对象后,需要映射访问 Servlet 的 URL。此操作使用<servlet-mapping>标签进行配置。<servlet-mapping>标签包含两个子元素,分别为<servlet-name>与<url-pattern>。其中,<servlet-name>元素与<servlet>标签中的<servlet-name>元素相对应,不可以随意命名。<url-pattern>元素用于映射访问 URL。其配置方法如下:

```
<servlet-mapping>
  <servlet-name>SimpleServlet</servlet-name>
  <url-pattern>/SimpleServlet</url-pattern>
</servlet-mapping>
```

例 4.02 Servlet 的创建及配置。(实例位置: 光盘\TM\Instances\4.02)

(1) 创建名为 MyServlet 的 Servlet 对象,它继承了 HttpServlet 类。在此类中重写 doGet()方法,用于处理 HTTP 的 get 请求,通过 PrintWriter 对象进行简单输出。其关键代码如下:

```
public class MyServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        response.setCharacterEncoding("GBK");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("  <HEAD><TITLE>Servlet 实例</TITLE></HEAD>");
        out.println("  <BODY>");
        out.print("    Servlet 实例:  ");
        out.print(this.getClass());
        out.println("  </BODY>");
        out.println("</HTML>");
    }
}
```



```
        out.flush();
        out.close();
    }
}
```

(2) 在 web.xml 文件中对 MyServlet 进行配置, 其中访问 URL 的相对路径为“/servlet/MyServlet”。其关键代码如下:

```
<servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>com.lyq.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/servlet/MyServlet</url-pattern>
</servlet-mapping>
```

本实例使用 MyServlet 对象对请求进行处理, 其处理过程非常简单, 通过 PrintWriter 对象向页面中打印信息, 其运行结果如图 4.5 所示。

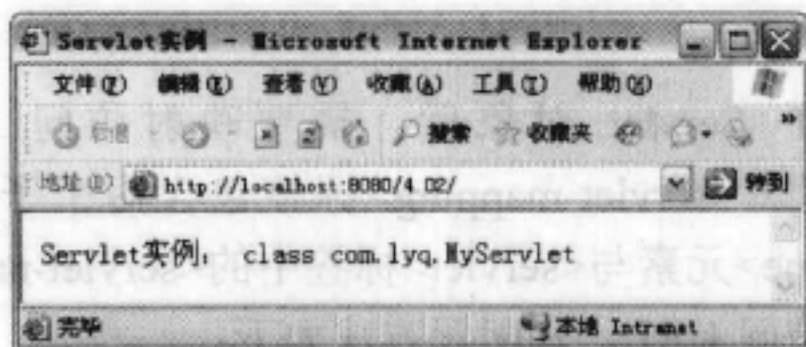



图 4.5 实例运行结果

4.3 Servlet API 编程常用的接口和类

 视频讲解: 光盘\TM\Video\4\Servlet API 编程常用的接口和类.exe

Servlet 是运行在服务器端的 Java 应用程序, 由 Servlet 容器对其进行管理, 当用户对容器发送 HTTP 请求时, 容器将通知相应的 Servlet 对象进行处理, 完成用户与程序之间的交互。在 Servlet 编程中, Servlet API 提供了标准的接口与类, 这些对象对 Servlet 的操作非常重要, 它们为 HTTP 请求与程序回应提供了丰富的方法。

4.3.1 Servlet 接口

Servlet 的运行需要 Servlet 容器的支持, Servlet 容器通过调用 Servlet 对象提供了标准的 API 接口, 对请求进行处理。在 Servlet 开发中, 任何一个 Servlet 对象都要直接或间接地实现 javax.servlet.Servlet 接口。在此接口中包含 5 个方法, 其功能及作用如表 4.1 所示。

表 4.1 Servlet 接口中的方法及说明

方 法	说 明
<code>public void init(ServletConfig config)</code>	Servlet 实例化后, Servlet 容器调用此方法来完成初始化工作
<code>public void service(ServletRequest request, ServletResponse response)</code>	此方法用于处理客户端的请求
<code>public void destroy()</code>	当 Servlet 对象应该从 Servlet 容器中移除时, 容器调用此方法, 以便释放资源
<code>public ServletConfig getServletConfig()</code>	此方法用于获取 Servlet 对象的配置信息, 返回 ServletConfig 对象
<code>public String getServletInfo()</code>	此方法返回有关 Servlet 的信息, 它是纯文本格式的字符串, 如作者、版本等

4.3.2 ServletConfig 接口

ServletConfig 接口位于 `javax.servlet` 包中, 它封装了 Servlet 的配置信息, 在 Servlet 初始化期间被传递。每一个 Servlet 都有且只有一个 ServletConfig 对象。此对象定义了 4 个方法, 如表 4.2 所示。

表 4.2 ServletConfig 接口中的方法及说明

方 法	说 明
<code>public String getInitParameter(String name)</code>	此方法返回 String 类型名称为 name 的初始化参数值
<code>public Enumeration getInitParameterNames()</code>	获取所有初始化参数名的枚举集合
<code>public ServletContext getServletContext()</code>	用于获取 Servlet 上下文对象
<code>public String getServletName()</code>	返回 Servlet 对象的实例名

4.3.3 HttpServletRequest 接口

HttpServletRequest 接口位于 `javax.servlet.http` 包中, 继承了 `javax.servlet.ServletRequest` 接口, 是 Servlet 中的重要对象, 在开发过程中较为常用, 其常用方法及说明如表 4.3 所示。

表 4.3 HttpServletRequest 接口的常用方法及说明

方 法	说 明
<code>public String getContextPath()</code>	返回请求的上下文路径, 此路径以 “/” 开关
<code>public Cookie[] getCookies()</code>	返回请求中发送的所有 cookie 对象, 返回值为 cookie 数组
<code>public String getMethod()</code>	返回请求所使用的 HTTP 类型, 如 get、post 等
<code>public String getQueryString()</code>	返回请求中参数的字符串形式, 如请求 <code>MyServlet?username=mr</code> , 则返回 <code>username=mr</code>
<code>public String getRequestURI()</code>	返回主机名到请求参数之间部分的字符串形式
<code>public StringBuffer getRequestURL()</code>	返回请求的 URL, 此 URL 中不包含请求的参数。注意此方法返回的数据类型为 <code>StringBuffer</code>
<code>public String getServletPath()</code>	返回请求 URI 中的 Servlet 路径的字符串, 不包含请求中的参数信息
<code>public HttpSession getSession()</code>	返回与请求关联的 <code>HttpSession</code> 对象

例 4.03 HttpServletRequest 接口的使用。（实例位置：光盘\TM\Instances\4.03）

（1）创建名为 MyServlet 的类（它是一个 Servlet），在此类中通过 PrintWriter 对象向页面中输出调用 HttpServletRequest 接口中的方法所获取的值。其关键代码如下：

```
public class MyServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html");  
        response.setCharacterEncoding("GBK");  
        PrintWriter out = response.getWriter();  
        out.print("<p>上下文路径: " + request.getServletPath() + "</p>");  
        out.print("<p>HTTP 请求类型: " + request.getMethod() + "</p>");  
        out.print("<p>请求参数: " + request.getQueryString() + "</p>");  
        out.print("<p>请求 URI: " + request.getRequestURI() + "</p>");  
        out.print("<p>请求 URL: " + request.getRequestURL().toString() + "</p>");  
        out.print("<p>请求 Servlet 路径: " + request.getServletPath() + "</p>");  
        out.flush();  
        out.close();  
    }  
}
```

（2）在 web.xml 文件中，对 MyServlet 类进行配置。其关键代码如下：

```
<servlet>  
    <servlet-name>MyServlet</servlet-name>  
    <servlet-class>com.lyq.MyServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
    <servlet-name>MyServlet</servlet-name>  
    <url-pattern>/servlet/MyServlet</url-pattern>  
</servlet-mapping>
```

在浏览器地址栏中输入“http://localhost:8080/4.03/servlet/MyServlet?action=test”，运行结果如图 4.6 所示。



图 4.6 实例运行结果

4.3.4 HttpServletResponse 接口

HttpServletResponse 接口位于 javax.servlet.http 包中，它继承了 javax.servlet.ServletResponse 接口，同样是一个非常重要的对象，其常用方法与说明如表 4.4 所示。

表 4.4 HttpServletResponse 接口的常用方法及说明

方 法	说 明
public void addCookie(Cookie cookie)	向客户端写入 cookie 信息
public void sendError(int sc)	发送一个错误状态码为 sc 的错误响应到客户端
public void sendError(int sc, String msg)	发送一个包含错误状态码及错误信息的响应到客户端，参数 sc 为错误状态码，参数 msg 为错误信息
public void sendRedirect(String location)	使用客户端重定向到新的 URL，参数 location 为新的地址

例 4.04 在程序开发过程中，经常会遇到异常的产生，本实例使用 HttpServletResponse 向客户端发送错误信息。（实例位置：光盘\TM\Instances\4.04）

创建一个名称为 MyServlet 的 Servlet 对象，在 doGet() 方法中模拟一个开发过程中的异常，并将其通过 throw 关键字抛出。其关键代码如下：

```
public class MyServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        try {
            //创建一个异常
            throw new Exception("数据库连接失败");
        } catch (Exception e) {
            response.sendError(500, e.getMessage());
        }
    }
}
```

程序中的异常通过 catch 进行捕获，使用 HttpServletResponse 对象的 sendError() 方法向客户端发送错误信息，运行结果如图 4.7 所示。

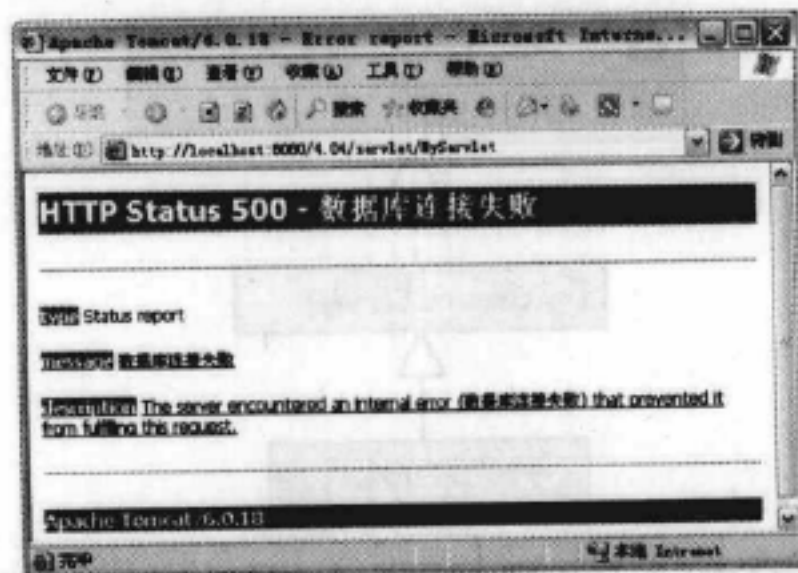


图 4.7 实例运行结果

4.3.5 GenericServlet 类

在编写一个 Servlet 对象时，必须实现 `javax.servlet.Servlet` 接口，但在 Servlet 接口中包含 5 个方法，也就是说创建一个 Servlet 对象要实现这 5 个方法，这样操作非常不方便。`javax.servlet.GenericServlet` 类简化了此操作，实现了 Servlet 接口。

```
public abstract class GenericServlet
    extends Object
    implements Servlet, ServletConfig, Serializable
```

`GenericServlet` 类是一个抽象类，分别实现了 `Servlet` 接口与 `ServletConfig` 接口。此类实现了除 `service()` 之外的其他方法，在创建 Servlet 对象时，可以继承 `GenericServlet` 类来简化程序中的代码，但需要实现 `service()` 方法。

4.3.6 HttpServlet 类

`GenericServlet` 类实现了 `javax.servlet.Servlet` 接口，为程序的开发提供了方便；但在实际开发过程中，大多数的应用都是使用 Servlet 处理 HTTP 协议的请求，并对请求作出响应，所以通过继承 `GenericServlet` 类仍然不是很方便。`javax.servlet.http.HttpServlet` 类对 `GenericServlet` 类进行了扩展，为 HTTP 请求的处理提供了灵活的方法。

```
public abstract class HttpServlet
    extends GenericServlet implements Serializable
```

`HttpServlet` 类仍然是一个抽象类，实现了 `service()` 方法，并针对 HTTP 1.1 中定义的 7 种请求类型提供了相应的方法——`doGet()` 方法、`doPost()` 方法、`doPut()` 方法、`doDelete()` 方法、`doHead()` 方法、`doTrace()` 方法、`doOptions()` 方法。在这 7 个方法中，除了对 `doTrace()` 方法与 `doOptions()` 方法进行简单实现外，`HttpServlet` 类并没有对其他方法进行实现，需要开发人员在使用过程中根据实际需要对其进行重写。

`HttpServlet` 类继承了 `GenericServlet` 类，通过其对 `GenericServlet` 类的扩展，可以很方便地对 HTTP 请求进行处理及响应。该类与 `GenericServlet` 类、`Servlet` 接口的关系如图 4.8 所示。

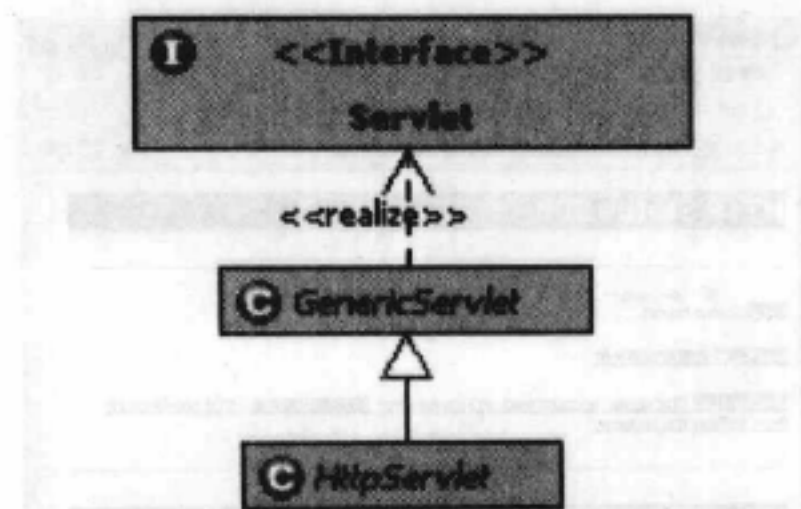


图 4.8 HttpServlet 类与 GenericServlet 类、Servlet 接口的关系

4.4 Servlet 过滤器

 视频讲解：光盘\TM\Video\4\Servlet 过滤器.exe

过滤器是 Web 程序中的可重用组件，在 Servlet 2.3 规范中被引入，应用十分广泛，给 Java Web 程序的开发带来了更加强大的功能。本节将介绍 Servlet 过滤器的结构体系及其在 Web 项目中的应用。

4.4.1 过滤器概述

Servlet 过滤器是客户端与目标资源间的中间层组件，用于拦截客户端的请求与响应信息，如图 4.9 所示。当 Web 容器接收到一个客户端请求时，将判断此请求是否与过滤器对象相关联，如果相关联，则将这一请求交给过滤器进行处理。在处理过程中，过滤器可以对请求进行操作，如更改请求中的信息数据。在过滤器处理完成之后，再将这一请求交给其他业务进行处理。当所有业务处理完成，需要对客户端进行响应时，容器又将响应交给过滤器进行处理，过滤器完成处理后将响应发送到客户端。

在 Web 程序开发过程中，可以放置多个过滤器，如字符编码过滤器、身份验证过滤器等，Web 容器对多个过滤器的处理方式如图 4.10 所示。

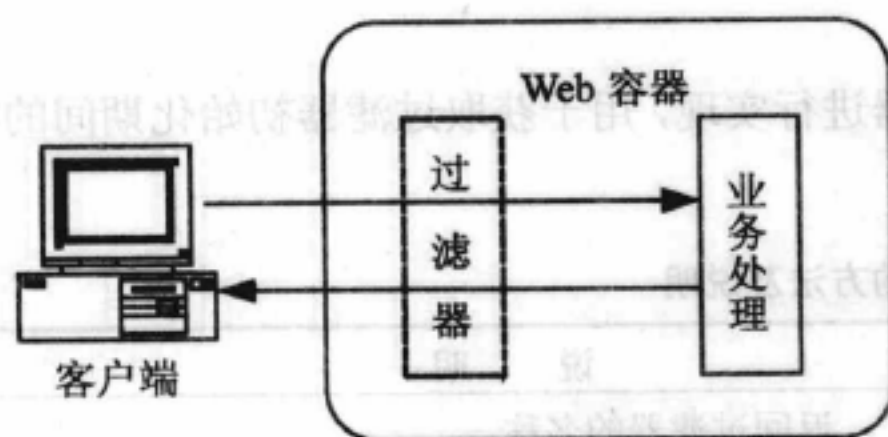


图 4.9 过滤器的应用

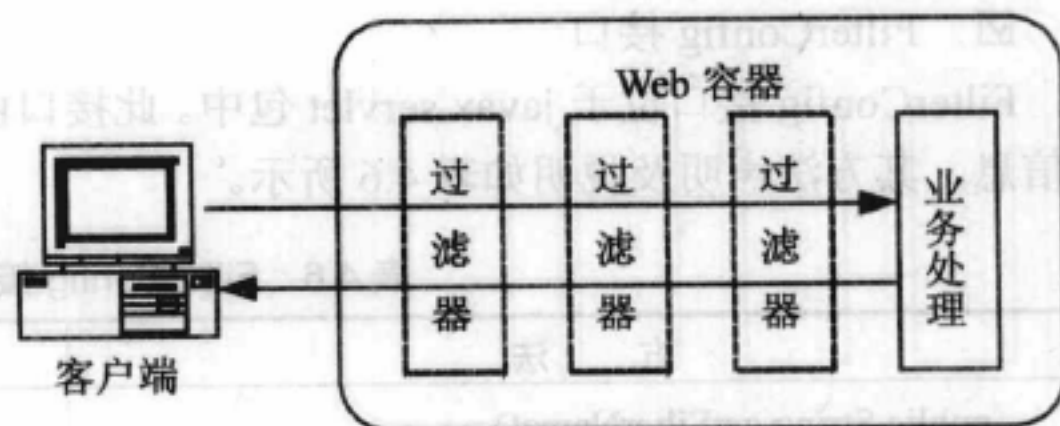


图 4.10 多个过滤器的应用

在多个过滤器的处理方式中，容器首先将客户端请求交给第一个过滤器处理，处理完成之后交给下一个过滤器处理，以此类推，直到最后一个过滤器。当需要对客户端回应时，如图 4.10 所示，将按照相反的方向对回应进行处理，直到交给第一个过滤器，最后发送到客户端回应。

4.4.2 Filter API

过滤器与 Servlet 非常相似，它的使用主要是通过 3 个核心接口分别为 Filter 接口、FilterChain 接口与 FilterConfig 接口进行操作。

☒ Filter 接口

Filter 接口位于 javax.servlet 包中，与 Servlet 接口相似，当定义一个过滤器对象时需要实现此接口。在 Filter 接口中包含 3 个方法，其方法声明及作用如表 4.5 所示。

表 4.5 Filter 接口中的方法及说明

方 法	说 明
<code>public void init(FilterConfig filterConfig)</code>	过滤器的初始化方法，容器调用此方法完成过滤的初始化。对于每一个 Filter 实例，此方法只被调用一次
<code>public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)</code>	此方法与 Servlet 的 <code>service()</code> 方法相类似，当请求及响应交给过滤器时，过滤器调用此方法进行过滤处理
<code>public void destroy()</code>	在过滤器生命周期结束时调用此方法，用于释放过滤器所占用的资源

☑ FilterChain 接口

FilterChain 接口位于 `javax.servlet` 包中，此接口由容器进行实现，在 FilterChain 接口只包含一个方法，其方法声明如下：

```
void doFilter(ServletRequest request,
              ServletResponse response)
              throws IOException,
              ServletException
```

此方法主要用于将过滤器处理的请求或响应传递给下一个过滤器对象。在多个过滤器的 Web 应用中，可以通过此方法进行传递。

☑ FilterConfig 接口

FilterConfig 接口位于 `javax.servlet` 包中。此接口由容器进行实现，用于获取过滤器初始化期间的参数信息，其方法声明及说明如表 4.6 所示。

表 4.6 FilterConfig 接口中的方法及说明

方 法	说 明
<code>public String getFilterName()</code>	返回过滤器的名称
<code>public String getInitParameter(String name)</code>	返回初始化名称为 name 的参数值
<code>public Enumeration getInitParameterNames()</code>	返回所有初始化参数名的枚举集合
<code>public ServletContext getServletContext()</code>	返回 Servlet 的上下文对象

了解了过滤器的这 3 个核心接口，就可以通过实现 Filter 接口来创建一个过滤器对象。其代码结构如下：

```
public class MyFilter implements Filter {
    //初始化方法
    public void init(FilterConfig arg0) throws ServletException {
    }
    //过滤处理方法
    public void doFilter(ServletRequest request, ServletResponse response,
                        FilterChain chain) throws IOException, ServletException {
        //传递给下一个过滤器
        chain.doFilter(request, response);
    }
    //销毁方法
}
```



```
public void destroy() {
    }
}
```

4.4.3 过滤器的配置

在创建一个过滤器对象之后，需要对其进行配置才可以使用。过滤器的配置方法与 Servlet 的配置方法相类似，都是通过 web.xml 文件进行配置，具体步骤如下。

(1) 声明过滤器对象

在 web.xml 文件中，通过<filter>标签声明一个过滤器对象。在此标签下包含 3 个常用子元素，分别为<filter-name>、<filter-class>和<init-param>。其中，<filter-name>元素用于指定过滤器的名称，此名称可以为自定义的名称；<filter-class>元素用于指定过滤器对象的完整位置，包含过滤器对象的包名与类名；<init-param>元素用于设置过滤器的初始化参数。其配置方法如下：

```
<filter>
  <filter-name>CharacterEncodingFilter</filter-name>
  <filter-class>com.lyq.util.CharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>GBK</param-value>
  </init-param>
</filter>
```

<init-param>元素包含两个常用的子元素，分别为<param-name>与<param-value>。其中，<param-name>元素用于声明初始化参数的名称，<param-value>元素用于指定初始化参数的值。

(2) 映射过滤器

在 web.xml 文件中声明了过滤器对象后，需要映射访问过滤器的过滤的对象。此操作使用<filter-mapping>标签进行配置。在<filter-mapping>标签中主要需要配置过滤器的名称、过滤器关联的 URL 样式、过滤器对应的请求方式等。其配置方法如下：

```
<filter-mapping>
  <filter-name>CharacterEncodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

- ☑ <filter-name>元素用于指定过滤器的名称，此名称与<filter>标签中的<filter-name>相对应。
- ☑ <url-pattern>元素用于指定过滤器关联的 URL 样式，设置为“/*”表示关联所有 URL。
- ☑ <dispatcher>元素用于指定过滤器对应的请求方式，其可选值及使用说明如表 4.7 所示。

表 4.7 <dispatcher>的可选值及说明

可 选 值	说 明
REQUEST	当客户端直接请求时，通过过滤器进行处理
INCLUDE	当客户端通过 RequestDispatcher 对象的 include()方法请求时，通过过滤器进行处理

续表

可 选 值	说 明
FORWARD	当客户端通过 RequestDispatcher 对象的 forward()方法请求时, 通过过滤器进行处理
ERROR	当声明式异常产生时, 通过过滤器进行处理

4.4.4 过滤器典型应用

在 Java Web 项目的开发中, 过滤器的应用十分广泛, 其中比较典型的应用就是字符编码过滤器。由于 Java 程序可以在多种平台下运行, 其内部使用 Unicode 字符集来表示字符, 所以处理中文数据会产生乱码的情况, 需要对其进行编码转换才可以正常显示。

例 4.05 字符编码过滤器。(实例位置: 光盘\TM\Instances\4.05)

(1) 创建字符编码过滤器类 CharacterEncodingFilter, 此类实现了 Filter 接口, 并对其 3 个方法进行了实现。关键代码如下:

```
public class CharacterEncodingFilter implements Filter {
    // 字符编码(初始化参数)
    protected String encoding = null;
    // FilterConfig 对象
    protected FilterConfig filterConfig = null;
    // 初始化方法
    public void init(FilterConfig filterConfig) throws ServletException {
        // 对 filterConfig 赋值
        this.filterConfig = filterConfig;
        // 对初始化参数赋值
        this.encoding = filterConfig.getInitParameter("encoding");
    }
    // 过滤器处理方法
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
        // 判断字符编码是否有效
        if (encoding != null) {
            // 设置 request 字符编码
            request.setCharacterEncoding(encoding);
            // 设置 response 字符编码
            response.setContentType("text/html; charset="+encoding);
        }
        // 传递给下一过滤器
        chain.doFilter(request, response);
    }
    // 销毁方法
    public void destroy() {
        // 释放资源
        this.encoding = null;
        this.filterConfig = null;
    }
}
```


CharacterEncodingFilter 类的 init()方法用于读取过滤器的初始化参数,这个参数(encoding)为本例中所用到的字符编码;在 doFilter()方法中,分别将 request 对象及 response 对象中的编码格式设置为读取到的编码格式;最后在 destroy()方法中将其属性设置为 null,将被 Java 垃圾回收器回收。

(2) 在 web.xml 文件中,对过滤器进行配置。其关键代码如下:

```
<!-- 声明字符编码过滤器 -->
<filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>com.lyq.util.CharacterEncodingFilter</filter-class>
    <!-- 设置初始化参数 -->
    <init-param>
        <param-name>encoding</param-name>
        <param-value>GBK</param-value>
    </init-param>
</filter>
<!-- 映射字符编码过滤器 -->
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <!-- 与所有请求关联 -->
    <url-pattern>/*</url-pattern>
    <!-- 设置过滤器对应的请求方式 -->
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

在 web.xml 配置文件中,需要对过滤器进行声明及映射,其中声明过程通过<init-param>指定了初始化参数的字符编码为 GBK。

(3) 通过请求对过滤器进行验证。本例中使用表单向 Servlet 发送中文信息进行测试,其中表单信息放置在 index.jsp 页面中。其关键代码如下:

```
<form action="MyServlet" method="post">
    <p>
        请输入你的中文名字:
        <input type="text" name="name">
        <input type="submit" value="提交">
    </p>
</form>
```

这一请求由 Servlet 对象 MyServlet 类进行处理,此类使用 doPost()方法接收表单的请求,并将表单中的 name 属性输出到页面中。其关键代码如下:

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    //获取表单参数
    String name = request.getParameter("name");
    if(name != null && !name.isEmpty()){
        out.print("你好 " + name);
    }
}
```



```

        out.print("<br>欢迎来到我的主页。");
    }else{
        out.print("请输入你的中文名字!");
    }
    out.print("<br><a href=index.jsp>返回</a>");
    out.flush();
    out.close();
}

```

实例运行结果如图 4.11 所示,输入中文“明日科技”进行测试,其经过过滤器处理的效果如图 4.12 所示,没有经过过滤器处理的效果如图 4.13 所示。

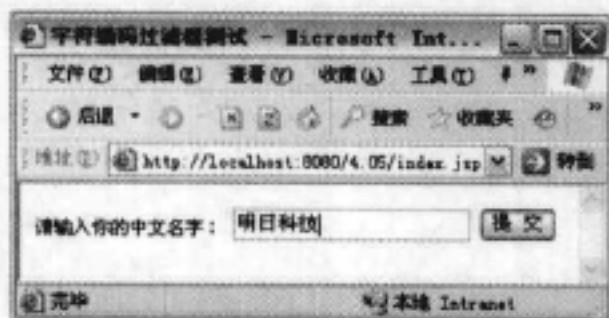


图 4.11 实例运行结果

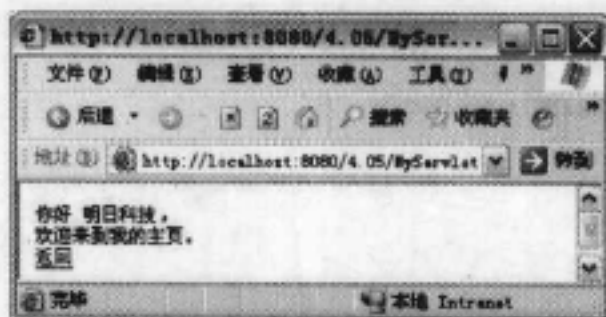


图 4.12 过滤后的效果

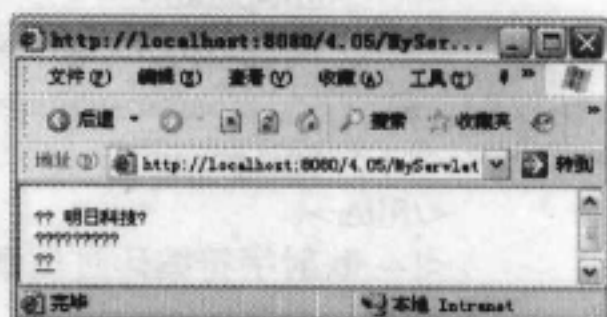



图 4.13 未经过滤的效果

4.5 实 战

Servlet 是使用 Java Servlet 接口 (API) 运行在 Web 应用服务器上的 Java 程序,其应用十分广泛,在 Java Web 项目的开发中非常重要。本节将结合具体实例全面讲解 Servlet 技术及 Servlet 过滤器技术的实际应用。

4.5.1 JSP 与 Servlet 实现用户注册

 视频讲解: 光盘\TM\Video\04\JSP 与 Servlet 实现用户注册.exe

用户注册模块是网站中经常用到的,通过它可对网站的来访用户进行管理,如用户身份认证、用户对网站的操作权限等。下面以用户注册为例,向读者介绍 Servlet 技术的实际应用方法。

例 4.06 JSP 与 Servlet 实现用户注册。(实例位置: 光盘\TM\Instances\4.06)

(1) 创建数据表 tb_user,用于存储用户的注册信息,其结构如图 4.14 所示。

Column Name	Datatype	PK	Null	Flags	Default Value	Comment
id	INTEGER	✓	✓	UNSIGNED <input checked="" type="checkbox"/> ZEROFILL <input type="checkbox"/>		主键
username	VARCHAR(45)	✓		BINARY <input type="checkbox"/>		用户名
password	VARCHAR(45)	✓		BINARY <input type="checkbox"/>		密码
sex	VARCHAR(45)	✓		BINARY <input type="checkbox"/>		性别
question	VARCHAR(45)	✓		BINARY <input type="checkbox"/>		密码问题
answer	VARCHAR(45)	✓		BINARY <input type="checkbox"/>		密码答案
email	VARCHAR(45)	✓		BINARY <input type="checkbox"/>		邮箱

图 4.14 表 tb_user 的结构

(2) 创建名为 RegServlet 的类,用于处理用户注册请求(它是一个 Servlet 对象)。在此类中重写 init()方法与 doPost()方法,其中在 init()方法中获取数据库连接。其关键代码如下:


```

//数据库连接 Connection
private Connection conn;
//初始化方法
public void init() throws ServletException {
    super.init();
    try {
        //加载驱动
        Class.forName("com.mysql.jdbc.Driver");
        //数据库连接 url
        String url = "jdbc:mysql://localhost:3306/db_database04";
        //获取数据库连接
        conn = DriverManager.getConnection(url, "root", "111");
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

init()方法是Servlet的初始化方法，此方法只运行一次，实例中在此方法中加载数据库驱动，并获取数据库连接对象Connection。在获取数据库连接对象之后，通过doPost()方法处理用户注册请求。其关键代码如下：

```

public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //设置 request 与 response 的编码
    response.setContentType("text/html");
    request.setCharacterEncoding("GBK");
    response.setCharacterEncoding("GBK");
    //获取表单中属性值
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    String sex = request.getParameter("sex");
    String question = request.getParameter("question");
    String answer = request.getParameter("answer");
    String email = request.getParameter("email");
    //判断数据库是否连接成功
    if (conn != null) {
        try {
            //插入注册信息的 SQL 语句(使用?占位符)
            String sql = "insert into tb_user(username,password,sex,question,answer,email) "
                + "values(?,?,?,?,?,?)";
            //创建 PreparedStatement 对象
            PreparedStatement ps = conn.prepareStatement(sql);
            //对 SQL 语句中的参数动态赋值
            ps.setString(1, username);
            ps.setString(2, password);
            ps.setString(3, sex);
            ps.setString(4, question);
            ps.setString(5, answer);
            ps.setString(6, email);

```



```

        //执行更新操作
        ps.executeUpdate();
        //获取 PrintWriter 对象
        PrintWriter out = response.getWriter();
        //输出注册结果信息
        out.print("<h1 aling='center'>");
        out.print(username + "注册成功! ");
        out.print("</h1>");
        out.flush();
        out.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
} else {
    //发送数据库连接错误提示信息
    response.sendError(500, "数据库连接错误!");
}
}

```

由于本实例并没有配置字符编码过滤器, 所以需要将 response 与 request 对象的字符编码设置为 GBK, 否则处理中文将出现乱码。request 对象的 getParameter() 方法用于获取请求中的参数值, 实例中使用此方法获取用户的注册信息, 在获取后将其写入到数据库之中。RegServlet 类的 Servlet 配置代码如下:

```

<servlet>
    <servlet-name>RegServlet</servlet-name>
    <servlet-class>com.lyq.RegServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>RegServlet</servlet-name>
    <url-pattern>/RegServlet</url-pattern>
</servlet-mapping>

```

(3) 创建 index.jsp 页面 (程序中的首页), 在此页面中放置用户注册所需要的表单。其关键代码如下:

```

<form action="RegServlet" method="post" onsubmit="return reg(this);">
    <table align="center" border="0" width="500">
        <tr>
            <td align="right" width="30%">用户名: </td>
            <td><input type="text" name="username" class="box"></td>
        </tr>
        <tr>
            <td align="right">密 码: </td>
            <td><input type="password" name="password" class="box"></td>
        </tr>
        <tr>
            <td align="right">确认密码: </td>
            <td><input type="password" name="repassword" class="box"></td>
        </tr>
    </table>
</form>

```



```

</tr>
<tr>
 ☒  </tr> <tr>  <input type="text" name="question" class="box"></td>  </tr> <tr>  <input type="text" name="answer" class="box"></td>  </tr> <tr>  <input type="text" name="email" class="box"></td>  </tr> <tr>  </tr> </table> </form> | | | | | |
```

此表单的提交地址为 RegServlet, 其请求方法为 post, 即它将由映射到 RegServlet 类的 post 方法进行处理。本实例的主页运行结果如图 4.15 所示, 正确填写用户信息后, 单击“注册”按钮, 用户注册信息将被写入到数据库之中。

天梯海论坛
tan ti hai lun tan

用户注册

用户名:

密码:

确认密码:

性别: ☒男 ☐女

密码找回问题:

密码找回答案:

邮箱:

服务器地址: 2431-0678081 8080808 808-075-1005
网址: http://www.singsoft.com http://www.mibook.com

图 4.15 实例运行结果

4.5.2 过滤非法文字

 视频讲解：光盘\TM\Video\04\过滤非法文字.exe

在一些大型网站中，首先需要设有非法文字过滤功能，如色情关键字、脏话等，以免产生不良影响。在用户发布信息时，首先需要对发布的内容进行过滤。但网站中提交请求很多，如果对每一个请求都加入过滤代码来实现，未免过于繁琐。在下面的实例中，将通过使用 Servlet 过滤器对所有请求进行非法文字过滤。

例 4.07 过滤非法文字。（实例位置：光盘\TM\Instances\4.07）

(1) 创建名为 WordFilter 的类，此类实现了 Filter 接口，是非法文字的过滤器。此过滤器的功能比较强大，不仅可对非法文字进行过滤处理，还可对字符编码的转换进行处理，所以在 WordFilter 类中定义了非法字符数组属性 words 与字符编码属性 encoding，并在过滤器的初始化方法 init() 中对其进行实例化。其关键代码如下：

```
public class WordFilter implements Filter {
    //非法字符数组
    private String words[];
    //字符编码
    private String encoding;
    //实现 Filter 接口 init()方法
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        //获取字符编码
        encoding = filterConfig.getInitParameter("encoding");
        //初始化非法字符数组
        words = new String[]{"糟糕","混蛋"};
    }
    //省略其他代码
}
```

非法字符包含多个，所以需要对其进行逐一处理。在此创建 filter() 方法，此方法通过循环非法字符对提交内容逐一过滤，将非法字符替换为“****”。其关键代码如下：

```
public String filter(String param){
    try {
        //判断非法字符是否被初始化
        if(words != null && words.length > 0){
            //循环替换非法字符
            for (int i = 0; i < words.length; i++) {
                //判断是否包含非法字符
                if(param.indexOf(words[i]) != -1){
                    //将非法字符替换为"****"
                    param = param.replaceAll(words[i], "****");
                }
            }
        }
    }
}
```



```

    } catch (Exception e) {
        e.printStackTrace();
    }
    return param;
}

```

在网站的信息发布中, 使用 `ServletRequest` 对象获取表单所提交的数据 (主要通过 `getParameter()` 方法与 `getParameterValues()` 方法获取)。在此创建内部类 `Request`, 重写了 `getParameter()` 方法与 `getParameterValues()` 方法, 并在重写的这两个方法中实现过滤。其关键代码如下:

```

class Request extends HttpServletRequestWrapper{
    //构造方法
    public Request(HttpServletRequest request) {
        super(request);
    }
    //重写 getParameter()方法
    @Override
    public String getParameter(String name) {
        //返回过滤后的参数值
        return filter(super.getRequest().getParameter(name));
    }
    //重写 getParameterValues()方法
    @Override
    public String[] getParameterValues(String name) {
        //获取所有参数值
        String[] values = super.getRequest().getParameterValues(name);
        //通过循环对所有参数值进行过滤
        for (int i = 0; i < values.length; i++) {
            values[i] = filter(values[i]);
        }
        //返回过滤后的参数值
        return values;
    }
}

```

⚠ 注意: `HttpServletRequestWrapper` 类是 `ServletRequest` 接口的实现类, 实例创建内部类 `Request` 继承 `HttpServletRequestWrapper` 类, 并重写其获取参数值的两个方法。

在过滤器的 `doFilter()` 方法中, 将传递的 `ServletRequest` 对象转换为自定义的对象 `Request`, 即可实现非法字符的过滤。其关键代码如下:

```

//实现 Filter 接口的 doFilter()方法
@Override
public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain) throws IOException, ServletException {
    //判断字符编码是否有效
    if (encoding != null) {
        //设置 request 字符编码
        request.setCharacterEncoding(encoding);
    }
}

```



```

        //将 request 转换为重写后的 Request 对象
        request = new Request((HttpServletRequest) request);
        //设置 response 字符编码
        response.setContentType("text/html; charset="+encoding);
    }
    chain.doFilter(request, response);
}

```

最后通过 `destroy()` 方法释放过滤器中的资源，其关键代码如下。

```

//实现 Filter 接口的 destroy() 方法
@Override
public void destroy() {
    this.words = null;
    this.encoding = null;
}

```

(2) 创建处理用户留言反馈的 Servlet 对象 `MessageServlet` 类，此类使用 `doPost()` 方法对用户留言信息进行处理。其关键代码如下：

```

public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //获取标题
    String title = request.getParameter("title");
    //获取内容
    String content = request.getParameter("content");
    //将标题放置到 request 中
    request.setAttribute("title", title);
    //将内容放置到 request 中
    request.setAttribute("content", content);
    //转发到 result.jsp 页面
    request.getRequestDispatcher("index.jsp").forward(request, response);
}

```

(3) 对 Servlet 以及过滤器进行统一配置，其配置信息写入到 `web.xml` 文件中。其关键代码如下：

```

<!-- Servlet 配置 -->
<servlet>
    <servlet-name>MessageServlet</servlet-name>
    <servlet-class>com.lyq.MessageServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>MessageServlet</servlet-name>
    <url-pattern>/MessageServlet</url-pattern>
</servlet-mapping>
<!-- 过滤器配置 -->
<filter>
    <filter-name>WordFilter</filter-name>
    <filter-class>com.lyq.WordFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>

```



```

        <param-value>GBK</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>WordFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

```

(4) 创建 index.jsp 页面 (程序的首页), 用于显示用户的留言反馈信息以及留言反馈的表单, 其中显示留言的关键代码如下。

```

<%
    String title = (String)request.getAttribute("title");
    String content = (String)request.getAttribute("content");
    if(title != null && !title.isEmpty()){
        out.println("<span class='tl'>" + title + "</span>");
    }
    if(content != null && !content.isEmpty()){
        out.println("<span class='ct'>" + content + "</span>");
    }
%>

```

此页面运行结果如图 4.16 所示, 输入实例中所设置的非法文字, 单击“提交”按钮, 所输入的非法文字将以“****”的形式显示。

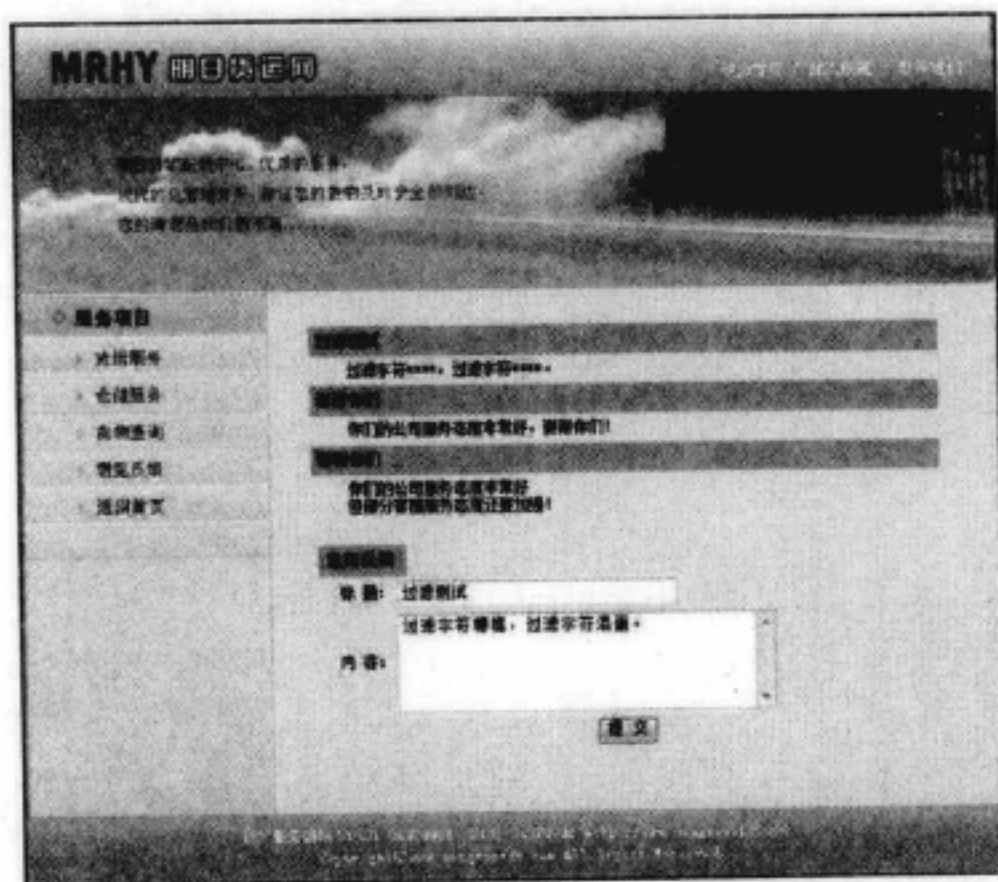


图 4.16 实例运行结果

4.6 本章小结

本章主要向读者介绍了 Servlet 与 Servlet 过滤器的应用。这两项技术十分重要, 都是 J2EE 开发必须要掌握的知识。学习 Servlet 的使用, 需要掌握 Servlet API 中的主要接口及实现类、Servlet 的生命周

期以及 doXXX() 方法对 Http 请求的处理。对于 Servlet 过滤器的应用, 要理解实现过滤的原理, 以保证在实际应用过程中合理地使用。

4.7 实战练习

1. 简易 Servlet 计算器。(答案位置: 光盘\TM\Instances\4.08)
2. 过滤器验证用户登录。(答案位置: 光盘\TM\Instances\4.09)
3. 过滤器统计流量。(答案位置: 光盘\TM\Instances\4.10)
4. JSP 与 Servlet 实现用户登录。(答案位置: 光盘\TM\Instances\4.11)

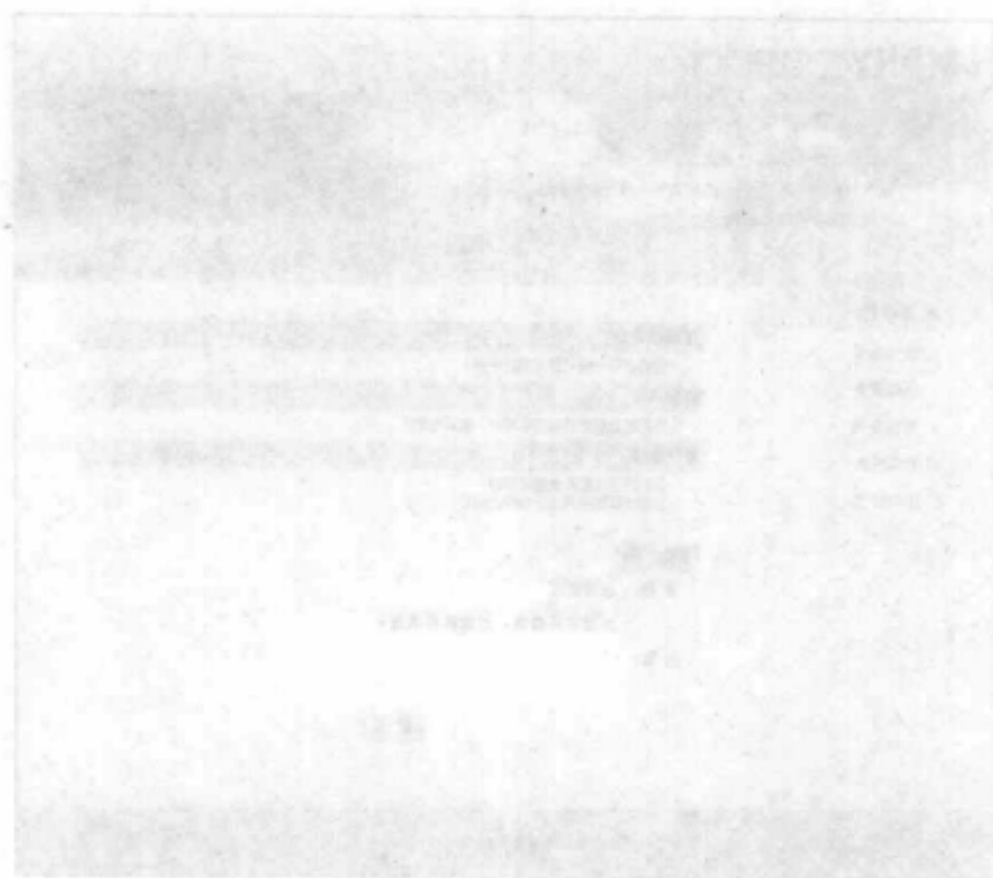



图 4.10 简易计算器

4.6 本章小结

本章主要介绍了 Servlet 与 JSP 的结合使用, 以及如何在 Web 应用中实现用户登录、流量统计等功能。通过本章的学习, 读者应该能够掌握 Servlet 与 JSP 的基本用法, 并能够独立完成相关的开发任务。

第 5 章

JSP 使用 Model2 实现登录模块

( 视频讲解：48 分钟)

在 JSP 开发过程中有两种开发模型可供选择：一种是 JSP 与 JavaBean 相结合，这种方式称为 Model1；另外一种 JSP、JavaBean 与 Servlet 相结合，这种方式称为 Model2。本章将针对这两种开发模型对 JSP 的架构方式进行详细讲解，并结合实例分析两种模型的优缺点。

通过阅读本章，您可以：

- » 掌握 `<jsp:userBean>` JSP 动作标签用法
- » 掌握 `<jsp:setProperty>` JSP 动作标签用法
- » 掌握 `<jsp:getProperty>` JSP 动作标签用法
- » 掌握 JavaBean 的作用域
- » 掌握 Model1 开发模式
- » 掌握 Model2 开发模式
- » 理解 MVC 设计原理

5.1 JavaBean

 视频讲解：光盘\TM\Video\5\JavaBean.exe

在 JSP 网页开发的初级阶段，并没有所谓的框架与逻辑分层的概念，JSP 网页代码是与业务逻辑代码写在一起的。这种零乱的代码书写方式，给程序的调试及维护带来了很大的困难，直至 JavaBean 的出现，这一问题才得到了些许改善。

5.1.1 JavaBean 简介

JavaBean 是用于封装某种业务逻辑或对象的 Java 类，此类具有特定的功能，即它是一个可重用的 Java 软件组件模型。由于这些组件模型都具有特定的功能，将其进行合理的组织后，可以快速生成一个全新的程序，实现代码的重用。JavaBean 的功能是没有任何限制的，对于任何可以使用 Java 代码实现的部分或需求的对象，都可以使用 JavaBean 进行封装，如创建一个实体对象、数据库操作、字符串操作等。它对简单或复杂的功能都可以进行实现。

JavaBean 可分为两类，即可视化的 JavaBean 与非可视化的 JavaBean。可视化的 JavaBean 是一种传统的应用方式，主要用于实现一些可视化界面，如一个窗体、按钮、文本框等。非可视化的 JavaBean 主要用于实现一些业务逻辑或封装一些业务对象，并不存在可视化的界面。此种方式的应用比较多，在 JSP 编程之中被大量采用。

将 JavaBean 应用到 JSP 编程中，使 JSP 的发展进入了一个崭新的阶段。它将 HTML 网页代码与 Java 代码相分离，使其业务逻辑变得更加清晰。在 JSP 页面中，可以通过 JSP 提供的动作标签来操作 JavaBean 对象。其中主要包括<jsp:userBean>、<jsp:setProperty>与<jsp:getProperty>3 个标签，这 3 个标签为 JSP 内置的动作标签。在使用过程中，不需要引入任何第三方的类库。

5.1.2 <jsp:userBean>

<jsp:userBean>标签用于在 JSP 页面中创建一个 JavaBean 实例，并通过属性的设置将此实例存放到 JSP 指定的范围内。其语法格式如下：

```
<jsp:userBean
    id="变量名"
    scope="page|request|session|application"
    {
        class="完整类名"|
        type="数据类型"|
        class="完整类名" type="数据类型"|
        beanName="完整类名" type="数据类型"
    }>
```

<jsp:userBean>语法中的“完整类名”包含一个类的包名称，如 com.lyq.user。参数说明如下。

- ☑ id 属性: 用于定义一个变量名 (可以理解为 JavaBean 的一个代号), 程序中通过此变量名对 JavaBean 进行引用。
- ☑ scope 属性: 设置 JavaBean 的作用域。它有 4 种范围, 即 page、request、session、application, 默认情况下为 page。
- ☑ class 属性: 指定 JavaBean 的完整类名 (包名与类名结合的方式), 如 class="com.lyq.User"。此属性与 BeanName 属性不能同时存在。
- ☑ type 属性: 指定 id 属性所定义的变量类型。
- ☑ beanName 属性: 指定 JavaBean 的完整类名, 此属性不能与 class 属性同时存在。

例 5.01 在 JSP 页面中实例化一个 JavaBean 对象。(实例位置: 光盘\TM\Instances\5.01)

首先创建一个名为 Bean 的类 (它是一个 JavaBean), 此类中有一个名为 name 的属性及相应的 getXXX()与 setXXX()方法。代码如下:

```
package com.lyq;
public class Bean {
    private String name;
    public Bean(){
    }
    public String getName() {
        return name + " 的 JavaBean 程序! ";
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

创建了 JavaBean 之后, 在 index.jsp 页面中通过<jsp:useBean>标签实例化此对象, 并调用此对象的方法。代码如下:

```
<%@ page language="java" contentType="text/html" pageEncoding="GBK"%>
<jsp:useBean id="bean" class="com.lyq.Bean"></jsp:useBean>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>主页</title>
</head>
<body>
<%
    bean.setName("Tom");
%>
<h1 align="center"><%=bean.getName()%></h1>
</body>
</html>
```

此 JavaBean 实例非常简单, 在实例化 Bean 对象后, 对其 name 属性赋值, 并通过 getName()方法在网页中输出结果信息, 如图 5.1 所示。

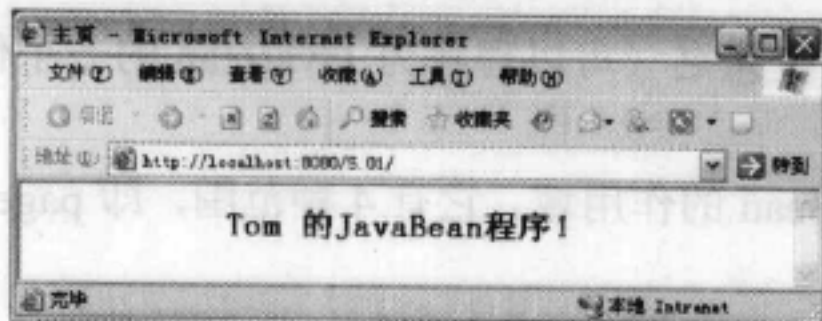


图 5.1 index.jsp 运行结果

5.1.3 <jsp:setProperty>

<jsp:setProperty>标签用于对 JavaBean 中的属性赋值，但 JavaBean 的属性要提供相应的 setXXX() 方法。通常情况下该标签与<jsp:useBean>标签配合使用。其语法格式如下：

```
<jsp:setProperty
  name="实例名"
  {
    property="*" |
    property="属性名" |
    property="属性名" param="参数名" |
    property="属性名" value="值"
  }/>
```

<jsp:setProperty>标签中的属性说明如下。

- ☒ name 属性：指定 JavaBean 的引用名称。
- ☒ property 属性：指定 JavaBean 中的属性名，此属性是必需的，其取值有两种，分别为“*”、“JavaBean 的属性名称”。
- ☒ param 属性：指定 JSP 请求中的参数名，通过该参数可以将 JSP 请求参数的值赋给 Java 的属性。
- ☒ value 属性：指定一个属性的值。

例 5.02 使用“property=“*””对 JavaBean 的属性赋值。（实例位置：光盘\TM\Instances\5.02）

(1) 创建一个名为 Student 的类（它是一个 JavaBean），此类封装了一个学生实体对象。其关键代码如下：

```
public class Student {
    private int id;        //学号
    private int age;       //年龄
    private String classes; //班级
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
```



```

        this.age = age;
    }
    public String getClasses() {
        return classes;
    }
    public void setClasses(String classes) {
        this.classes = classes;
    }
}

```

(2) 创建一个 form 表单，将其放置在 index.jsp 页面中。其关键代码如下：

```

<form action="student.jsp" method="post">
    <p>学 号: <input type="text" name="id"></p>
    <p>年 龄: <input type="text" name="age"></p>
    <p>
        <input type="submit" value="提 交">
        <input type="reset" value="重 置">
    </p>
</form>

```

(3) 创建 student.jsp 页面，用于实例化 Student 对象，并输出相应的属性值。其关键代码如下：

```

<body>
    <jsp:useBean id="student" class="com.lyq.Student"></jsp:useBean>
    <jsp:setProperty name="student" property="*" />
    <div align="center">
        <p>学号: <%=student.getId()%></p>
        <p>年龄: <%=student.getAge()%></p>
    </div>
</body>

```

student.jsp 页面通过 property="*" 属性将请求中的参数与 JavaBean 中的属性进行匹配，并对其赋值。使用此种方式要注意的是，请求中的参数必须与 JavaBean 的属性名相同。

实例运行结果如图 5.2 所示，填写表单信息后，单击“提交”按钮，请求被发送到 student.jsp 页面，此页面将对 Student 类进行实例化，并通过 <jsp:setProperty name="student" property="*" /> 对 Student 类中的属性赋值，运行结果如图 5.3 所示。



图 5.2 index.jsp 页面

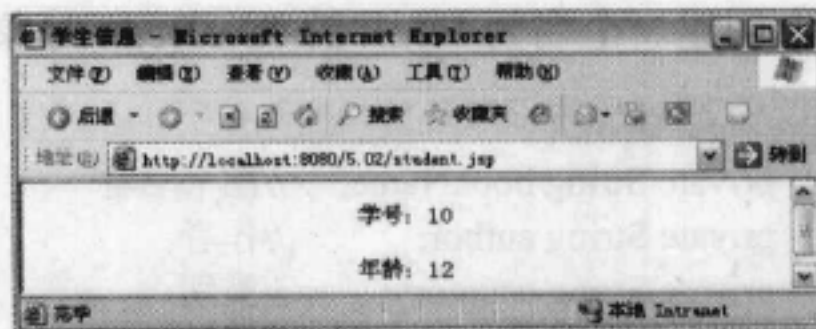


图 5.3 实例运行结果

例 5.03 对 JavaBean 的指定属性赋值。（实例位置：光盘\TM\Instances\5.03）

如果对 JavaBean 的指定属性赋值，可以使用 property="指定属性名"的方式。下面在例 5.02 的基础

上更改 student.jsp 页面，对 Student 类指定属性进行赋值。其关键代码如下：

```
<body>
  <jsp:useBean id="student" class="com.lyq.Student"></jsp:useBean>
  <jsp:setProperty name="student" property="id"/>
  <jsp:setProperty name="student" property="classes" value="一年一班"/>
  <div align="center">
    <p>学号: <%=student.getId()%></p>
    <p>年龄: <%=student.getAge()%></p>
    <p>班级: <%=student.getClasses()%></p>
  </div>
</body>
```

此页面在实例化 Student 类后，通过<jsp:setProperty>标签分别对 Student 类的 id、classes 属性赋值。其中 Student 类的 classes 属性使用了固定值，它通过<jsp:setProperty>标签的 value 属性进行实现。在填写表单后，实例运行结果如图 5.4 所示。由于实例中并没有对 Student 类的 age 属性赋值，所以它保持 int 型的默认值 0。

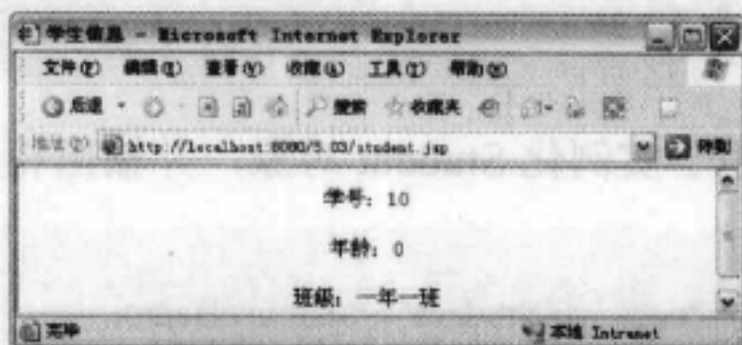


图 5.4 实例运行结果

5.1.4 <jsp:getProperty>

<jsp:getProperty>标签用于获取 JavaBean 中的属性值，但要求 JavaBean 的属性必须具有相对应的 getXXX() 方法。其语法格式如下：

```
<jsp:getProperty name="实例名" property="属性名"/>
```

☑ name 属性：指定存在某一范围的 JavaBean 实例的引用。

☑ property 属性：指定 JavaBean 的属性名称。

例 5.04 利用<jsp:getProperty>标签输出 JavaBean 中的属性。(实例位置：光盘\TM\Instances\5.04)
首先创建一个名为 Book 的 JavaBean 对象，此类用于封装图书信息。其关键代码如下：

```
public class Book {
    private String bookName; //图书名称
    private String author;   //作者
    private String category; //类别
    private double price;    //价格
    public String getBookName() {
        return bookName;
    }
    public void setBookName(String bookName) {
```



```

        this.bookName = bookName;
    }
    //省略 setXXX()方法与 getXXX()方法
}

```

注意：要通过<jsp:getProperty>标签输出 JavaBean 中的属性值，要求在 JavaBean 中必须包含 getXXX() 方法，<jsp:getProperty>标签将通过此方法获取 JavaBean 的属性值。

创建图书对象 Book 后，通过 index.jsp 页面对此对象进行操作。其关键代码如下：

```

<body>
    <!-- 实例化 Book 对象 -->
    <jsp:useBean id="book" class="com.lyq.Book"></jsp:useBean>
    <!-- 对 Book 对象赋值 -->
    <jsp:setProperty name="book" property="bookName" value="《JAVA 程序设计标准教程》"/>
    <jsp:setProperty name="book" property="author" value="明日科技"/>
    <jsp:setProperty name="book" property="category" value="Java 图书"/>
    <jsp:setProperty name="book" property="price" value="59.00"/>
    <table align="center" border="1" cellpadding="1" width="350" height="100" bordercolor="green">
        <tr>
            <td align="right">图书名称:</td>
            <td><jsp:getProperty name="book" property="bookName"/></td>
        </tr>
        <tr>
            <td align="right">作 者:</td>
            <td><jsp:getProperty name="book" property="author"/></td>
        </tr>
        <tr>
            <td align="right">所属类别:</td>
            <td><jsp:getProperty name="book" property="category"/></td>
        </tr>
        <tr>
            <td align="right">价 格:</td>
            <td><jsp:getProperty name="book" property="price"/></td>
        </tr>
    </table>
</body>

```

在此页面中，首先通过<jsp:useBean>标签实例化 Book 对象，再使用<jsp:setProperty>标签对 Book 对象中的属性赋值，最后通过<jsp:getProperty>标签输出 Book 对象的属性值，运行结果如图 5.5 所示。

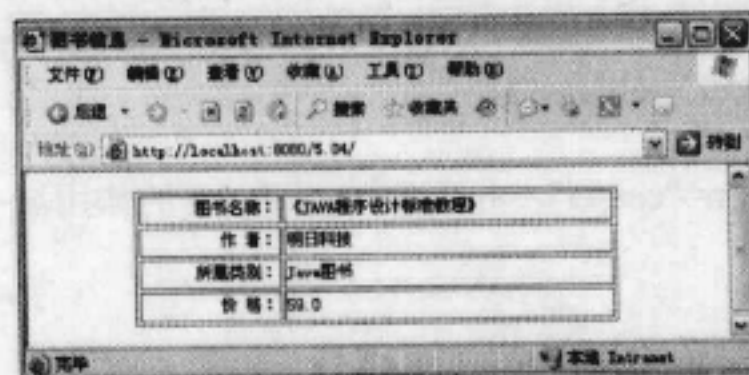


图 5.5 图书信息

5.1.5 JavaBean 的作用域

JavaBean 的生命周期存在于 4 种范围之中，分别为 page、request、session、application，它们通过 `<jsp:useBean>` 标签的 `scope` 属性进行设置。这 4 种范围虽然存在很大的区别，但它们与 JSP 页面中的 page、request、session、application 范围相对应。

- ☑ page 范围：与当前页面相对应，JavaBean 的生命周期存在于一个页面之中，当页面关闭时 JavaBean 被销毁。
- ☑ request 范围：与 JSP 的 request 生命周期相对应，JavaBean 的生命周期存在于 request 对象之中，当 request 对象销毁时 JavaBean 也被销毁。
- ☑ session 范围：与 JSP 的 session 生命周期相对应，JavaBean 的生命周期存在于 session 会话之中，当 session 超时或会话结束时 JavaBean 被销毁。
- ☑ application 范围：与 JSP 的 application 生命周期相对应，在各个用户与服务器之间共享，只有当服务器关闭时 JavaBean 才被销毁。

这 4 种作用范围与 JavaBean 的生命周期是息息相关的，当 JavaBean 被创建后，通过 `<jsp:setProperty>` 标签与 `<jsp:getProperty>` 标签调用时，将会按照 page、request、session 和 application 的顺序来查找这个 JavaBean 实例，直至找到一个实例对象为止，如果在这 4 个范围内都找不到 JavaBean 实例，则抛出异常。

例 5.05 JavaBean 在 session 范围与 application 范围的比较。（实例位置：光盘\TM\Instances\5.05）

本实例通过一个简单的计数器 JavaBean 对 session 范围与 application 范围进行比较，其中计数器的 JavaBean 对象为 Counter 类。其关键代码如下：

```
public class Counter {
    private int count = 0; //访问数量
    public int getCount() {
        return ++count;
    }
}
```

创建计数器对象 Counter 后，在 index.jsp 页面分别创建 session 与 application 范围内的实例对象。其关键代码如下：

```
<body>
<!-- 创建一个 session 范围的 Counter 对象 -->
<jsp:useBean id="counter_session" class="com.lyq.Counter" scope="session"/>
<!-- 创建一个 application 范围的 Counter 对象 -->
<jsp:useBean id="counter_application" class="com.lyq.Counter" scope="application"/>
<table align="center" width="350" border="1">
    <tr>
        <td colspan="2" align="center"><br><h1>JavaBean 的作用域</h1></td>
    </tr>
    <tr>
        <td align="right" width="30%">session</td>
        <td><jsp:getProperty name="counter_session" property="count" /></td>
    </tr>
</table>
```



```

<tr>
  <td align="right">application</td>
  <td><jsp:getProperty name="counter_application" property="count" /></td>
</tr>
</table>
</body>

```

此页面分别输出了 session 范围与 application 范围的计数器的数值，刷新页面后其数值不断自增，如图 5.6 所示，说明 Counter 对象实例存在于此次会话之中。

当开启一个新的浏览器窗口时，session 的生命周期结束，与之对应的 Counter 对象也将被销毁，但 application 范围中的 Counter 对象依然存在，如图 5.7 所示。

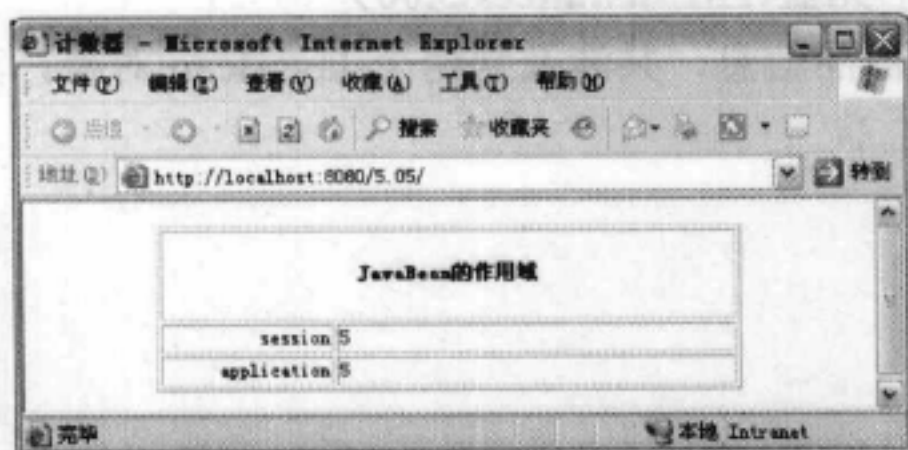


图 5.6 实例运行结果

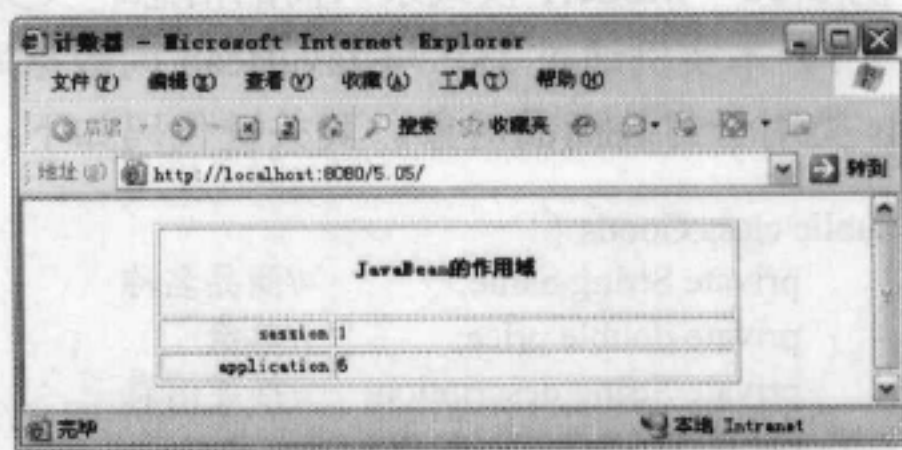


图 5.7 实例运行结果

5.2 Model1 模式

 视频讲解：光盘\TM\Video\5\Model1 模式.exe

JSP 的发展主要经历了两个历程，分别为 Model1 模式、Model2 模式。Model1 模式指的是 Jsp+JavaBean 的程序开发方式，Model2 模式则是指 MVC 的程序开发方式。Model1 模式目前已被遗弃，但比较适合初学者学习 JSP 程序，本节将对其进行简单介绍。

Model1 模式与纯 JSP 开发方式相比是一次进步。在纯 JSP 开发方式中，JSP 网页代码与所有业务逻辑代码写在一起，如图 5.8 所示。

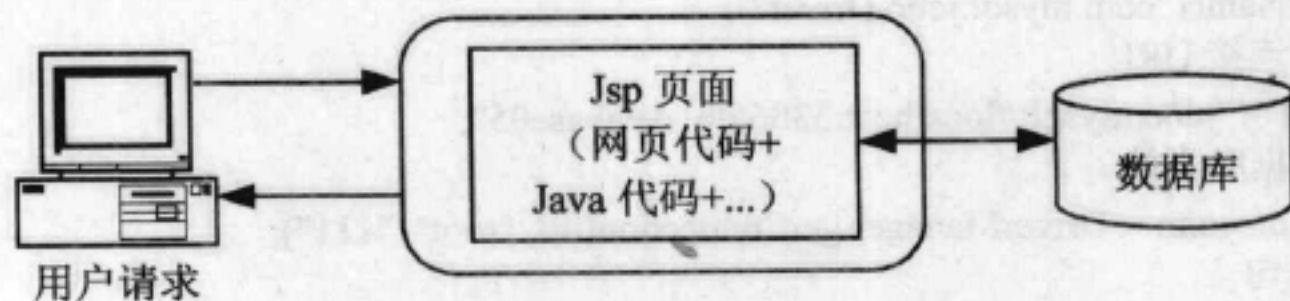


图 5.8 纯 JSP 开发方式

此种开发方式虽然简单，但它也为 Web 程序的开发及应用带来很多不便；在混合交织的代码中，程序的可读性是非常差的，出现了错误不能进行快速调试，给程序的维护与扩展也带来了诸多不便，更谈不上代码重用等。

JavaBean 的产生使 HTML 网页代码与 Java 代码分离开来，在应用 JavaBean 的 JSP 程序中，其业务逻辑变得更加清晰，JSP 页面也显得整洁了很多，如图 5.9 所示。

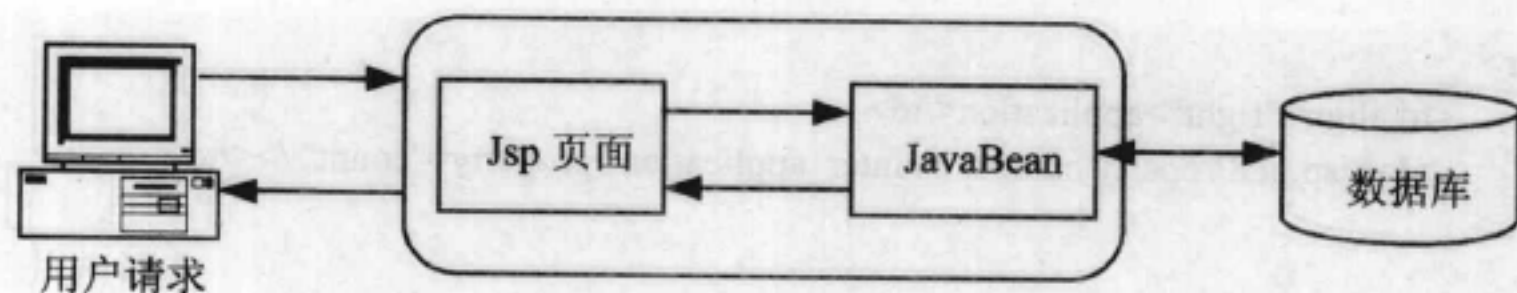


图 5.9 JSP+JavaBean 模式

从图 5.9 可以看出，JSP 页面与 JavaBean 代码相分离。在此种模式中，Web 应用程序的开发开始有了层次概念，JSP 页面用于显示一个视图，JavaBean 用于处理各种业务逻辑，Model1 模式从 JSP 页面之中分离出了业务逻辑层。

例 5.06 Model1 模式录入商品信息。（实例位置：光盘\TM\Instances\5.06）

（1）本实例通过 JSP 与 JavaBean 向数据库中添加商品信息，共涉及到两个 JavaBean 对象，其中 Goods 类用于封装商品对象。其关键代码如下：

```

public class Goods {
    private String name;      //商品名称
    private double price;     //单价
    private String description; //描述信息
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    //省略部分代码
}
  
```

创建名为 GoodsDao 的类，用于封装商品的数据库操作。在此类中编写保存商品信息的方法 saveGoods()，该方法的入口参数为商品对象 Goods。其关键代码如下：

```

public void saveGoods(Goods goods){
    try {
        //加载驱动
        Class.forName("com.mysql.jdbc.Driver");
        //数据库连接 URL
        String url = "jdbc:mysql://localhost:3306/db_database05";
        //获取数据库连接
        Connection conn = DriverManager.getConnection(url, "root", "111");
        //SQL 语句
        String sql = "insert into tb_goods(name,price,description) values(?,?,?)";
        //创建 PreparedStatement 对象
        PreparedStatement ps = conn.prepareStatement(sql);
        //对 SQL 语句中的参数赋值
        ps.setString(1, goods.getName());
        ps.setDouble(2, goods.getPrice());
        ps.setString(3, goods.getDescription());
        ps.executeUpdate(); //更新操作
        ps.close(); //关闭 ps
    }
}
  
```



```

        conn.close();    //关闭 conn
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

(2) 在编写了这两个 JavaBean 对象后, 创建名为 index.jsp 的文件 (程序中的首页文件), 用于提供录入商品信息的表单。其关键代码如下:

```

<form action="service.jsp" method="post" onsubmit="return save(this);">
  <table border="1" align="center" width="300">
    <tr>
      <td align="center" colspan="2">
        <br><h1>录入商品信息</h1>
      </td>
    </tr>
    <tr>
      <td align="right">商品名称: </td>
      <td><input type="text" name="name"></td>
    </tr>
    <tr>
      <td align="right">价 格: </td>
      <td><input type="text" name="price"></td>
    </tr>
    <tr>
      <td align="right">商品描述: </td>
      <td><textarea name="description" cols="30" rows="3"></textarea></td>
    </tr>
    <tr>
      <td align="center" colspan="2">
        <input type="submit" value="提 交">
        <input type="reset" value="重 置">
      </td>
    </tr>
  </table>
</form>

```

(3) 创建 service.jsp 文件, 用于处理表单请求并向数据库中添加数据。其关键代码如下:

```

<body>
  <%
    request.setCharacterEncoding("GBK");
  %>
  <jsp:useBean id="goods" class="com.lyq.Goods"></jsp:useBean>
  <jsp:setProperty name="goods" property="*" />
  <jsp:useBean id="goodsDao" class="com.lyq.GoodsDao"></jsp:useBean>
  <%
    goodsDao.saveGoods(goods);
  %>
</body>

```

注意: `request.setCharacterEncoding("GBK")` 用于设置 request 请求的编码格式, 通过此设置可以解决中文乱码问题。

在 `service.jsp` 页面中对添加商品信息的请求作了处理, 在其中并没有出现 Java 代码与 HTML 代码混合交织的情况, 因为处理业务逻辑的方法由 JavaBean 来完成, 此种开发方式便是 Model1 模式。

提示: 实例运行结果如图 5.10 所示, 正确输入商品信息后, 单击“提交”按钮, 商品信息将被保存到数据库中。



图 5.10 录入商品信息

5.3 Model2 模式

视频讲解: 光盘\TM\Video\5\Model2 模式.exe

与纯 JSP 开发方式相比, Model1 开发模式是一次进步, 但在业务逻辑的控制方面仍然由 JSP 页面充当。针对 Model1 的缺陷, Model2 提出了 MVC 的设计理念, 分别将显示层、控制层、模型层相分离, 使这 3 层结构各负其责, 达到一种理想的设计状态。

5.3.1 MVC 原理

MVC 是一种经典的程序设计理念, 此模式将应用程序分成 3 个部分, 分别为模型层 (Model)、视图层 (View)、控制层 (Controller), MVC 便是这 3 个部分英文字母的缩写, 在 Java Web 开发中其应用如图 5.11 所示。

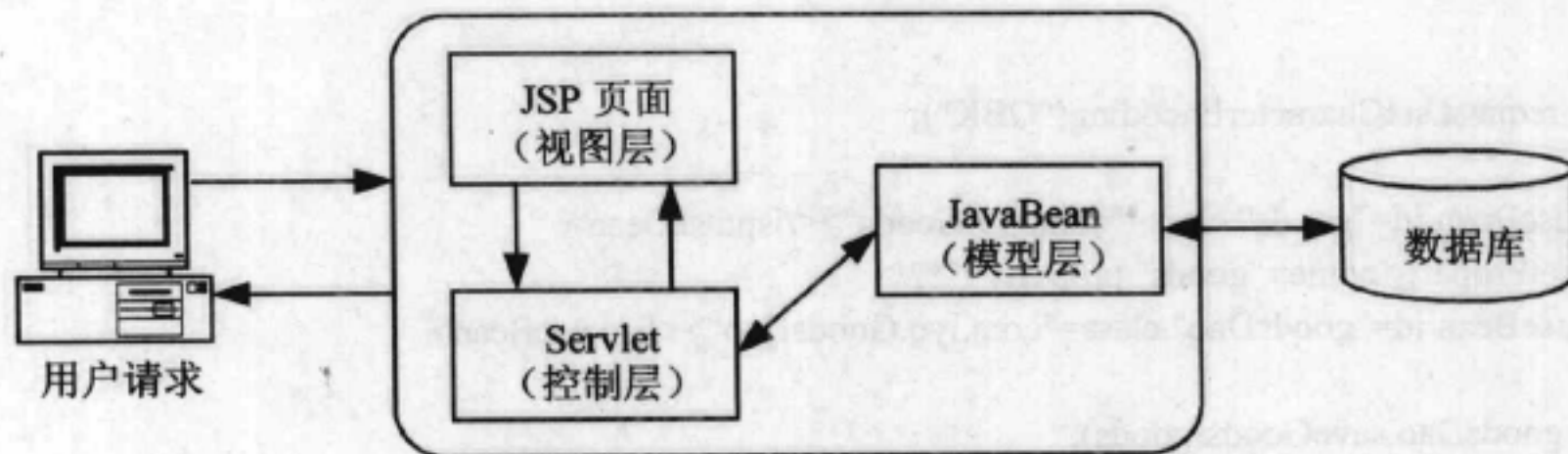


图 5.11 MVC 设计模式