# 第5章 JSP技术

## 5.1 JSP运行与生命周期

在前面的章节中已经介绍了利用 Servlet 进行 Web 应用开发的方法。Servlet 能够非常方便地处理来自客户端的请求，进而使用 session、Cookie 等机制进行相关业务的操作。但对于客户而言，最终还需要通过由 HTML 元素组成的页面展示其响应结果。Servlet 对于 HTML 元素的输出过于低效，为了能更快捷地进行页面的显示，JSP 技术应运而生。

一个典型的 JSP 页面由声明、表达式、程序段、注释、指令、动作以及静态 HTML 标签等部分组成。JSP 底层实现仍然是利用 Servlet。与 Servlet 一样，JSP 文件也存在生命周期的概念。JSP 的生命周期如图 5-1 所示，包括编译、初始化、执行及销毁 4 个阶段。

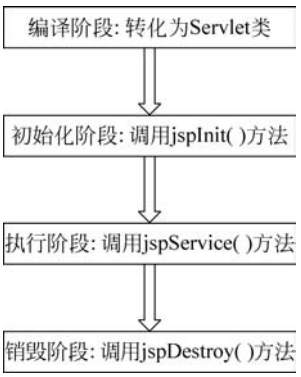


图 5-1 JSP 的生命周期

JSP 文件在生命周期内需要执行如下步骤。

(1) 编译阶段：JSP 在运行时和 Servlet 类似，当客户端通过 HTTP 请求对一个 JSP 页面进行访问时，通过 Connector 监听到请求，转发给 Engine，然后通过 URL 匹配到主机 Host，找到对应 Context 下的 JSP 文件，此时需要查找是否已经存在 JSP 对应的 Servlet 文件，如果该 Servlet 文件存在，那么执行即可；如果不存在，就说明是第一次运行该 JSP 文件，就应将 JSP 文件转换为 Servlet 文件，并进行编译。

(2) 初始化阶段：加载与 JSP 对应的 Servlet 类，创建其实例，并调用它的 jspInit()方法。

(3) 执行阶段：主要调用 jspService(request，response)方法。该方法等价于 Servlet 中的 service()方法，其功能是负责接收用户请求，并对其进行处理，然后将响应返回给客户。

(4) 销毁阶段：当 JSP 页面从容器中移除时，执行 jspDestroy()方法，等价于 Servlet 的 destroy()方法。

## 5.2 JSP 基础语法

### 5.2.1 变量声明与表达式

JSP 嵌入的 Java 语句必须包含在<% %>标签里面。Java 代码可以嵌入 JSP 文件的任意地方。可以使用 JSP 表达式进行字符串变量的输出，在进行表达式输出前，需要对变量进行声明以及初始化。变量的声明语法格式如下：

```
<%!变量声明语句;%>
```

例如：<%! int i = 0; %>或者<% Date e = new Date(); %>。

注意，变量声明语句前面的感叹号!表示该变量的声明范围为整个页面，因此不管使用该变量的语句是在其前面或者后面，都是可以的。如果去掉该感叹号，就不能在变量声明之前使用该变量，否则报错。

表达式的语法格式为<% =变量/表达式%>，用于将声明的变量以字符串的形式输出到客户端。

注意，表达式不能以分号;结尾，否则报错。

下面通过例 5-1 来演示 JSP 中的变量声明以及表达式的使用方法。


视频讲解

**【例 5-1】** JSP 中的变量声明以及表达式的使用方法。

(1) 新建一个 Web 项目 Chapt_05，在 WebContent 目录下新建一个 JSP 文件，命名为 express.jsp，编写代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Insert title here</title>
</head>
<body>
<% =name +"," %>
<%!
String name="小明";
String msg="你好!今天的日期是:";
 %>
<% =msg +(new java.text. SimpleDateFormat ("yyyy-MM-dd HH:mm:ss")).format(new java.
util.Date()) %>
</body>
</html>
```

(2) 右击 express.jsp 文件,选择 Run As→Run on Server,选择 Tomcat 9,部署并启动该项目。然后打开浏览器,输入 http://localhost:8080/Chapt_05/expression.jsp 进行访问。express.jsp 页面输出变量和表达式的结果如图 5-2 所示。

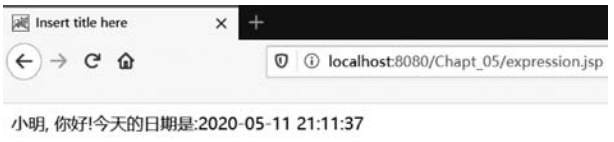


图 5-2 express.jsp 页面输出变量和表达式的结果

可以看到 name 变量的使用语句在声明之前,但是由于声明的时候使用了页面全局变量的设置,因此语句也是可以执行的。

表达式虽然对变量输出很方便,但是表达式不能出现多条语句,并且只能用于字符串类型的变量以及表达式语句的输出。

## 5.2.2 程序段

表达式一般只用于简单的字符串输出,如果想实现较为复杂的功能,表达式显然是不够的。实际上,可以在 JSP 文件中插入多行甚至多段 Java 代码,只需要将这些代码包含在<% %>标签里面即可。在 3.6 节中曾使用 Servlet 生成了一个 10 以内的加法表,下面通过例 5-2 来演示将 Java 程序段嵌入到 JSP 页面中,完成相同的加法表。

**【例 5-2】** JSP 页面中程序段的使用。

在 WebContent 下新建一个 JSP 文件,取名为 additiontable.jsp,代码如下:


视频讲解

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<style type='text/css'>
  table{border-collapse:collapse;}
  td{border:1px solid black;}
</style>
<title>JSP 页面显示加法表格</title>
</head>
<body>
  <table>
    <% for(int i=1;i<=9;i++) {%>
    <tr><%for(int j=1;j<=i;j++){%>
      <td><%=i-j+1%>+<%=j%>=<%=i+1%></td><%}%>
    </tr>
    <%}%>
  </table>
</body>
</html>
```

运行该 JSP 页面,效果如图 3-24 所示。

可以看到,在 JSP 页面中可以灵活地将 Java 程序段、表达式以及 HTML 标签混合使用,Tomcat 针对 JSP 页面中的代码进行如下处理。

(1) <% %>标签里的代码将被认为是 Java 语句而被 Tomcat 服务器进行编译执行。

(2) 没有被<% %>标签包含的代码被认为是 HTML 以及文本元素,直接交由浏览器进行解释执行。

在 JSP 页面编写 Java 代码时,如果要插入非 Java 代码,需要先使用%>标签结束 Java 代码的编写。然后在输入完 HTML 标记或者文本代码后,再继续使用<% %>标签编写 Java 代码。采用 JSP 的方式进行表格的输出,从页面结构来说更加直观,只用关注动态的部分即可,静态不变的标签和样式部分不用像 Servlet 中逐句地通过代码进行输出。

### 5.2.3 JSP 注释

在 JSP 中可以使用两种类型的注释,一种是 HTML 风格的注释,一种是 JSP 类型的注释。

(1) HTML 风格的注释:写法与 HTML 静态页面中的一样,即<! -- 注释内容 -->的形式。注释中可以加入 JSP 表达式,编译并执行以动态生产注释内容。客户端可以接收 HTML 风格注释的内容。

(2) JSP 类型的注释:其又可以分为 Java 代码注释和 JSP 语法注释。

① Java 代码注释:单行使用//注释内容,多行采用/ * 注释内容 * /的形式。

② JSP 语法注释:使用<%-- 注释内容--%>的形式。

JSP 类型的注释不会发送到客户端,在客户端不会看到该类型注释的内容。

下面通过例 5-3 来演示 JSP 中两种注释方式的使用方法。


视频讲解

**【例 5-3】** JSP 中注释的使用方法。

新建一个 JSP 文件,取名为 comment.jsp,在< body >标签体内插入代码如下:

```
<% int a = 1; %>
<!-- 这是一个 HTML 风格的注释,此类注释会送给客户端
     并且可以插入 JSP 表达式,如 a 的值为<% = a %>-->
<% --这是一个 JSP 风格的注释,不会发送到客户端 -- %>
<%
// int b = 2;
//上面为 JSP 中代码的单行注释
/* 下面为 JSP 中代码的多行注释
for(int i = 0;i < 9;i++){
    out.println(i);
}
*/
%>
<% --上面的 JSP 代码注释同样不会发送到客户端 -- %>
```

访问 comment.jsp 页面,此时页面显示为空白内容,右击“查看页面源代码”菜单,弹出 JSP 页面中不同风格的注释显示,如图 5-3 所示。

可以发现,HTML 风格的注释被发送到客户端,并且其中插入的 JSP 表达式也能正确

解析并显示。而 JSP 风格的注释以及 JSP 代码注释都没有发送到客户端。

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE html >
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Insert title here</title>
</head>
<body>

<!-- 这是一个HTML风格的注释，此类注释会送给客户端
        并且可以插入JSP表达式. 如a的值为1-->


</body>
</html>
```

图 5-3 JSP 页面中不同风格的注释显示

# 5.3 JSP 指令与动作

## 5.3.1 JSP 指令

JSP 指令的作用是定义页面如何编译，指令中不包含逻辑控制，也不会在页面中产生任何可见的输出。其语法格式如下：

```
<% @ 指令类别 属性 1 = "属性值 1" 属性 2 = "属性值 2"… 属性 n = "属性值 n" %>
```

常见的 JSP 指令及其功能如表 5-1 所示。

**表 5-1 常见的 JSP 指令及其功能**

| 指　令 | 功　能 |
|---|---|
| page | 用于导入需要的类、指定页面编码方式、输出内容类型、处理错误页面等属性 |
| include | 用于引入其他文件 |
| taglib | 用于引入标签库 |

本节主要介绍 page 以及 include 指令，taglib 指令将在第 8 章介绍 EL 与 JSTL 中进行讲解。

**1. page 指令**

page 指令主要有以下作用。

(1) 导入包。在 express.jsp 中，实例化 Date 以及 SimpleDateFormat 对象时，使用了该对象的全类名，显然这样写比较麻烦。如果想引用 JDK 中的类或者自定义类，可以使用 page 指令的 import 属性来引入，语法格式如下：

```
<% @ page import = "包名 1.类名 1","包名 2.类名 2",…,"包名 n.类名 n" %>
```

也可以每引入一个类，使用一条 import 指令。例如，在 express.jsp 中的开头使用以下语句。

```
<% @ page import = "java.util.Date">
<% @ page import = " java.text.SimpleDateFormat">
```

通过 import 指令引入包中的类以后,在实例化时就可以直接使用类名。如果想引入包中所有的类,和在普通 Java 类中引入时一样,可以使用通配符 * 。

(2) 设定字符集。在 page 指令中可以使用 pageEncoding 属性来指定页面的编码字符集。

注意,此处的 pageEncoding 的值是指定 JSP 文件在编译成 Servlet 源代码时的编码方式。为了保证页面显示不同语言文字的兼容性,建议统一使用 UTF-8 编码方式。因此,可以利用 page 指令进行如下设置。

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

(3) 指定 MIME 类型和字符编码。使用 ContentType 和 charset 指定页面输出的类型以及编码字符集,代码如下:

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

表示该页面的输出为 HTML 格式的文本类型,使用 UTF-8 进行编码。

(4) 设置错误信息提示页面。当后台程序发生错误时,服务器会直接将错误信息输出到页面,此时用户看到了错误信息,影响了使用体验,因此这种方式不太符合程序对鲁棒性的要求。因此比较合适的做法是,当程序出现错误时,在内部跳转到一个指定页面并显示该页面中的提示信息给用户查看。在 JSP 页面中可以使用 errorPage 属性指定发生错误时跳转的页面。提示错误信息的页面使用 isErrorPage=true 来指定。下面通过例 5-4 来演示 page 指令设置错误提示页面的操作步骤。


视频讲解

**【例 5-4】** page 指令设置错误提示页面。

(1) 新建两个 JSP 文件,分别取名为 error.jsp 和 isError.jsp。在 error.jsp 页面的 <body>标签体内编写代码如下:

```
<%
int [] num = {1,2,3,4,5};
for(int i = 0;i < 6;i++){ %>
<% = num[i] %>
<% }
%>
```

上述代码通过 for 循环语句输出数组中的数字,但此时变量 i 的最大数值超出了数组长度而造成了越界。

注意,此时的编译能够通过,但访问该 JSP 页面且在页面运行时,错误直接将出错信息显示出来,如图 5-4 所示。

(2) 在 error.jsp 中,通过<%@ page errorPage="isError.jsp"%>指令将 isError.jsp 设置为发生错误时进行提示的页面。

(3) 在 isError.jsp 页面中,通过 page 指令设置 isErrorPage 属性为 true,并在<body>标签内进行信息提示,代码如下:


图 5-4 访问 JSP 页面且在页面运行时，错误直接将出错信息显示出来

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" isErrorPage="true" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Insert title here</title>
</head>
<body>
页面出现错误,请返回
</body>
</html>
```

（4）此时再次运行 error.jsp 页面，发现页面不再直接输出错误信息，而是通过 isError.jsp 页面显示提示信息，如图 5-5 所示。

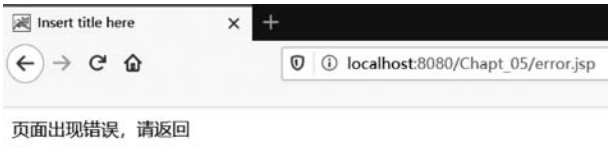


图 5-5 通过 isError.jsp 页面显示提示信息

### 2. include 指令

include 指令的作用是引入外部文件，可以是 JSP、HTML、Java 程序以及其他脚本程序。在实际开发中，很多页面会有很多部分是类似的，比如位于页面顶部的导航栏，以及页面底部的显示一些基本网站信息的部分。为了复用这些信息，可以将这两部分抽离出来，形成单独的文件，然后在页面上使用 include 指令引入头部和尾部文件。其语法格式如下所示。

```
<%@ include file="文件名" %>
```

下面通过例 5-5 来演示 include 指令的使用方法。

**【例 5-5】** include 指令的使用方法。

(1) 新建 3 个 JSP 文件,分别取名为 header.jsp、footer.jsp 和 includefile.jsp。其中 header.jsp 的代码如下:


视频讲解

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

footer.jsp 的代码如下:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
     pageEncoding="UTF-8"%>
 <hr>
<p>Copyright XXXX, All Rights Reserved 联系电话:4000000000</p>
```

在 includefile.jsp 的<body>标签内添加代码如下:

```
<%@ include file="header.jsp" %>
这是主体部分
<%@ include file="footer.jsp" %>
```

(2) 访问 includefile.jsp 页面,include 指令引入外部文件并显示,如图 5-6 所示。


图 5-6 include 指令引入外部文件并显示

注意,外部引入的文件里面如果定义了变量,就不要和主页面中定义的变量名一致;否则会出现错误。因为 include 指令相当于把被引入文件直接插入本页面,然后整个页面代码再进行编译,相当于重复定义变量。因此,要避免这种情况的出现。


视频讲解

## 5.3.2 JSP 动作

JSP 动作是使用 XML 标记语法来控制 JSP 页面的行为。语法如下:

```
<jsp:动作名 属性名1="属性值1" 属性值2="属性值2" …属性名n="属性值n"/>
```

或者使用如下形式：

```
<jsp: 动作名 属性名1="属性值1" 属性值2="属性值2" …属性名n="属性值n">
……
</jsp>
```

JSP 动作的语法及其功能如表 5-2 所示。

**表 5-2 JSP 动作的语法及其功能**

| 语　法 | 描　述 |
|---|---|
| jsp:include | 类似于 include 指令，用于引入文件 |
| jsp:forward | 请求转发到另外的页面 |
| jsp:useBean | 寻找一个 JavaBean |
| jsp:setProperty | 设置 JavaBean 属性 |
| jsp:getProperty | 获取 JavaBean 属性 |
| jsp:plugin | 根据浏览器类型为 Java 插件生成 OBJECT 或 EMBED 标记 |

本节主要介绍 include 以及 forward 两个动作，关于 JavaBean 的指令将在第 6 章介绍，此外 jsp:element、jsp:attribute、jsp:body、jsp:text 等动作使用较少，感兴趣的读者可以自行查阅资料。

### 1. include 动作

include 动作的基本语法如下：

```
<jsp:include page="文件名"/>
```

include 动作也是用于外部文件的被引入，与 include 指令的区别如下所述。

(1) include 动作会将被包含文件先编译，然后将输出包含到主页面中，因此不会因为相同变量的定义导致报错的问题。

(2) include 动作会自动检查被包含页面的内容的变化，并实时更新。include 指令则不会实时检测。

### 2. forward 动作

forward 动作用于实现页面的跳转，语法如下所示。

```
<jsp:forward page="文件名"/>
```

注意，跳转页面必须是服务器内部资源，不能是外部链接。该动作等价于 Servlet 的 request 对象的请求转发。执行该动作后，当前页面将不再执行，而直接跳到指定的页面。例如，新建一个 JSP 文件，命名为 forward.jsp，在 body 标签体内添加代码如下：

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

运行 forward.jsp，页面直接跳转到 additiontable.jsp 页面并显示加法表格。

## 5.4 JSP 内置对象

所谓内置对象,是指 JSP 页面生成后,通过 Web 容器来实现和管理,自动载入不需要实例化就可以直接使用的对象。通过内置对象的使用,可以提高开发效率。事实上,由于 JSP 是对 Servlet 进行封装,所以 JSP 的内置对象和 Servlet 中使用的对象很相似。

JSP 共有以下 9 个内置对象。

(1) out 对象:负责对客户端的输出。其等同于在 Servlet 中通过 response.getWriter()方法获取的客户端输出对象,主要的方法和 Servlet 中使用的类似。

(2) request 对象:负责得到客户端的请求信息。其等同于 Servlet 中相关方法(一般为 doGet 或者 doPost)中的 request 参数,主要的方法和 Servlet 中使用的类似。

(3) response 对象:负责向客户端返回响应。其等同于 Servlet 中 service()方法中的 response 参数,主要的方法和 Servlet 中使用的类似。

(4) session 对象:负责保存客户端在一次会话过程中的信息。其等同于 Servlet 中使用 request.getSession()方法获取的 HttpSession 对象,主要的方法和 Servlet 中使用的类似。

(5) application 对象:表示整个应用的环境信息。其等同于 Servlet 中使用 request.getServletContext()方法获取的 ServletContext 对象,主要的方法和 Servlet 中使用的类似。

(6) exception 对象:表示页面上发生的异常,该对象对应 java.lang.Exception 接口,可以获得页面的异常信息。可以在 isError.jsp 中添加如下代码,获取该页面中出现的错误信息以及信息的详细描述。

```
<% = exception.getMessage() %><br>
<% = exception.toString() %><br>
```

(7) page 对象:该对象的对应类型是 java.lang.Object,表示 JSP 页面本身,可以使用 Object 对象的方法。在页面中可以用 this 替代,一般较少使用该对象。

(8) pageContext 对象:该对象的对应类型是 javax.servlet.jsp.PageContext,表示 JSP 的上下文。通过该对象,可以获取除自身外的其他 8 个对象,也可以直接访问绑定在 page、request、session 以及 application 对象上的 Java 对象。例如,假设 request 对象通过 setAttribute()方法设置了一个 username 对象,则可以使用以下语句直接获取该 username 对象。

```
<% = pageContext.request.username %>
```

(9) config 对象:该对象的对应类型是 javax.servlet.jsp.ServletConfig,表示 JSP 初始化时所需要的参数以及服务器的相关信息,等同于 Servlet 中使用 request.getServletConfig()方法获取的 ServletConfig 对象。这在实际开发中使用较少。

上述 9 大对象中,使用较多的是前面 5 个,其用法在 Servlet 中已经做过介绍。此外,在 JSP 中如果想处理 Cookie,与 Servlet 中的一样,可通过 response 对象向客户端写入 Cookie 对象,利用 request 对象读取 Cookie 对象的数组。

## 5.5 JSP 与 Servlet 共同开发

在前面的章节中已经讨论过 JSP 与 Servlet 之间的关系以及各自的优势，下面通过例 5-6 来演示 JSP 与 Servlet 是如何共同进行开发的。


视频讲解

### 5.5.1 需求分析

**【例 5-6】** JSP 与 Servelt 共同开发。

该例子主要实现如下的功能。

(1) 用户在登录页面输入用户名和密码，如果用户名和密码相等，就进入欢迎页面。在欢迎页面显示有用户名和一个安全退出超链接。在不关闭浏览器的情况下，页面进行回退后，若再次访问欢迎页面，则仍能显示用户信息。

(2) 如果不是单击安全退出超链接退出，而是直接关闭浏览器，那么当再次访问登录页面时，用户名和密码信息就会自动输入，不用用户手动填写，单击登录按钮即可直接登录。

(3) 如果是单击安全退出超链接退出，那么在返回到登录页面时，需要重新填写用户的登录信息。

(4) 若在没有登录成功的情况下直接访问欢迎页面，则直接跳转到登录页面。

### 5.5.2 实现思路

该例子需要结合 session 和 Cookie 机制，大致的思路如下所述。

(1) 在登录页面需要首先判断是否存在键名为 userinfo 的 Cookie 对象，如果存在，就将对应的键值字符串以“|”符号进行分离。将分离后的两个子字符串分别赋给表单中用户名和密码的值；如果不存在 Cookie 对象，就按照正常登录处理。

(2) 单击登录后提交给 Servlet 进行处理，该 Servlet 在处理完用户名和密码的校验后，新建一个键名为 userinfo 的 Cookie 对象，键值设置为 username|password，其中 username 和 password 为用户提交的用户名和密码的字符串信息。然后向客户端写入该 Cookie，以及向 HttpSession 对象中存储用户名信息，并跳转到欢迎页面。

(3) 欢迎页面需要判断 session 对象是否存在，并获取 username 的值。

(4) 安全退出由负责退出的 Servlet 处理，该 Servlet 需要将 Httpsession 对象销毁，并且删除 Cookie 信息。

### 5.5.3 代码实现

下面按照上述思路进行代码的编写。

(1) 首先新建一个 JSP 页面，取名为 login.jsp，在< body >标签内编写代码如下：

```
<%
String username = "";
String password = "";
Cookie[] cookies = request.getCookies();
```

```
if(cookies!= null) {
String userinfo = null;
for(int i = 0;i < cookies.length;i++) {                    //循环遍历 Cookie
if(cookies[i].getName().equals("userinfo")) {              //找到 Cookie 对应的键名
userinfo = cookies[i].getValue();                          //获取 Cookie 的键值
break;
}
}
if(userinfo!= null){
//Cookie 在写入时用户名和密码以“|”作为分隔
String[] information = userinfo.split("\\|");
username = information[0];
password = information[1];
}
}
%>
<form action = "LoginServlet" method = "post">
用户名:<input type = "text" name = "username" value = <% = username %>>
<br>
密    码:
<input type = "password" name = "password" value = <% = username %>><br>
<input type = "submit" value = "登录" ><input type = "reset" value = "重置" >
</form>
```

在获取到 userinfo 的 Cookie 后,以“|”为分隔符,分别取出 Cookie 中存储的 username 和 password 的值,通过 JSP 表达式赋给填写用户名文本框和密码框元素的 value 值,从而达到自动填入的功能。

(2) 新建一个 Servlet,取名为 LoginServlet,通过注解@WebServlet("/LoginServlet")设置其映射路径为/LoginServlet,在 doGet()方法中编写代码如下:

```
request.setCharacterEncoding("UTF-8");
response.setCharacterEncoding("UTF-8");
response.setHeader("Content-type","text/plain;charset = UTF-8") ;
String username = request.getParameter("username");
String password = request.getParameter("password");
if(username!= null&&password!= null) {                     //表单提交内容不为空
if(username.equals(password)) {                            //验证用户名密码
HttpSession session = request.getSession();                //创建一个 session 对象
session.setAttribute("username", username);                //将用户名存储在 session 中
//创建一个键名为 userinfo 的 Cookie 对象,键值为 username|password
Cookie userinfocookie = new Cookie("userinfo",username + "|" + password);
userinfocookie.setMaxAge(60 * 5);                          //Cookie 有效值为 5 分钟
response.addCookie(userinfocookie);                        //将用户名的 Cookie 发送到客户端
response.sendRedirect("welcome.jsp");                      //跳转到欢迎页面
}else {
response.getWriter().append("用户名密码错误,请重新登录,
5 秒后回到登录页面……");
response.setHeader("Refresh", "5;URL = login.jsp");
```

```
}
}else {//防止未经表单提交,直接访问该 Servlet
response.getWriter().append("禁止直接访问,5 秒后回到登录页面……");
response.setHeader("Refresh", "5;URL=login.jsp");
}
```

在 LoginServlet 中,对用户名和密码进行校验后,创建 HttpSession 对象存储用户名,并创建 Cookie 对象,其值是 username 和 password 使用"|"连接而成。

(3) 创建一个 JSP 文件,取名为 welcome.jsp,在 body 标签体内编写代码如下:

```
<%
//以下三条语句是为了使页面不使用缓存
response.setHeader("Pragma","No-Cache");
response.setHeader("Cache-Control","No-Cache");
response.setDateHeader("Expires", 0);
if (session != null) {                          // 如果 session 存在,说明不是直接访问
// 从 session 中读取用户名
String username = (String) session.getAttribute("username");
if (username != null)                           // 进一步判断会话中是否存在用户名
{out.print("欢迎您," + username + "<a href='LogoutServlet'>
安全退出</a>");
}
else {
out.print("你还没有登录,5 秒后跳转到登录页面……");
response.setHeader("Refresh", "5;URL=login.jsp");
}
} else {                                        // 如果 session 不存在,说明是直接访问该页面
out.print("禁止直接访问,5 秒后跳转到登录页面……");
response.setHeader("Refresh", "5;URL=login.jsp");
}
%>
```

该页面通过内置 session 对象获取用户名信息。另外开头的三条语句是为了防止页面使用缓存和显示过期的信息。

(4) 新建一个 Servlet,取名为 LogoutServlet,通过注解@WebServlet("/LogoutServlet")设置其映射路径为/LogoutServlet,在 doGet()方法中编写代码如下:

```
response.setCharacterEncoding("UTF-8");
response.setHeader("Content-type","text/plain;charset=UTF-8") ;
//判断当前请求是否存在对象
HttpSession session = request.getSession(false);
if(session!=null) {                             //如果会话存在,去除会话中的登录状态
session.invalidate();                           //销毁整个会话
Cookie usernameinfo=new Cookie("userinfo","");
usernameinfo.setMaxAge(0);                      //设置 Cookie 有效时间为 0,即立即删除
//发送到客户端覆盖之前 Cookie 的设置
response.addCookie(usernameinfo);
response.getWriter().append("注销成功,5 秒后跳转到登录页面……");
```

```
    response.setHeader("Refresh", "5;URL=login.jsp");
    }else {//会话不存在,说明是直接访问该页面
    response.getWriter().append("你还未登录,无须注销,
    5秒后跳转到登录页面……");
    response.setHeader("Refresh", "5;URL=login.jsp");
    }
```

在页面中单击安全退出超链接后,除了销毁会话外,还应该去除之前存储在客户端的Cookie,达到安全退出的目的。

以上就是该例子的代码实现,读者可以在编写完毕后运行相关页面,验证是否实现相关功能。

在本例中,Servlet更多的是进行业务逻辑的处理,即用户的登录、退出以及针对HttpSession和Cookie的设置。而JSP页面更多的是根据Servlet处理后的结果,进行相关信息的显示。这也正是二者各自的优势所在,后续章节的讲解和示例基本也是遵循这样的分工。