第5章

JSP基础

JSP是一种动态网页开发技术。本章主要介绍了JSP基本语法、内置对象、Servlet、应用监听器、与数据库交互等内容。熟悉JSP,有助于建立B/S模式下前后端交互处理的初步思维,为后续章节的学习做好准备。

5.1 JSP概述

5.1.1 JSP简介

JSP(Java Server Page,Java服务页面)是由原美国Sun(Sun Microsystems)公司倡导、很多公司(例如Oracle、IBM等)参与建立、基于Java语言的动态Web网页开发技术。

由于JSP是基于Java语言的,先天就具有多平台支持特性,在实践中仍然具有一定市场,例如中国工商银行、华夏基金等官网就是用JSP编写的。

5.1.2 JSP基本页面结构

JSP文件的扩展名是.jsp,基本页面结构如下:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
      <title>Title</title>
</head>
<body>
<%
      out.print("Hello,JSP!");
 %>
</body>
</html>
```

整个页面可以概要地划分成三大部分：

（1）JSP 指令部分。

第 1 行代码，属于 JSP 的页面指令。该指令一般放在第 1 行，指明了页面内容是纯文本的 HTML，字符集编码为 UTF-8，使用的语言是 Java。

（2）JSP 代码部分。

粗体部分代码，就是 JSP 代码，用于执行各种 JSP 操作。

（3）HTML 代码部分。

除了第 1 行代码、粗体部分代码，其他都属于 HTML 代码内容。当然，也可以加入前面学习过的 CSS 和 JS 等内容。

5.1.3 配置 Tomcat 依赖

为了后续示例方便，先创建好一个 Maven Webapp 项目 chapter5（若忘记创建方法，请回看第 1 章内容）。

由于我们使用 Tomcat 作为 JSP 的应用服务器，因此需要在项目中加入 Tomcat 依赖。打开项目 pom.xml 文件，输入 Tomcat 依赖代码：

```
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-catalina</artifactId>
    <version>9.0.50</version>
</dependency>
```

注意添加的位置！单击图 5-1 中画圈处的 Load Maven Changes 按钮，重新加载。

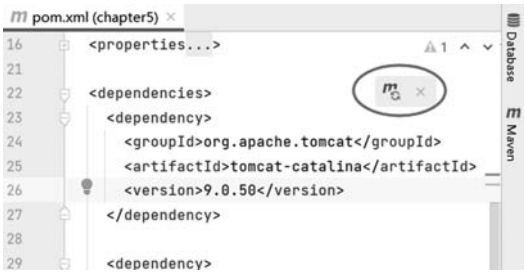

图 5-1 添加 Tomcat 依赖

5.2 JSP 基本语法

5.2.1 程序段

这里所说的程序段，是指包含变量、方法、表达式等被<%和%>括起来的部分，不包含前面 5.1.2 节中所说的 JSP 指令。如果要在项目中创建 test.jsp，代码如下：

```
<body>
<%
     int sum = 0;
     for (int i = 0; i <= 100; i++)
          sum += i;
 %>
<b>计算结果:</b><% = sum %>
</body>
```

这段代码中,JSP 程序段和 HTML 代码混在一起。启动 Tomcat,在浏览器中打开页面:http://localhost:8080/chapter5/test.jsp,将在浏览器中输出“**计算结果:5050**”。

5.2.2 表达式

JSP 表达式可以将数据转换成字符串,直接输出到页面,其语法格式如下:

```
<% = 表达式 %>
```

注意:中间不能有任何空格和分号。

示例:使用表达式动态输出页面链接。

```
<body>
<%
     String[] url = new String[3];
     url[0] = "https://www.hust.edu.cn";
     url[1] = "https://www.whu.edu.cn/";
     url[2] = "index.jsp";
 %>
<a href = "<% = url[0] %>" target = "_blank">华中科技大学</a>
<a href = "<% = url[1] %>" target = "_blank">武汉大学</a>
<a href = "<% = url[2] %>" target = "_self">首页</a>
</body>
```

5.2.3 JSP 中的注释

注释是对程序的说明性文字,为程序的可读性及日后系统的维护带来极大方便。JSP 注释不会执行,也不会发送给用户,即用户是看不到的。

```
格式 1: <% -- 注释内容 -- %>
格式 2: <% //注释内容 %>
```

下面是一个完整的 JSP 页面,包含了两种格式的注释。加粗字体是格式 1 注释,有下画线的是格式 2 注释:

```
<% --
编写者：杨过
说明：示例程序
 -- %>
<%@ page contentType = "text/html;charset = UTF - 8" language = "java" %>
<% -- 导入日期处理相关的类 -- %>
<%@ page import = "java.text.SimpleDateFormat" %>
<%@ page import = "java.util.Date" %>
<html><head>
    <title>注释示例</title>
</head>
<body>
<% -- 显示图片、格式化的日期 -- %>
<%
    out.println("<img src = 'images/m0.png'>");     //显示图片
    Date today = new Date();
    //格式化
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy - MM - dd HH:mm:ss");
    //输出到页面
    out.print("<h2>" + sdf.format(today) + "</h2>");
 %>
</body></html>
```

通过注释，即使不熟悉相关代码的人，也可以很快理解代码的含义，提高了可读性。

5.3 JSP 内置对象

内置对象是隐含对象，无须声明和创建，可以直接使用。

5.3.1 out

out 对象主要用于向客户端发送数据，在前面其实已经接触到。out 对象提供了很多方法，这里介绍其中的 2 个常用方法。

◇ print()/println()：输出数据/输出数据后换行。

◇ close()：关闭输出流。

示例：页面停顿 3s 后返回主页。

```
<body>
<%
    out.print("<span style = 'color:#f00'>请稍候，3 秒后返回到主页.....</span>");
    out.print("<script>");                                                    ①
    out.print("setTimeout(\"location.href = 'index.jsp'\",3000);");          ②
    out.print("</script>");
    out.close();
    out.println("<b>欢迎再回来!</b>");                                        ③
```

```
%>
</body>
```

从上面代码可以体会到 out 对象的强大输出功能：不但可以输出普通文字，还可以输出 HTML 的各种标签，例如< span >，甚至可以输出 JS 脚本代码！

① 输出< script >脚本声明。

② 输出 JS 代码的 setTimeout()函数，停顿 3s 后，通过 location. href 将页面转向 index. jsp。

③ 这句欢迎再回来的文字，不会在页面显示，因为前面的 out. close()已经关闭输出了。

提示：尽管可以在 JSP 代码中用 out 输出 JS，但不建议这样做，这会导致代码的可读性变差，日后的维护也会极其困难。限于我们目前掌握的知识有限，目的只是演示一下 out 的强大输出能力而已。

5.3.2 request

请求对象 request 可获取通过 HTTP 传送到客户端的数据。下面是 request 的常用方法。

◇ getRequestURI()：获得所请求的 URL 地址。

◇ getServerName()：获得服务器名称。

◇ getServerPort()：获得服务器提供的 HTTP 服务的端口号。

◇ getRemoteAddr()：获得 IP 地址。

◇ setCharacterEncoding("编码类型")：设置请求的编码类型。

◇ getRemoteHost()：获得主机名，一般为 IP 地址。

◇ setAttribute(String s,Object o)：创建一个新属性并赋值为 o。

◇ getAttribute(String s)：获取指定属性名的值。

◇ getParameter(String name)：获得客户端传给服务器端的参数值。

◇ getRequestDispatcher(String var)：将请求转发到由 var 指定的页面。

◇ request. getParameterMap()：获得客户端请求数据的键值对。

示例 1：在主页 index. jsp 中打印相关信息。

```
<body>
<%
     String uri = request.getRequestURI();
     String serverName = request.getServerName();
     int port = request.getServerPort();
     String ip = request.getRemoteAddr();
     String host = request.getRemoteHost();
     out.print(uri + " " + serverName + " " + port + " " + ip + " " + host);
%>
</body>
```

页面上将显示如下内容：

```
/chapter5/index.jsp localhost 8080 127.0.0.1 127.0.0.1
```

示例 2：制作包含用户名、密码的表单页面 login.jsp，提交给 doLogin.jsp 后显示用户输入的用户名、密码。

先看 login.jsp 的主要代码：

```
<body>
<form name="form1" method="post" action="doLogin.jsp">
    用户名<input type="text" name="usr">
    密 码<input type="password" name="pwd">
    <button>登 录</button>
</form>
</body>
```

大家对代码应该比较熟悉：通过表单，以 post 模式提交用户名、密码数据给 doLogin.jsp。再看 doLogin.jsp 的代码：

```
<body>
<%
    request.setCharacterEncoding("utf-8");          //必须,否则用户名是中文时,会输出乱码
    String username = request.getParameter("usr");  //接收客户端传送的用户名
    String password = request.getParameter("pwd");  //接收客户端传送的密码
    out.print("你输入的用户名:" + username + " 密码:" + password);
%>
</body>
```

这个例子，显示了数据在不同页面间传送，以及服务端进行接收的基本方法。

5.3.3 response

响应对象 response 将数据从服务器端发送到客户端，以响应客户端的请求。下面是 response 的几个常用方法。

◇ addHeader(String name, String value)：添加指定名称和值的 HTTP 文件头信息。

◇ setHeader(String name, String value)：设置指定名称和值的 HTTP 文件头信息。

◇ sendRedirect(String url)：重定向到 url 地址，即打开 url 所代表的网页。

◇ sendError(int code)：发送错误信息编码，例如常见的“404”表示网页找不到。

示例 1：将 JSP 中的古诗自动下载为 Word 文档，效果如图 5-2 所示。

图 5-2 响应为 Word 文档

由于下载为 Word 文档，所以需要设置 HTTP 头，指示浏览器，将文件内容作为 Word 文档下载。首先，需要设置 JSP 响应内容为“application/msword; charset=GB18030”，其中 GB18030 在第 1 章接触过。然后，还需

要设置 HTTP 的 Content-Disposition 响应头，指示浏览器页面内容的展示形式：inline(内联形式，即作为网页的一部分)、attachment(附件形式，下载并保存到本地)。显然，我们需要设置为附件形式，并指定文件名为 poem.doc：

```
response.setHeader("Content-Disposition", "attachment;filename=poem.doc");
```

新建文件 mypoem.jsp，主要代码如下：

```
<body>
<%
     response.setContentType("application/msword;charset=GB18030");
     response.setHeader("Content-Disposition", "attachment;filename=poem.doc");
     out.print("<b>芙蓉楼送辛渐</b><br/>");
     out.print("【唐】王昌龄<br/>");
     out.print("寒雨连江夜入吴,<br/>");
     out.print("平明送客楚山孤.<br/>");
     out.print("洛阳亲友如相问,<br/>");
     out.print("一片冰心在玉壶.");
     out.close();
%>
</body>
```

在浏览器地址栏输入 http://localhost:8080/chapter5/mypoem.jsp，将自动下载为 Word 文件 poem.doc。

示例 2：根据登录用户名、密码的不同，打开不同页面。修改上 5.3.2 节中的 doLogin.jsp 文件：当用户名为 admin、密码为 007 时，打开 admin.jsp 页面，否则显示“用户名或密码错误，登录失败!”。

只需要修改 doLogin.jsp 的代码如下：

```
<%
    request.setCharacterEncoding("utf-8");
    String username = request.getParameter("usr");
    String password = request.getParameter("pwd");
    if (username.equals("admin") && password.equals("007"))
          response.sendRedirect("admin.jsp");
    else
          out.print("用户名或密码错误,登录失败!");
%>
```

5.3.4 session

会话 session 代表服务器与客户端所建立的状态信息。由于 HTTP 协议是无状态协议，即服务器并不知道客户端的状态，例如用户登录后，直接关闭了浏览器，这时候服务器并不知道。另外，购物时，当在 A 页面购买一双鞋子，再到 B 页面去购买袜子，这时候因为 HTTP 无状态性质，无法知

道 A 页面购买了什么，因此需要 session 来保存客户端状态信息。每个打开网站的用户，都有自己私有的 session。下面是 session 的几个常用方法。

◇ setAttribute(String s,Object o)：设置 session 属性名为指定值。

◇ getAttribute(String s)：获取指定属性名的 session 值。

◇ removeAttribute(String s)：删除指定属性名的 session。

◇ invalidate()：使 session 失效。

◇ isNew()：是否为新的 session。

◇ setMaxInactiveInterval(int time)：设置 session 的有效期限，单位为秒。

示例：在主页 index.jsp 中放置三个链接：用户登录，链接到 login.jsp；用户注销，链接到 logout.jsp；文件下载，链接到 download.jsp，登录成功的用户才可以打开该链接，否则弹框提示需要先登录。

我们用 session 在不同页面之间分享信息，下面是主要代码。

1. index.jsp

```
<body>
<a href="login.jsp">用户登录</a>
<a href="logout.jsp">用户注销</a>
<%
      Object loginer = session.getAttribute("loginer");       //loginer 用于记录用户登录状态
      if (loginer == null || !loginer.equals("sucess"))       //success 表示登录成功状态
            out.print("<a href='#' onclick=alert('请先登录!')>文件下载</a>");
      else
            out.print("<a href='download.jsp')>文件下载</a>");
      %>
</body>
```

大家应该已经注意到，文件下载链接是根据 session 的值动态生成的。

2. login.jsp

与前面 5.3.2 节中的一样。

3. doLogin.jsp

```
<%
      request.setCharacterEncoding("utf-8");
      String username = request.getParameter("usr");
      String password = request.getParameter("pwd");
      out.print("<script>");
      if (username.equals("admin") && password.equals("007")) {
            session.setAttribute("loginer", "sucess");     //用 session 设置用户登录状态
            out.print("alert('登录成功!');");
      } else
            out.print("alert('用户名或密码错误,登录失败!');");
      out.print("location.href='index.jsp'");              //转向打开主页
      out.print("</script>");
%>
```

4. logout.jsp

```
<%
      session.invalidate();                        //令 session 失效
      response.sendRedirect("index.jsp");          //转向主页
%>
```

5.3.5 application

服务器启动后,会自动创建 application 对象。与 session 属于每个用户独有不同,application 存放公共数据,网站的所有用户都可存取其数据,也就是说 application 是网站所有用户共享的。下面是 application 的三个常用方法。

◇ setAttribute(String s,Object o):设置指定名称、值的 application 对象。

◇ getAttribute(String s):获取指定属性名的 application 值。

◇ removeAttribute(String s):删除指定属性名的 application。

示例:用 application 实现网站访问量,如图 5-3 所示。

图 5-3 访问量计数器

这个计数器的每位数字,用< span >显示,并用 CSS 定义了一个样式 numSpan。JSP 代码如下:

```
<%
    if (application.getAttribute("counter") == null) {                    ①
          application.setAttribute("counter", "6357438");
    }
    String counter = application.getAttribute("counter").toString();      ②
    int i = Integer.parseInt(counter);
    i++;
    application.setAttribute("counter", i);                               ③
    out.print("您是第");
    for (int j = 0; j < counter.length(); j++) {                          ④
          char c = counter.charAt(j);
          out.println("< span class = 'numSpan'>" + c + "</span >");
    }
    out.print("位访问者!");
%>
```

说明:

① 服务器启动后,首次访问主页时,counter 并不存在,所以设置并赋初值。这里为了演示效果,设置了一个比较大的数。

② 获取当前 counter 的值,并转换为字符串。注意 application.getAttribute("counter")返回的是 Object 对象。

③ 访问量加 1 后,要记住这个新的值,所以给 counter 重新赋以新值。

④ 遍历总访问量字符串中的每位数字,用 CSS 修饰过的< span >显示在页面上。

样式 numSpan 代码如下：

```
<style>
    .numSpan {
        border-radius: 50%;
        text-align: center;
        display: inline-block;
        color: #fff;
        background-color: #f15555;
        width: 1.8em;
        height: 1.7em;
        line-height: 1.7em;
        font-size: 0.6em;
    }
</style>
```

这个计数器其实有两个问题：

(1) 一旦服务器停止运行，再次启动时计数器将还原为初值；

(2) 每刷新一次页面，计数器将加 1。第一个问题，可以用以后的数据库知识解决：将访问量存放到数据库里面，从数据库里面读取，这样一来服务器的启停就毫无影响了。至于第二个问题，用本章前面所学知识，即可完美解决。作为练习，请大家思考完成。

5.4 使用 Servlet

5.4.1 Servlet 简介

Servlet 是基于多线程技术、运行于服务端的 Java 类。由于 Servlet 是编译好的类，运行速度比 JSP 文件要快，因为 JSP 文件最终都要被 Tomcat 服务器编译成 Servlet 再运行的。这个动态编译的过程，是需要消耗时间的。

在浏览器中打开站点主页 index.jsp，实际上 Tomcat 是这样处理的：先将 index.jsp 的内容自动生成名为 index_jsp.java 的 Servlet 文件，然后再编译成 index_jsp.class，最终运行这个 index_jsp.class。

如果将 chapter5 项目发布到 Tomcat 的 webapps 下，启动站点后打开目录 D:\apache-tomcat-9.0.50\work\Catalina\localhost\chapter5\org\apache\jsp，就可以看到 index_jsp.java 和 index_jsp.class 这两个文件。

5.4.2 Servlet 生命周期

Servlet 从被服务器创建到销毁的过程称为生命周期。整个生命周期，如图 5-4 所示。

对读者而言，日常应用会更多地关注于 Servlet 的响应处理以及输出响应信息的过程。这会涉及 Servlet 的两个重要方法。

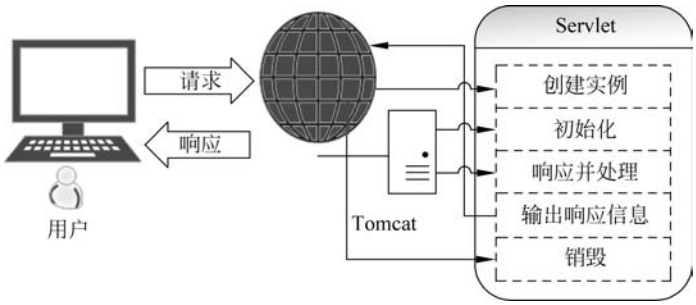

图 5-4 Servlet 生命周期

5.4.3 doGet()和 doPost()方法

doGet()方法用来处理客户端的 get 方式请求,而 doPost()自然是处理来自客户端的 post 请求。关于 get、post 这两种方式的区别,请参阅第 2 章 2.4.6 节的内容。这两个方法,都是 HttpServlet 抽象类的内部方法,需要我们覆盖这两个方法。形式如下:

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    //在这里完成我们的各种任务
    }
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
          doGet(request, response);
}
```

读者可能注意到:在 doPost()方法里面调用了 doGet()方法!这是一种增强 Servlet 适应性的技巧。这样一来,我们的任务代码,只需要写在 doGet()方法里面。无论用户采用 get 还是 post 方式提交数据,无须修改代码,Servlet 程序都能适应。

5.4.4 加入 Servlet 依赖

要方便创建 Servlet,需要加入 Servlet 依赖。打开 pom.xml,加入以下代码并更新 Maven:

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>
```

5.4.5 创建 Servlet

我们需要创建一个包 servlets，用来存放 Servlet 文件。先来创建 java 文件夹，这个文件夹是必需的，以后创建其他各种包，都必须放在 java 文件夹下。创建方法：在项目的 src→main 上右击，再选择 New→Directory，在弹出的对话框中选择 java 即可，如图 5-5 所示。

图 5-5 新建 java 文件夹

现在，在 java 文件夹上右击鼠标，再选择 New→Package，完成 servlets 包的创建。

示例 1：创建名为 GoHome 的 Servlet，在页面上显示“请稍候，3s 后返回主页……”。过了 3s，自动打开主页 index.jsp。

（1）在 servlets 包上右击鼠标，再选择 New→Servlet，如图 5-6 所示。

（2）弹出 New Servlet 对话框。在 Name 框中输入 GoHome，再单击 OK 按钮完成创建，如图 5-7 所示。

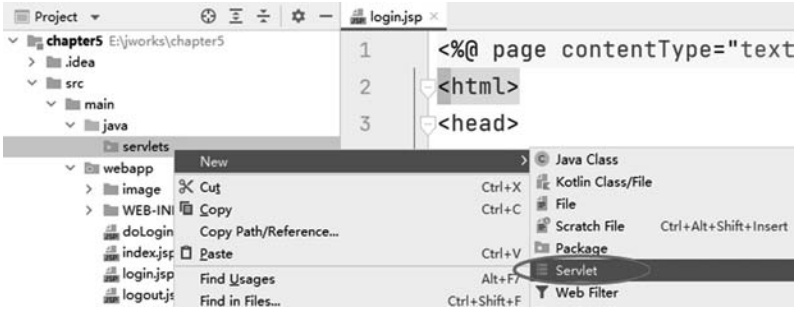

图 5-6 新建 Servlet

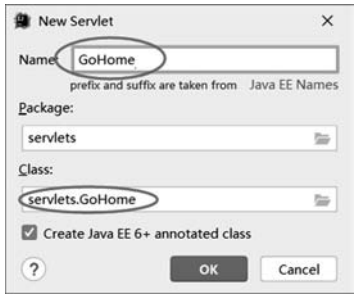

图 5-7 输入 Servlet 名称

（3）完整代码如下：

```
@WebServlet(name = "GoHome", value = "/toHome")                    ①
public class GoHome extends HttpServlet {                          ②
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");        ③
        PrintWriter out = response.getWriter();                    ④
        response.setHeader("refresh", "3;URL=index.jsp");          ⑤
        out.println("<span style='color:#ff0000;font-size:0.8em'>请稍候,3s 后自动返回主
页……</span>");
        out.close();
    }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request, response);
    }
}
```

说明：

① 这是一个注解(有些教程称为注释)，用来标注该类是一个 Servlet，同时设置 GoHome 的映射名称为 toHome。映射，就是将 Servlet 虚拟成 Web 网络访问名，这意味着 GoHome 类的 Web 访问地址是：http://localhost:8080/chapter5/toHome。可以将映射名设置成需要的形式，例如，如果将 value = "/toHome"修改成 value = "/to/go.html"，则该 Servlet 的访问地址变成：http://localhost:8080/chapter5/to/go.html。实际上，go.html 在站点中并不存在，这就是所谓的“映射”。后面，我们还会用到各种注解。

② 所有的 Servlet 类，包括 GoHome，都是 HttpServlet 抽象类的子类。

③ 设置响应内容为纯文本 HTML，使用 utf-8 编码。

④ 这其实就是我们前面在 JSP 文件中使用的 out 对象的真正来源。

⑤ 5.3.3 节用过 setHeader()方法，这里通过设置 HTTP 响应头的 refresh，达到 3s 后自动刷新并打开 index.jsp 的效果。

示例 2：修改前面 5.3.4 节的用户登录处理，用 Servlet 类 DoLogin(映射名 login.do)来完成登录处理。

(1) 修改 login.jsp，将原 action="doLogin.jsp"修改为 action="login.do"；

(2) 新建 Servlet 类 DoLogin.java，代码如下：

```
@WebServlet(name = "GoHome", value = "/login.do")
public class DoLogin extends HttpServlet {
      @Override
         protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
            response.setContentType("text/html;charset=utf-8");
            request.setCharacterEncoding("utf-8");
            PrintWriter out = response.getWriter();
            String username = request.getParameter("usr");
            …
            request.getSession().setAttribute("loginer", "sucess");
            …
      }
}
```

大部分代码与 5.3.4 节 doLogin.jsp 中的代码相同，予以省略。加粗代码是 Servlet 中设置 session 属性值的方式，与 doLogin.jsp 中是有差异的，提请注意。

5.5 EL 表达式语言

5.5.1 EL 概述

EL(Expression Language，表达式语言)的目的是使 JSP 更简单、更易读。EL 的使用，需要 JSTL(JSP Standard Tag Library，JSP 标准标签库)的支持。

EL 的使用方式非常简单：

```
${表达式}
```

来看看 JSP、EL 在页面输出“你好,EL!”的差别。JSP 是这样的：

```
<body><% out.print("你好,EL!"); %></body>
```

而 EL 则是这样：

```
<body>${"你好,EL!"}</body>
```

5.5.2 加入 JSTL 依赖

在项目 pom.xml 中加入 JSTL 依赖：

```
<dependency>
    <groupId>javax.servlet.jsp.jstl</groupId>
    <artifactId>javax.servlet.jsp.jstl-api</artifactId>
    <version>1.2.2</version>
</dependency>
<dependency>
    <groupId>org.apache.taglibs</groupId>
    <artifactId>taglibs-standard-impl</artifactId>
    <version>1.2.5</version>
</dependency>
```

5.5.3 内置对象

EL 提供了一些可以直接使用的内置对象,下面列出了一些常用的 EL 内置对象。

◇ param：客户端请求参数对象。

◇ pageScope：当前页面属性对象。

◇ requestScope：本次请求属性对象。

◇ sessionScope：会话对象。

◇ applicationScope：应用对象。

后面结合 EL 的条件输出或循环输出,再举例说明对象的使用方法。

5.5.4 条件输出

根据条件是否成立来决定内容的显示与否。

单条件表达式：

```
<c:if test="表达式">
        …
</c:if>
```

多条件分支表达式：

```
<c:choose>
      <c:when test="表达式">
            …
      </c:when>
            … …
      <c:otherwise>
            …
      </c:otherwise>
</c:choose>
```

这里面的表达式，判断的是服务端的某个变量、对象、session 等，下面结合示例来学习其用法。

示例：用 EL 表达式改写 5.4 节中的登录示例。

首先，改写 5.4.5 节中的 DoLogin 类，去掉 Servlet 中输出 JS 代码的不合理做法；然后，修改主页 index.jsp：用户登录操作完毕后，用红色文字显示登录成功与否的提示。

1）修改 DoLogin 类的 doGet()方法

```
@Override
protected void doGet ( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {
      request.setCharacterEncoding("utf-8");
      String username = request.getParameter("usr");
      String password = request.getParameter("pwd");
      if (username.equals("admin") && password.equals("007"))//登录成功
            request.getSession().setAttribute("loginer", "sucess");
      else                                                    //登录失败
            request.getSession().setAttribute("loginer", "fail");
      response.sendRedirect("index.jsp");                     //转向打开主页
}
```

现在代码简洁了很多。在服务端用 session 属性 loginer 记住用户登录状态：若登录成功，其值设置为 sucess；登录失败则设置为 fail。在前端，JSP 页面用 EL 表达式，取出 session 的值，并进行判断。

2）修改主页 index.jsp

为了方便与 5.3.4 节中的代码进行比较，下面给出了完整代码：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page isELIgnored="false" %>                                          ①
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>主页</title></head>
<body>
<a href="login.jsp">用户登录</a>
<a href="logout.jsp">用户注销</a>
```

```
<c:choose>                                                          ②
    <c:when test="${sessionScope.loginer=='sucess'}">               ③
        <a href='download.jsp'>文件下载</a>
        <span style="color: #FF0000"><br/>登录成功!</span>
    </c:when>
    <c:when test="${sessionScope.loginer!='sucess'}">               ④
        <a href='#' onclick=alert('请先登录!')>文件下载</a>
        <c:if test="${sessionScope.loginer=='fail'}">
            <span style="color: #FF0000"><br/>用户名或密码错误,登录失败!</span>
        </c:if>
    </c:when>
</c:choose>
</body></html>
```

说明：

① 这句页面指令，启用 EL 表达式。紧跟的下面这句指令，则定义使用 JSTL 标签库。要在 JSP 页面使用 EL，不要忘了放置这两条加粗字体的指令！

② 使用多条件分支表达式，进行登录成功与否判断，也可以改为<c:if>语句。

③ 利用 EL 内置对象中的 sessionScope，获取后台 Servlet 类 DoLogin 中设置好的 loginer 值，进行判断。这里也可简写为：<c:when test="${loginer=='sucess'}">。

④ loginer!='sucess'，包含两种情况：用户压根就没进行登录操作，这时候 loginer 为 null 值，不能在页面显示登录失败的类似字样；用户进行了登录操作，但失败，这时候 loginer 值为 fail。

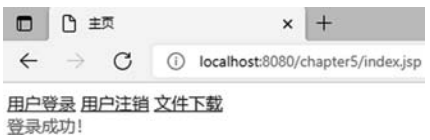

图 5-8　登录成功

图 5-8 是登录成功后的效果图。使用 EL 表达式后，index.jsp 页面不再有任何 JSP 代码块，可读性更好。请大家与 5.3.4 节中的 index.jsp 进行比较，充分理解代码的含义。

5.5.5　循环输出

循环输出语法格式如下：

```
<c:forEach var="变量" [items="集合"][begin="起始位置"][end="结束位置"] [step="步长"]>
    …
</c:forEach>
```

中括号[]中的内容是可选的。下面的示例，动态输出 5 行文本框：

```
<c:forEach var="i" begin="1" end="10" step="2">
    文本框${i}<input type="text" name="txt${i}" placeholder=
"EL${i}…"><br/>
</c:forEach>
```

图 5-9　动态生成文本框

在网页中的效果如图 5-9 所示。

5.6 监听器

5.6.1 监听器类型

监听器(Listener)是一种特殊的 Servlet 类,专门用来监听 Web 应用或特定 Java 对象的初始化或销毁、方法调用、属性改变等事件。当被监听者发生相应事件后,监听器会立即执行事先实现的方法。主要有以下三种类型的监听器。

◇ ServletContextListener:Web 应用监听器。主要有两个方法——contextInitialized(),Web 应用发布时调用此方法,可在此方法中做一些初始化工作;contextDestroyed(),Web 应用被注销或服务器停止时被调用,可在此方法中做一些扫尾工作。在第 7 章会使用这个监听器。

◇ HttpSessionListener:会话 session 监听器。主要有两个方法——sessionCreated(),创建 session 调用;sessionDestroyed(),销毁 session 时调用。

◇ HttpSessionAttributeListener:session 属性监听器。主要有三个方法——attributeAdded(),向 session 中添加新属性时调用;attributeRemoved(),从 session 中移除属性时被调用;attributeReplaced(),session 属性被替换时调用。

5.6.2 基于监听器的在线用户统计

用监听器统计在线用户,基本思想是:先定义一个 HashMap 对象,存放到 application 对象里面。当有用户打开站点时,sessionCreated()被触发,就可以在 sessionCreated()里面将代表该用户的 session id 存入 HashMap 对象里面。若用户超时导致 session 失效,会触发 sessionDestroyed(),可在此将对应的 session id 从 HashMap 对象中移除。这样,HashMap 对象的容量大小,就表示在线用户的数量,而 JSP 页面通过 application.getAttribute()即可获取到该数量值。

在 servlet 包上面右击鼠标,在弹出的快捷菜单中选择 New→Web Listener,输入监听器类名称 AppListener,如图 5-10 所示。

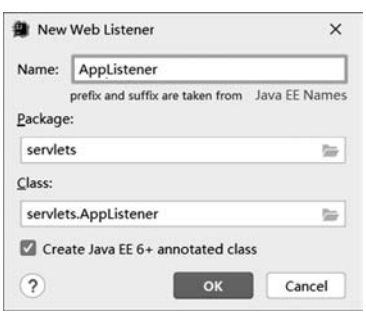

图 5-10 创建监听器

修改 AppListener 类,主要代码如下:

```
@WebListener //注解为监听器
public class AppListener implements ServletContextListener, HttpSessionListener {
                                                                                    ①
    private Map<String, HttpSession> map = new ConcurrentHashMap(1);                ②

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        sce.getServletContext().setAttribute("online", map);                        ③
    }
    @Override
```

```
        public void contextDestroyed(ServletContextEvent sce) {
            map.clear();                                    //清除内容
        }
        @Override
        public void sessionCreated(HttpSessionEvent se) {
            HttpSession session = se.getSession(); 获取 session 对象
            session.setMaxInactiveInterval(60 * 20);                                    ④
            map.put(session.getId(), session);          //当前 session 存入 map
        }
        @Override
        public void sessionDestroyed(HttpSessionEvent se) {
            map.remove(se.getSession().getId());        //移除 session
        }
    }
```

说明：

① 只需要使用两个监听器类：Web 应用监听器、会话 session 监听器。

② 新建一个线程安全、支持高并发的 ConcurrentHashMap 对象，以便保存 session。这里将 HashMap 对象初始值设为 1，后面往 HashMap 里面 put 内容时，其容量会自动增长。

③ 设置 application 对象属性 online，这样前端 JSP 页面就可通过 EL 内置对象 applicationScope 获取到 online 的值。

④ 设置 session 的有效期限为 20min。在前面学习过 HTTP 是无状态协议，只好通过 session 记录状态信息。有些网站在用户登录后，基于安全考虑，会设置了一个有效时间期限，超限则予以自动注销，就是基于设置 session 有效期限的基本原理。

现在，只需要在 JSP 页面简单使用下面的语句，即可显示在线用户数。

```
在线用户数：${applicationScope.online.size()}
```

提示：为了保证在线用户数的准确性，建议大家将站点发布到 Tomcat 下运行，而不是在 IntelliJ IDEA 里面运行测试。

5.7 与数据库交互

数据库是计算机技术领域最重要的发展产物，是大多数信息管理系统的基础框架，是现代应用最广泛的技术之一。很多网站的背后，都有数据库技术的支持。

5.7.1 创建 users 表并加入数据库依赖

1. 创建 users 表

先设计一个用户表 users，以便后续使用，其结构如图 5-11 所示。这些字段分别表示用户名、密码、用户头像、角色（例如 teacher、student）、email 地址。

图 5-11 users 表结构

2. 加入数据库依赖

要与数据库交互,需要在 pom.xml 中加入以下数据库依赖:

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.2.23</version>
</dependency>
```

5.7.2 数据库连接

1. 连接原理

按照第 1 章确定的技术环境,连接数据库的过程如图 5-12 所示。

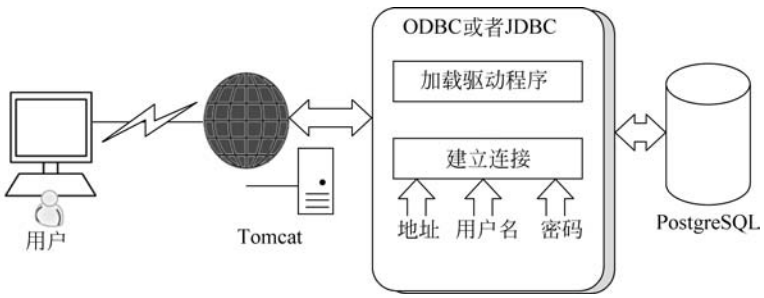

图 5-12 数据库连接原理

从图可知,ODBC 或 JDBC 驱动相当于 Web 应用与数据库之间的“桥梁”。由于 ODBC(Open Database Connectivity,开放数据库连接)连接数据库的效率低于 JDBC,且影响了跨平台特性,现实中使用较少,这里就不做介绍了。JDBC(Java DataBase Connectivity,Java 数据库连接)是由原 Sun 公司提供的统一的标准应用程序编程接口,能够与各种数据库进行交互。

2. 驱动程序

JDBC 驱动程序通常是一个或多个 jar 文件，它实际上是封装数据库访问的类库。驱动程序名一般采用“包名.类名”的形式。例如：

◇ org.postgresql.Driver：PostgreSQL 数据库。其中，org.postgresql 是包名，而 Driver 才是真正的驱动程序类。

◇ com.mysql.jdbc.Driver：MySQL 数据库。

◇ com.microsoft.jdbc.sqlserver.SQLServerDriver：SQL Server 数据库。

驱动程序需要进行如下加载后才能使用。

```
Class.forName("org.postgresql.Driver");
```

3. 连接地址

建立连接还需提供数据库地址 URL、能够访问数据库的用户（含用户名、密码）。URL 一般由三部分组成，各部分间用冒号分隔，格式如下：

```
jdbc:<子协议>:<子名称>
```

<子协议>是指数据库连接机制名。不同数据库厂商的数据库连接机制名是不同的，例如 MySQL 数据库使用的是 mysql，PostgeSQL 数据库使用的是 postgresql。<子名称>提供定位数据库的更详细信息，包括服务器、端口、数据库名等。例如：

◇ jdbc:postgresql://localhost:5432/tamsdb：通过 5432 端口连接本地服务器上的 PostgreSQL 数据库 tamsdb。

◇ jdbc:mysql://localhost:3306/mydb：通过 3306 端口连接本地服务器上的 MySQL 数据库 mydb。

◇ jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=jspdb：通过 1433 端口连接本地服务器上的 SQL Server 数据库 jspdb。

5.7.3 JDBC 应用

1. 驱动管理器 DriverManager

DriverManager 是 JDBC 的管理层，作用于用户和驱动程序之间，负责加载驱动程序，建立数据库和驱动程序之间的连接。DriverManager 有三种使用格式：

◇ DriverManager.getConnection(String url)

◇ DriverManager.getConnection(String url, String user, String password)

◇ DriverManager.getConnection(String url,Properties info)

示例：

```
Class.forName("org.postgresql.Driver");
String url = "jdbc:postgresql://localhost:5432/tamsdb";
Connection conn = null;
conn = DriverManager.getConnection(url, "admin", "007");
```

或者：

```
Properties prop = new Properties();
prop.put("user", "admin");
prop.put("password", "007");
conn = DriverManager.getConnection(url, prop);
```

2. 连接 Connection

一个连接对象代表与特定数据库的会话。Connection 的重要方法如下。

◇ close()：关闭连接。

◇ createStatement()：创建会话声明对象。

◇ prepareStatement(String sql)：预编译 SQL 语句。

◇ setAutoCommit(boolean autoCommit)：设置 Connection 是否处于自动提交模式，默认为自动提交。如果是自动提交，则所有的 SQL 语句将被立即执行。否则，SQL 语句整体作为一个数据库的事务，由 commit()提交，由 rollback()撤销。

◇ commit()：提交事务。

◇ rollback()：回滚事务。撤销当前事务处理所做的任何改变，回到最初的状态。

3. 会话 Statement 和 PreparedStatement

Statement 代表发送 SQL 语句的一次会话，常用方法有以下几种。

◇ close()：关闭会话。

◇ executeQuery(String sql)：执行 SQL 语句并返回结果集。

◇ setFetchSize(int rows)：设置从数据库预读取 rows 条记录。

◇ setMaxRows(int max)：设置从数据库返回结果中结果集的最大行号数为 max。

◇ executeUpdate(String sql)：执行数据更新(update、delete、insert)SQL 语句。

◇ addBatch(String sql)：将 SQL 语句加入批量更新列表。批量更新可以提高程序执行效率，优化数据处理过程。

◇ executeBatch()：执行批量更新。

PreparedStatement 是 Statement 的子类，可存储一条预编译的 SQL 语句，并可高效、多次执行该语句。PreparedStatement 在一些场景应用中比 Statement 执行效率更高。实践中，对于重复性发生的业务，使用预编译有性能上的优势，例如用户登录，这是一种操作没变只是操作的数据发生变化的业务，建议使用预编译方法。

使用 Statement：

```
conn = DriverManager.getConnection(url, "admin", "007");
Statement s = conn.createStatement();
```

第 2 句代码可以换成预编译方式：

```
String sql = " select username,logo,role from users";
PreparedStatement ps = conn.prepareStatement(sql);
```

4. 结果集

通过 SQL 语句查询数据库后，获得需要的数据结果（结果集，或称记录集），然后再根据业务要求进行各种处理后，将数据返回给前端 JSP 页面进行显示。获取结果集 ResultSet 的方法如下：

```
Statement s = conn.createStatement();
ResultSet rs = s.executeQuery("select username,logo,role from users");
```

或者：

```
String sql = "select username,logo,role from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
```

拿到结果集后，就可以对其中的每个对象（对应于数据表中的每条记录）进行操作。ResultSet 提供了许多 getXXX()方法，来获取当前行对象的属性值。这里的 XXX，表示要根据属性的类型，选择相应的方法。例如：

```
String sname = rs.getString(1);   //获取 sname 姓名
String sex = rs.getString("sex"); //获取性别
int score = rs.getInt(3);         //获取 score 分数，也可以用 rs.getInt("score")
```

一般来说，使用索引方式（获取姓名代码）而非名称（获取性别代码）执行速度更快。与此相对应，ResultSet 提供了许多 setXXX()方法来设置属性值。除了 getXXX()、setXXX()方法外，表 5-1 列出了 ResultSet 其他常用方法。

表 5-1 ResultSet 常用方法

方 法	功 能 说 明
boolean next()	移动到下一行并判断是否到末尾
boolean absolute(int row)	移动到结果集指定行
void close()	关闭结果集
void deleteRow()	删除当前行
void first()	移动到结果集的第 1 行
int getRow()	获得当前行的行号
void insertRow()	插入新行
void updateRow()	更新当前行

示例 1：循环遍历 users 表的查询结果集并输出。

```
String sql = "select username,password from users";
PreparedStatement ps = conn.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
while (rs.next()) {                                          //循环遍历
     out.print(rs.getString(1) + " " + rs.getString(2));     //输出用户名、密码
}
rs.close();                                                  //关闭结果集
ps.close();
```

示例 2：修改用户 aaa 的密码为 123456。

```
String sql = "update users set password = '123456' where username = 'aaa'";
PreparedStatement ps = myconn.prepareStatement(sql);
ps.executeUpdate();
```

示例 3：修改 5.5.4 节 DoLogin 类的 doGet()方法，实现基于数据库的用户登录。

只需要修改 doGet()方法的代码如下：

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
      request.setCharacterEncoding("utf-8");
      String username = request.getParameter("usr");
      String password = request.getParameter("pwd");
      request.getSession().setAttribute("loginer", "fail");                                    ①
      String url = "jdbc:postgresql://localhost:5432/tamsdb";
      Connection conn = null;
      try {
            Class.forName("org.postgresql.Driver");                    //加载 JDBC 驱动程序
            conn = DriverManager.getConnection(url, "admin", "007");  //建立连接
            String sql = "select * from users where username = ? and password = ?";          ②
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, username);                                                       ③
            ps.setString(2, password);
            ResultSet rs = ps.executeQuery();
            if (rs.next())                                             //查找到记录,登录成功
                  request.getSession().setAttribute("loginer", "sucess");
            rs.close();
            ps.close();
      } catch (Exception e) { e.printStackTrace();
      } finally {
            if (conn != null) {
                  try { conn.close();
                  } catch (SQLException e) { e.printStackTrace();}
            }
            response.sendRedirect("index.jsp");
      }
}
```

说明：

① 用 session 记录登录状态，默认假定用户登录失败。

② 定义查询的 SQL 语句。这里的问号？是占位符，不妨这样理解：当不清楚用户名、密码的具体值时，用问号占位，等待后续传值过来。

③ 将前面接收的用户名，传值给第 1 个问号占位符。

5.8 场景应用示例


视频讲解

5.8.1 文件上传

文件上传是 Web 应用中很常见的功能，一般采用第三方上传组件来实现。读者也可以自力更生实现文件上传。

1. 上传数据的编码格式

浏览器在上传文件时，是按照一定数据编码格式进行的：传送数据时在头尾部分附加了额外信息，具体如下：

```
-----------------------------7d429871607fe
Content-Disposition: form-data;name="表单控件名";filename="上传文件的路径和文件名"
Content-Type: text/plain
上传文件的内容……
-----------------------------7d429871607fe
Content-Disposition: form-data; name="filename" Content-Type: application/octet-stream
-----------------------------7d429871607fe--
```

在传送数据的头部、尾部都辅以分割线，并附加了 Content-Disposition 信息，例如 form-data、表单文件控件名、上传文件名等。如果自己编写文件上传处理，将数据写入服务器时，要忽略掉这些附加数据。

2. 编写上传 JSP 文件 upload.jsp

```
<Script language="javascript">
      function check() {
            if (document.uploadForm.myfile.value == "") {
                  alert("请选择待上传文件!");
                  return false;
            }
            return true;
      }
</Script>
<form name="uploadForm" method="post" action="upload.do"
        enctype="multipart/form-data" onsubmit="return check()">
      请选择待上传文件<input type="file" id="myfile" name="myfile"><br>
      <input value="开始上传" type="submit">
</form>
```

这个表单与常规纯文本处理的表单稍有不同，请注意粗体部分的代码，用于定义文件传输的 HTTP 信息。另外，<input>的类型是 file。

3. 编写 Servlet 类 FileUpload

```
@WebServlet(name = "FileUpload", value = "/upload.do")          //映射为 upload.do
@MultipartConfig                                                //配置上传文件支持
public class FileUpload extends HttpServlet {
      @Override
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
            request.setCharacterEncoding("utf-8");
            response.setContentType("text/html;charset=UTF-8");
            String destPath = request.getServletContext().getRealPath("/upfiles");        ①
            Part part = request.getPart("myfile");              // myfile 对应表单的文件输入框
            String header = part.getHeader("content-disposition");
            String fileName = getFileName(header);              //获得文件名
            part.write(destPath + File.separator + fileName);  //写入服务器
            response.getWriter().print("文件上传成功!");
      }
      private String getFileName(String header) {               //获取文件名
            String fieldName = null;
            if (header != null && header.startsWith("form-data")) {
                  int start = header.indexOf("filename=\"");                                ②
                  int end = header.indexOf('"', start + 10);                               ③
                  if (start != -1 && end != -1) {
                        fieldName = header.substring(start + 10, end);
                  }
            }
            return fieldName;
      }
}
```

说明：

① 上传的文件都存放到站点 upfiles 文件夹下，事先需要在 webapp 下创建好该文件夹。

② 找到头部附加信息中“filename=”的位置，目的是从 filename 值中获取到文件名。

③ 从“filename=”后面一个字符开始(因为“filename=”长度为 9)，找到 filename 属性值字符串，即文件名。

5.8.2 在页面中显示 Excel 表格

使用 Servlet 显示上一节上传的 Excel 表格文件。图 5-13 为上传的 Excel 文件 reimb.xls，图 5-14 则是 Servlet 输出到网页后的效果。二者样式上会有些许差异，但整体还算不错。

1. 加入 POI 依赖

对 Excel 的处理，需要用到 Apache 软件基金会的著名 Excel 处理工具 POI。POI 提供了最完整、最正确的 Excel 格式读取实现，并采用了内存优化方式处理 Excel 表格，可以帮助读者轻松读写 Excel 文件。POI 免费、开源，可到其官网下载。在 pom.xml 中添加依赖项：

管理学院报销单

部门名称(签章)：						制表时间：		
编号	姓名	联系电话	身份证号	银行账号	开户行	税前金额	纳税金额	税后金额
1					省 市 银行 支行		0.00	0.00
2					省 市 银行 支行		0.00	0.00
3					省 市 银行 支行		0.00	0.00
4					省 市 银行 支行		0.00	0.00
5					省 市 银行 支行		0.00	0.00
6					省 市 银行 支行		0.00	0.00
7					省 市 银行 支行		0.00	0.00
8					省 市 银行 支行		0.00	0.00
9					省 市 银行 支行		0.00	0.00
10					省 市 银行 支行		0.00	0.00
					小计：	0.00	0.00	0.00
经费出处：					合计人民币： 元整（¥ 元）			

图 5-13 Excel 报销单

管理学院报销单

部门名称(签章)：						制表时间:		
编号	姓名	联系电话	身份证号	银行账号	开户行	税前金额	纳税金额	税后金额
1					省 市 银行 支行		0.00	0.00
2					省 市 银行 支行		0.00	0.00
3					省 市 银行 支行		0.00	0.00
4					省 市 银行 支行		0.00	0.00
5					省 市 银行 支行		0.00	0.00
6					省 市 银行 支行		0.00	0.00
7					省 市 银行 支行		0.00	0.00
8					省 市 银行 支行		0.00	0.00
9					省 市 银行 支行		0.00	0.00
10					省 市 银行 支行		0.00	0.00
					小计:	0.00	0.00	0.00
经费出处:					合计人民币: 元整（¥ 元）			

图 5-14 网页显示 Excel 报销单

```
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>5.0.0</version>
```

```
</dependency>
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-scratchpad</artifactId>
    <version>5.0.0</version>
</dependency>
```

2. 编写 Servlet 处理 Excel

```
@WebServlet(name = "WebExcel", value = "/excel")                    //地址映射为 excel
public class WebExcel extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException {
        String excelFile = request.getServletContext().getRealPath("upfiles/reimb.xls");
        HSSFWorkbook wb = ExcelToHtmlUtils.loadXls(new File(excelFile));
        wb.setSheetName(0, " ");                                    //去掉 Excel Sheet 名称
        writeToHTML(response, wb);
        wb.close();
    }

    private void writeToHTML(HttpServletResponse response, HSSFWorkbook wb) {
        response.setContentType("text/html;charset=UTF-8");
        try {
            Document doc = XMLHelper.newDocumentBuilder().newDocument();
            ExcelToHtmlConverter converter = new ExcelToHtmlConverter(doc);
            converter.setOutputColumnHeaders(false);                //不显示列头
            converter.setOutputRowNumbers(false);                   //不显示行号
            converter.processWorkbook(wb);

            Properties prop = new Properties();
            prop.setProperty("encoding", "UTF-8");                  //设置编码
            prop.setProperty("indent", "yes");                      //缩进
            prop.setProperty("method", "html");                     //转换模式: html
            Transformer tf = TransformerFactory.newInstance().newTransformer();
            tf.setOutputProperties(prop);
            tf.transform(new DOMSource(converter.getDocument()),
                    new StreamResult(response.getOutputStream()));//数据流输出
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

这里只是结合 Servlet,用一个小示例对 POI 抛砖引玉。实际上,POI 还可以帮助创建、读取或输出 Excel 文件,具有强大的 Excel 处理能力,读者需要在实践中不断探索、掌握其用法。

注意：上述方法只能处理 xls 文件，对于 xlsx 则需要更复杂的方法。


视频讲解

5.8.3 用 PDF 显示古诗

1. 应用需求

在前面已经领略到了 Servlet 强大的处理能力，接下来拓展其应用、加深理解：用 Servlet 在页面以 PDF 方式显示一首古诗。PDF 文件的应用非常广泛，很多网站常常使用 PDF 向用户展示内容。若要方便处理 PDF 文档，需要 iText 配合使用。

2. iText 简介

iText 是面向 Java、.NET 的强大的 PDF 处理工具，提供了一套通用、可编程和企业级 PDF 解决方案，并可嵌入到其他各类应用中，例如与 Servlet 结合。

iText 官网提供了开源免费版和商业版两种形式。

3. 加入 iText 依赖

要使用 iText，需先在 pom.xml 中加入依赖：

```
<dependency>
    <groupId>com.itextpdf</groupId>
    <artifactId>itext7-core</artifactId>
    <version>7.1.16</version>
    <type>pom</type>
</dependency>
```

4. 中文问题

iText 对中文支持不尽如人意。尽管官方也提供了处理中文的方法，但是略显遗憾的是，少数情况下某些中文会出现乱码。一个可行的解决方法是：使用 Windows 自带的字体来支持中文显示。只需要以下两个步骤。

(1) 在项目 webapp 下新建文件夹 font，用于存放字体文件；

(2) 将 C:\Windows\Fonts 文件夹下的某个字体文件，例如华文彩云(STCAIYUN.TTF)，复制到 font 文件夹下。

下面，就可以用这个字体文件，以 PDF 文件格式显示一首古诗。

5. 具体实现

在 servlets 包下新建 Servlet 类 PdfPoem.java，主要代码如下：

```
@WebServlet(name = "PdfPoem", value = "/pdf")                    //地址映射为 pdf
public class PdfPoem extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException {
        ServletOutputStream stream = response.getOutputStream();
        PdfWriter writer = new PdfWriter(stream);
        PdfDocument pdfDoc = new PdfDocument(writer);            //创建 PDF 文档对象
```

```
            Document doc = new Document(pdfDoc, PageSize.A4);          //页面 A4 大小
            //获取字体文件的绝对路径,getRealPath()返回绝对路径
            String font = request.getServletContext().getRealPath("/font/STCAIYUN.TTF");
            PdfFont fontChinese = PdfFontFactory.createFont(font,
                                  PdfEncodings.IDENTITY_H);            //创建 PDF 中文字体对象
            Paragraph header = new Paragraph("秋夕\n【唐】杜牧");      //Pdf 段落,\n 为换行
            header.setFont(fontChinese);                               //使用华文彩云字体
            header.setFontSize(16);
            header.setTextAlignment(TextAlignment.CENTER);
            doc.add(header);
            List poem = new List().setListSymbol("\u2022");            //无序列表
            poem.setFont(fontChinese);
            poem.setFontSize(14);
            poem.setTextAlignment(TextAlignment.CENTER);
            poem.add(new ListItem("银烛秋光冷画屏,"));                 //每句诗作为列表项添加到列表
            poem.add(new ListItem("轻罗小扇扑流萤."));
            poem.add(new ListItem("天阶夜色凉如水,"));
            poem.add(new ListItem("卧看牵牛织女星."));
            doc.add(poem);
            doc.close();
        }
        @Override
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {
            doGet(request, response);
        }
    }
```

从上述处理过程来看,iText 的代码还是非常简洁、清晰、易读的。在浏览器地址打开 http://localhost:8080/chapter5/pdf,将显示如图 5-15 所示的 PDF 文件。

图 5-15 PDF 古诗(1)

注意:这里在导入类的时候,若有多个选择,一般请选择属于 iText 的类。不少人出错,很大程度上是因为导入了错误的类。

5.9 场景任务挑战——有背景图的 PDF 古诗

在数据库中创建存放古诗的 poems 表，请合理设计出该表的相应字段。新建 Servlet 类 LatestPoem，地址映射为/poem，每次读取 poems 表中最新录入的那首古诗，然后用 6 行 1 列的无边框表格，显示整首诗。整个表格的背景需要设置成一张图片，并将字体改用其他字体，例如华文隶书，如图 5-16 所示。

图 5-16　PDF 古诗(2)