

第 1 章 JSP 简介	10
1.1 什么是 JSP	10
1.2 JSP 页面	10
1.3 JSP 的运行原理	12
1.4 安装配置 JSP 运行环境	16
1.5 JSP 页面的测试	20
1.6 JSP 与 Java Servlet 的关系	23
第 2 章 JSP 语法	25
2.1 JSP 页面的基本结构	25
2.2 变量和方法的声明	29
2.2.1 声明变量	29
2.2.2 声明方法	32
2.2.3 声明类	37
2.3 Java 程序片	40
2.4 表达式	45
2.5 JSP 中的注释	46
2.6 JSP 指令标签	49
2.6.1 page 指令	49

2.6.2	include 指令标签	54
2.7	JSP 动作标签	59
2.7.1	include 动作标签	59
2.7.2	param 动作标签	64
2.7.3	forward 动作标签	65
2.7.4	plugin 动作标签	69
2.7.5	useBean 动作标签	75
第 3 章 JSP 内置对象		76
3.1	request 对象	77
3.1.1	获取客户提交的信息	78
3.1.2	处理汉字信息	83
3.1.3	常用方法举例	85
3.1.4	用户注册	93
3.1.5	获取 HTML 表单提交的数据	101
3.1.6	表格	112
3.2	response 对象	116
3.2.1	动态响应 contentType 属性	116
3.2.2	response 的 HTTP 文件头	122
3.2.3	response 重定向	124

3.2.4	response 的状态行	125
3.3	session 对象	132
3.3.1	session 对象的 Id	132
3.3.2	session 对象与 URL 重写	135
3.3.3	session 对象的常用方法:	140
3.3.4	计数器	150
3.4	application 对象	153
3.4.1	application 对象的常用方法	153
3.4.2	用 application 制作留言板	155
3.5	out 对象	160
第 4 章 JSP 中的文件操作		166
4.1	File 类	166
4.1.1	获取文件的属性	166
4.1.2	创建目录	169
4.1.3	删除文件和目录	174
4.2	使用字节流读写文件	175
4.2.1	FileInputStream 和 FileOutputStream 类	177
4.2.2	BufferedInputStream 和 BufferedOutputStream 类	179
4.3	使用字符流读写文件	183

4.3.1	FileReader 和 FileWriter 类	184
4.3.2	BufferedReader 和 BufferedWriter 类	185
4.4	回压字符流	193
4.5	数据流	196
4.6	对象流	204
4.7	RandomAccessFile 类	214
4.8	文件上传	223
4.9	文件下载	232
4.10	分页读取文件	235
4.11	标准化考试	240
第 5 章 JSP 中使用数据库		251
5.1	数据源	251
5.2	JDBC-ODBC 桥接器	258
5.3	查询记录	259
5.3.1	顺序查询	262
5.3.2	游动查询	266
5.3.3	随机查询	271
5.3.4	参数查询	275
5.3.5	排序查询	282

5.3.6	分析结果集查询	286
5.3.7	使用统配符查询	289
5.4	更新记录	290
5.5	添加记录	297
5.6	删除记录	305
5.7	分页显示记录	311
5.8	连接数据库的其它方式	315
5.8.1	连接 Oracle 数据库	315
5.8.2	连接 MySQL 数据库	318
5.9	查询 Excel 电子表格	319
5.10	使用同步连接	323
5.11	网上投票	327
5.12	成绩录入查询系统	336
第 6 章	JSP 与 JavaBeans	362
6.1	编写 javabeans 和使用 javabeans	363
6.1.1	编写 beans	363
6.1.2	使用 beans	364
6.2	beans 的存放目录	372
6.3	获取和修改 beans 的属性	376

6.3.1	getProperty 动作标签	376
6.3.2	setProperty 动作标签	379
6.4	beans 的辅助类	389
6.5	带包名的 beans	393
6.6	JSP 与 beans 结合的简单例子	397
6.6.1	三角形 beans	397
6.6.2	计数器 beans	400
6.6.3	购物车 beans	402
6.6.4	读文件 beans	412
6.6.5	写文件 beans	420
6.6.6	查询数据库 beans	428
6.6.7	猜数字 beans	432
6.6.8	标准化考试 beans	437
6.6.9	日期 beans	446
6.6.10	分页显示记录 beans	450
第 7 章 基于会员制的网络交友		461
7.1	系统设计	461
7.2	数据库设计及连接	462
7.3	页面管理	464

7.4	各个页面的设计	466
7.4.1	会员注册	467
7.4.2	会员登录	479
7.4.3	浏览会员	486
7.4.4	查找会员	495
7.4.5	留言板	502
7.4.6	查看公共留言	514
7.4.7	查看私人留言	521
7.4.8	修改密码	532
7.4.9	修改个人信息	537
第 8 章	网上书店	543
8.1	系统设计	543
8.2	数据库设计及连接	544
8.3	页面管理	546
8.4	各个页面的设计	547
8.4.1	用户注册	548
8.4.2	用户登录	557
8.4.3	用户订购	565
8.4.4	查看订单	578

8.4.5	修改订单	581
8.4.6	浏览书目	591
8.4.7	修改密码	601
8.4.8	修改个人信息.....	606
第 9 章 Java Servlet		612
9.1	SERVLET 工作原理	612
9.1.1	Servlet 的生命周期	612
9.1.2	init 方法:	613
9.1.3	service 方法.....	614
9.1.4	destroy 方法.....	614
9.2	编译和安装 SERVLET	615
9.2.1	简单的 servlet 例子	615
9.2.2	编译 servlet.....	616
9.2.3	存放 servlet 的目录	617
9.2.4	运行 servlet.....	618
9.2.5	带包名的 servlet	620
9.3	通过 JSP 页面调用 SERVLET	621
9.3.1	通过表单向 servlet 提交数据	621
9.3.2	通过超链接访问 servlet	624

9.4	SERVLET 的共享变量	625
9.5	HTTPServlet 类	629
9.5.1	doGet 方法和 doPost 方法	629
9.5.2	处理 HTTP 请求头及表单信息	637
9.5.3	设置响应的 HTTP 头	646
9.6	用 Servlet 读写文件	651
9.6.1	读取文件的内容	652
9.6.2	写文件	659
9.7	用 Servlet 访问数据库	663
9.7.1	数据库记录查询	664
9.7.2	使用共享连接	672
9.8	会话管理	676
9.8.1	获取用户的会话	676
9.8.2	购物车	680
9.8.3	猜数字	687

第1章 JSP 简介

1.1 什么是 JSP

JSP 是 **Java Server Pages** 的缩写，是由 **Sun** 公司倡导、许多公司参与，于 1999 年推出的一种动态网页技术标准。**JSP** 是基于 **Java Servlet** 以及整个 **Java** 体系的 **Web** 开发技术，利用这一技术可以建立安全、跨平台的先进动态网站，这项技术还在不断的更新和优化中。你可能对 **Microsoft** 的 **ASP(Active Server Pages)** 比较熟悉，**ASP** 也是一个 **Web** 服务器端的开发技术，可以开发出动态的、高性能的 **Web** 服务应用程序。**JSP** 和 **ASP** 技术非常相似，**ASP** 的编程语言是 **VBScript** 和 **JavaScript**，**JSP** 使用的是 **Java**。与 **ASP** 相比，**JSP** 以 **Java** 技术为基础，又在许多方面做了改进，具有动态页面与静态页面分离，能够脱离硬件平台的束缚，以及编译后运行等优点，完全克服了 **ASP** 的脚本级执行的缺点。我们相信 **JSP** 会逐渐成为 **Internet** 上的主流开发工具。

需要强调的一点是：要想真正地掌握 **JSP** 技术，必须有较好的 **Java** 语言基础，以及 **HTML** 语言方面的知识。

1.2 JSP 页面

在传统的 **HTML** 页面文件中加入 **java** 程序片和 **JSP** 标签就构成了一个 **JSP** 页面文件,简单地说，一个**JSP** 页面除了普通的 **HTML** 标记符外，再使用标记符号“<%”,“%>”加入 **Java** 程序片。一个**JSP** 页面文件的扩展名是 **jsp**，文件的名字必须符合标识符规定，需要注意的是，**JSP** 技术基于 **Java** 语言，名字区分大小写。

为了明显地区分普通的 **HTML** 标记和 **Java** 程序片段以及 **JSP** 标签，我们用大写字母书写普通的 **HTML** 标记符号。

下面的例子 1 是一个简单的 **JSP** 页面。

例子 1（效果如图 1.1 所示）

Example1_1.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY BGCOLOR=cyan>

<FONT Size=1>

<P>这是一个简单的 JSP 页面

    <% int i, sum=0;

        for(i=1;i<=100;i++)

            { sum=sum+i;

              }

    %>

<P> 1 到 100 的连续和是:

<BR>

    <%=sum %>

</FONT>

</BODY>

<HTML>
```

用浏览器访问该 JSP 页面的效果如图 1.1 所示:

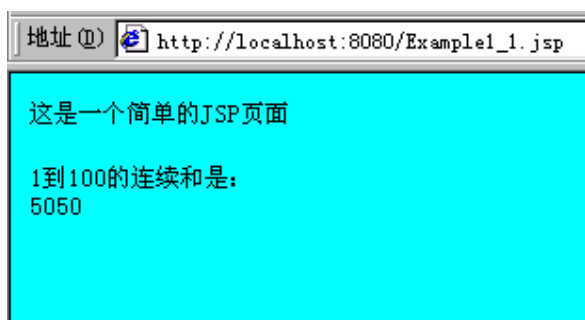


图 1.1 简单的 jsp 页面

1.3 JSP 的运行原理

当服务器上的一个 **JSP** 页面被第一次请求执行时，服务器上的 **JSP** 引擎首先将 **JSP** 页面文件转译成一个 **java** 文件，再将这个 **java** 文件编译生成字节码文件，然后通过执行字节码文件响应客户的请求，而当这个 **JSP** 页面再次被请求执行时，**JSP** 引擎将直接执行这个字节码文件来响应客户，这也是 **JSP** 比 **ASP** 速度快的一个原因。而 **JSP** 页面的首次执行往往由服务器管理者来执行。这个字节码文件的主要工作是：

- (1) 把 **JSP** 页面中普通的 **HTML** 标记符号（页面的静态部分）交给客户的浏览器负责显示。
- (2) 执行 “<%” 和 “%>” 之间的 **java** 程序片（**JSP** 页面中的动态部分），并把执行结果交给客户的浏览器显示。
- (3) 当多个客户请求一个 **JSP** 页面时，**JSP** 引擎为每个客户启动一个线程而不是启动一个进程，这些线程由 **JSP** 引擎服务器来管理，与传统的 **CGI** 为每个客户启动一个进程相比较，效率要高的多。

下面是 **JSP** 引擎生成的 **Example1_1.jsp** 的 **java** 文件，我们把 **JSP** 引擎交给客户端负责显示的内容做了（***）注释。

```

package org.apache.jsp;

import javax.servlet.*;

import javax.servlet.http.*;

import javax.servlet.jsp.*;

import org.apache.jasper.runtime.*;

public class first1$jsp extends HttpJspBase {

    static {

    }

    public first1$jsp() {

    }

    private static boolean _jspx_inited = false;

    public final void _jspx_init() throws org.apache.jasper.runtime.JspException {

    }

    public void _jspService(HttpServletRequest request, HttpServletResponse
response)

        throws java.io.IOException, ServletException {

        JspFactory _jspxFactory = null;

        PageContext pageContext = null;

        HttpSession session = null;

        ServletContext application = null;

        ServletConfig config = null;

        JspWriter out = null;

        Object page = this;

```

```

String _value = null;

try {

    if (_jspx_inited == false) {

        synchronized (this) {

            if (_jspx_inited == false) {

                _jspx_init();

                _jspx_inited = true;

            }

        }

    }

    _jspxFactory = JspFactory.getDefaultFactory();

    response.setContentType("text/html;charset=GB2312");

    pageContext = _jspxFactory.getPageContext(this, request, response,"", true,
8192, true);

    application = pageContext.getServletContext();

    config = pageContext.getServletConfig();

    session = pageContext.getSession();

    out = pageContext.getOut();

    (**) out.write("\r\n<HTML>\r\n<BODY>\r\n<P> 这是一个简单的 JSP 页面
\r\n");

    int i, sum=0;

    for(i=1;i<=100;i++)

        { sum=sum+i;

```

```

        }

        (***)    out.write("\r\n<P> 1        到 100 的连续和是: \r\n<BR>\r\n ");

        (***)    out.print(sum );

        (***)    out.write("\r\n</BODY>\r\n<HTML>\r\n");

    } catch (Throwable t) {

        if (out != null && out.getBufferSize() != 0)

            out.clearBuffer();

        if (pageContext != null) pageContext.handlePageException(t);

    } finally {

        if (_jspxFactory != null)

            _jspxFactory.releasePageContext(pageContext);

    }

}

}

}

```

下面是客户端浏览器查看到的 Example1_1.jsp 的源代码:

```

<HTML>

<BODY BGCOLOR=cyan>

<FONT Size=1>

<P>这是一个简单的 JSP 页面

<P> 1 到 100 的连续和是:

<BR>

5050

```


</BODY>

<HTML>

1.4 安装配置 JSP 运行环境

自从 JSP 发布以后，出现了各式各样的 JSP 引擎。1999 年 10 月 Sun 公司将 Java Server Page 1.1 代码交给 Apache 组织，Apache 组织对 JSP 进行了实用研究，并将这个服务器项目称为 Tomcat，从此，著名的 Web 服务器 Apache 开始支持 JSP。这样，Jakarta—Tomcat 就诞生了（Jakarta 是 JSP 项目的最初名称）。目前，Tomcat 能和大部分主流服务器一起高效率的工作。

我们重点讲述 Window98/Window2000 操作系统下 Tomcat 服务器的安装配置。

安装 Tomcat 之前，必须首先安装 JDK，这里我们安装 sun 公司的 JDK1.3。

假设 JDK 的安装目录是：

C:\Jdk1.3 。

然后，解压缩文件：jakarta-tomcat-4.0.zip，该文件可从 sun 公司的网站：<http://java.sun.com> 或 <http://jakarta.apache.org> 免费得到。假设解压缩文件到：D:\Tomcat。这时我们得到如下的目录结构：

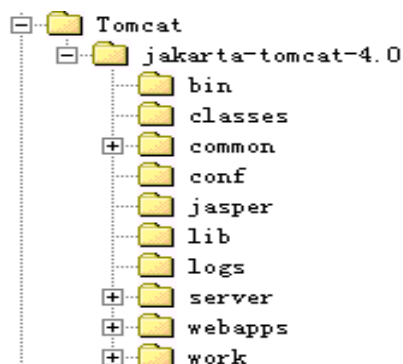


图 1.2 Tomcat 引擎目录结构

在启动 Tomcat 服务器之前，还需要进行几个环境变量的设置。

对于 window2000，用鼠标右键点击“我的电脑”，弹出菜单，然后选择属性，弹出“系统特性”对话框，再单击该对话框中的高级选项，然后点击按钮“环境变量”，分别添加如下的系统环境变量：

变量名：JAVA_HOME，变量值：C:\jdk1.3。

变量名：TOMCAT_HOME，变量值：D:\tomcat\Jakarta-tomcat-4.0。

变量名：CLASSPATH，变量值：C:\jdk1.3\jre\lib\rt.jar;.;。

变量名：PATH，变量值：C:\jdk1.3\bin

如果曾经设置过环境变量：CLASSPATH 和 PATH，可点击该变量进行编辑操作，将需要的值加入即可。

对于 Win9x，用记事本编辑 Autoexec.bat 文件，将如下的设置语句加入即可，

PATH C:\jdk1.3.1_01\bin;

```
SET CLASSPATH=.;C:\jdk1.3.1_01\jre\lib\rt.jar;
```

```
SET TOMCAT_HOME=D:\tomcat\jakarta-tomcat-4.0
```

```
SET JAVA_HOME=C:\jdk1.3.1_01
```

现在，就可以启动 Tomcat 服务器了。执行 Tomcat\Jakarta-Tomcat-4.0\bin 下的 startup.bat，出现如图 1.3 所示的窗口，表明服务器已经启动。

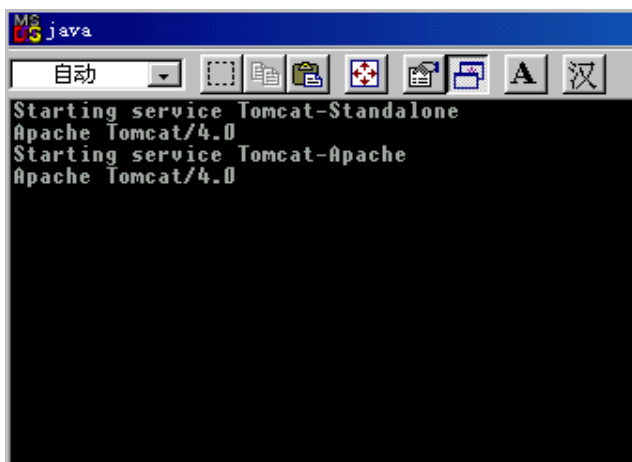


图 1.3 Tomcat 服务器启动后的窗口

在浏览器的地址栏中键入：<http://localhost:8080>，会出现如图 1.4 所示的 Tomcat 测试

页面。



图 1.4 Tomcat 测试页面

注：Tomcat 服务器内置 web 服务。

注：如果 Tomcat 不能启动，请首先检查环境变量的设置是否正确，对于 Win9x，如果出现“out of environment space”的错误提示，就需要修改 MS-DOS 属性，将属性中内存的初始环境更改为 4096。

注：8080 是 Tomcat 服务器的默认端口号。我们可以通过修改 Tomcat\Jakarta-tomcat-4.0\conf 文件下的主配置文件 server.xml，更改端口号。用记事本打开 server.xml 文件，找到出现

```
<!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
```

```
<Connector
```

```
className="org.apache.catalina.connector.http.HttpConnector"

    port="8080" minProcessors="5" maxProcessors="75"

    enableLookups="true" redirectPort="8443"

    acceptCount="10" debug="0" connectionTimeout="60000"/>
```

的部分，将其中的 port= “8080”更改为新的端口号即可，比如将 8080 更改为 9080 等。

注：可以通过执行 Tomcat\Jakarta-tomcat-4.0\bin 下的 shutdown.bat 来关闭 Tomcat 服务器。

1.5 JSP 页面的测试

(1)用记事本或更好的文本编辑器，编辑如下的 JSP 源文件：**Example1_2.jsp**。

例子 2 （效果如图 1.5 所示）

Example1_2.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.util.*" %>

<HTML>

<BODY>

<P> 现在的时间是:

    <% Date date=new Date();

    %>

<BR>

    <%=date%>

</BODY>

<HTML>
```

(2) 将 JSP 文件 命名为 **Example1_2.jsp** ， 保存 到 Tomcat\Jakarta-tomcat-4.0\webapps\root 下。在 浏览器 的地址栏 中输入：
http://localhost:8080/Example1_2.jsp 对 JSP 页面进行测试，出现如图 1.5 效果：

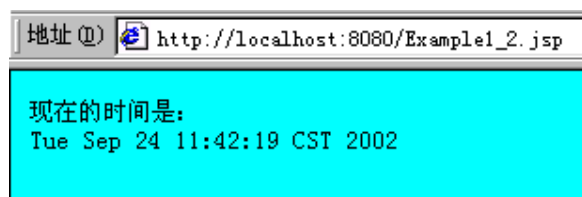


图 1.5 JSP 页面测试

这时，如果你查看 Tomcat\Jakarta-tomcat-4.0\webapps\work 目录，可以在 localhost 的一个根文件下找到 JSP 引擎生成的 Example1_2.jsp 的 java 文件和编译后的字节码文件，如图 1.6 所示。

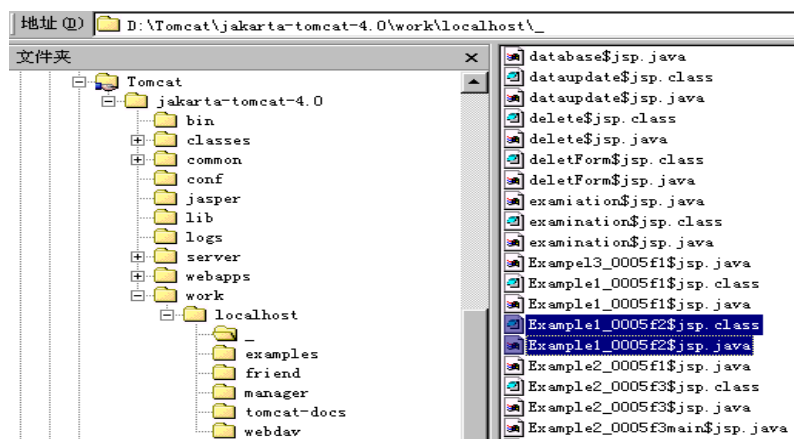


图 1.6 JSP 页面的 Java 文件和字节码文件所在的目录

注：Tomcat 服务器的 Web 服务的根目录是

tomcat\Jakarta-tomcat-4.0\webapps\root 。

另外 webapps 下还有几个 web 服务目录：

examples、tomcat-docs、webdav。

如果将 JSP 文件 保存到 webdav 中， 应当在在浏览 器器的 地址栏 中输入：

http://localhost:8080/webdav/Example1_2.jsp。

注：我们也可以建立新的 web 服务目录。

假设要将 D:\test 以及 c:\redsun 作为服务目录，并让用户分别使用 /test 和 /moon 虚拟目录访问。首先用记事本打开主配置文件 server.xml，找到出现

```
<!-- Tomcat Root Context -->
```

```
<!--
```

```
<Context path="" docBase="ROOT" debug="0"/>
```

```
-->
```

```
<!-- Tomcat Examples Context -->
```

```
<Context path="/examples" docBase="examples" debug="0"
```

```
reloadable="true">
```

```

<Logger className="org.apache.catalina.logger.FileLogger"
    prefix="localhost_examples_log." suffix=".txt"
    timestamp="true"/>
    ...      ...      ...
    ...      ...      ...
</Context>

```

（在这里加入您的 Web 服务目录）

```

</Host>

```

的部分。然后在</Context> 和 </Host> 之间加入：

```

<Context path="/test" docBase="d:/test" debug="0" reloadable="true">
</Context>

<Context path="/moon" docBase="C:/redsun" debug="0" reloadable="true">
</Context>

```

主配置文件 server.xml 修改后，必须重新启动 Tomcat 引擎。这样，就可以将 JSP 页面存放到 D:\test 或 c:\redsun 中，用户可以通过虚拟目录 test 或 moon 访问 JSP 页面，比如，在浏览器地址栏中键入：http://localhost:8080/moon/Example1_2.jsp。

当客户访问过上面 Web 服务目录的 JSP 页面后，服务器的 tomcat\jakart-tomcat-4.0\work 下会出现 moon 和 test 两个子目录，子目录存放了 JSP 页面对应的 Java 文件和 Java 文件的字节码文件。

1.6 JSP 与 Java Servlet 的关系

Java Servlet 是 Java 语言的一部分，提供了用于服务器编程的 API，Java Servlet 编写的 Java 程序称为一个 **servlet**。**servlet** 通过 **HTML** 与客户交互信息。**servlet** 的最大缺点是不能有效的管理页面的逻辑部分和页面的输出部分，导致 **servlet** 代码非常混乱，

用 **servlet** 来管理网站变成一件很困难的事情。为了克服 **servlet** 的缺点, **SUN** 公司用 **Java Servlet** 作为基础, 推出了 **Java Server Page**。 **JSP** 提供了 **servlet** 的几乎所有好处, 当一个客户请求一个 **JSP** 页面时, **JSP** 引擎根据 **JSP** 页面生成一个 **Java** 文件, 即一个 **servlet**。用 **JSP** 支持 **javabeans** 这一特点, 可以有效地管理页面的逻辑部分和页面的输出部分(见第 6 章)。另外, **JSP** 也可以和 **servlet** 有效地结合, 分离页面的逻辑部分和页面的输出部分 (见第 9 章)

第2章 JSP 语法

2.1 JSP 页面的基本结构

在传统的 **HTML** 页面文件中加入 **java** 程序片和 **JSP** 标签就构成了一个 **JSP** 页面文件。

一个 **JSP** 页面可由 5 种元素组合而成：

- (1) 普通的 **HTML** 标记符。
- (2) **JSP** 标签：如，指令标签、动作标签。
- (3) 变量和方法的声明。
- (4) **Java** 程序片。
- (5) **Java** 表达式。

我们称 (3)、(4)、(5) 形成的部分为 **JSP** 的脚本部分。

当服务器上的一个 **jsp** 页面被第一次请求执行时，服务器上的 **JSP** 引擎首先将 **JSP** 页面文件转译成一个 **java** 文件，再将这个 **java** 文件编译生成字节码文件，然后通过执行字节码文件响应客户的请求。这个字节码文件的任务就是：

- l 把 **JSP** 页面中普通的 **HTML** 标记符号，交给客户的浏览器执行显示。
- l **JSP** 标签、数据和方法声明、**Java** 程序片由服务器负责执行，将需要显示的结果发送给客户的浏览器。
- l **Java** 表达式由服务器负责计算，并将结果转化为字符串，然后交给客户的浏览器负责显示。

在下面的例子 1 中，客户通过表单向服务器提交三角形三边的长度，服务器将计算三角形的面积，并将计算的结果以及客户输入的三边长度返回给客户。为了讲解方便，下面的 **JSP** 文件加入了行号，它们并不是 **JSP** 源文件的组成部分。

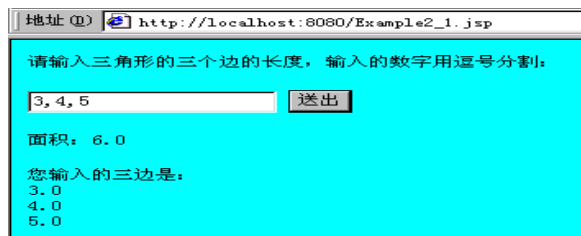


图 2.1 计算三角形面积

在下面的 Example2_1.jsp 中：

- ❏ (1)、(2) 行是 JSP 指令标签。

- ❏ (3) 至 (10) 行是 HTML 标记，其中 (7) 到 (10) 行是 HTML 表单，客户通过该表单向服务器提交数据。

- ❏ (11) 至 (13) 行是数据声明部分，该部分声明的数据在整个 JSP 页面内有效。

(14) 至 (42) 行是 Java 程序片，该程序片负责计算面积，并将结果返回给客户。该程序片内声明的变量只在该程序片内有效。

- ❏ (45)、(47)、(49) 行是 Java 表达式

例子 1（效果如图 2.1 所示）

Example2_1.jsp：

(1) `<%@ page contentType="text/html; charset=GB2312" %>`

(2) `<%@ page import="java.util.*" %>`

(3) `<HTML>`

(4) `<BODY BGCOLOR=cyan>`

- (5) **<P>** 请输入三角形的三个边的长度，输入的数字用逗号分割：
- (6) **
**
- (7) **<FORM action="Example2_1.jsp" method=post name=form>**
- (8) **<INPUT type="text" name="boy">**
- (9) **<INPUT TYPE="submit" value=" 送出" name=submit>**
- (10) **</FORM>**
- (11) **<%! double a[]=new double[3];**
- (12) **String answer=null;**
- (13) **%>**
- (14) **<% int i=0;**
- (15) **boolean b=true;**
- (16) **String s=null;**
- (17) **double result=0;**
- (18) **double a[]=new double[3];**
- (19) **String answer=null;**
- (20) **s=request.getParameter("boy");**
- (21) **if(s!=null)**
- (22) **{ StringTokenizer fenxi=new StringTokenizer(s,"** , **");**
- (23) **while(fenxi.hasMoreTokens())**
- (24) **{ String temp=fenxi.nextToken();**
- (25) **try{ a[i]=Double.valueOf(temp).doubleValue();**
- (26) **i++;**
- (27) **}**

```

(28)         catch(NumberFormatException e)
(29)         {out.print("<BR>"+"                请输入数字字符");
(30)         }
(31)     }
(32)
    if(a[0]+a[1]>a[2]&&a[0]+a[2]>a[1]&&a[1]+a[2]>a[0]&&b==true)
(33)     { double p=(a[0]+a[1]+a[2])/2;
(34)         result=Math.sqrt(p*(p-a[0])*(p-a[1])*(p-a[2]));
(35)         out.print("                面积: "+result);
(36)     }
(37)     else
(38)     {answer="                您输入的三边不能构成一个三角形";
(39)         out.print("<BR>" +answer);
(40)     }
(41) }
(42) %>
(43) <P>     您输入的三边是:
(44)     <BR>
(45)     <%=a[0]%>
(46)     <BR>
(47)     <%=a[1]%>
(48)     <BR>
(49)     <%=a[2]%>

```

(50) </BODY>

(51) </HTML>

2.2 变量和方法的声明

在“<%!”和“%>”标记符号之间声明变量和方法。

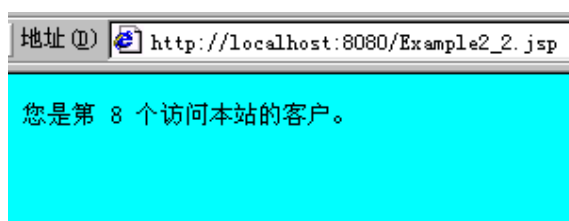
2.2.1 声明变量

在“<%!”和“%>”标记符之间声明变量，即在“<%!”和“%>”之间放置 Java 的变量声明语句，变量的类型可以是 Java 语言允许的任何数据类型，我们将这些变量称为 JSP 页面的成员变量。如，

```
<%! int a, b=10, c;  
  
String tom=null, jerry= "love JSP";  
  
Date date;  
  
%>
```

“<%!”和“%>”之间声明的变量在整个 JSP 页面内都有效，因为 JSP 引擎将 JSP 页面转译成 Java 文件时，将这些变量作为类的成员变量。这些变量的内存空间直到服务器关闭才释放。当多个客户请求一个 JSP 页面时，JSP 引擎为每个客户启动一个线程，这些线程由 JSP 引擎服务器来管理，这些线程共享 JSP 页面的成员变量，因此任何一个用户对 JSP 页面成员变量操作的结果，都会影响到其他用户。

下面的例子 2 利用成员变量被所有用户共享这一性质，实现了一个简单的计数器。



例子 2（效果如图 2.2 所示）

Example2_2.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY BGCOLOR=cyan><FONT size=1>

  <%!int i=0;

  %>

  <%i++;

  %>

  <P>您是第

    <%=i%>

    个访问本站的客户。

  </BODY>

</HTML>
```

在处理多线程问题时，我们必须注意这样一个问题：当两个或多个线程同时访问同一个共享的变量，并且一个线程需要修改这个变量时，我们应对这样的问题作出处理，否

则可能发生混乱。在上面的例子 2 中，可能发生两个客户同时请求 **Example2_2.jsp** 页面。在 **Java** 语言中我们已经知道，在处理线程同步时，可以将线程共享的变量放入一个 **synchronized** 块，或将修改该变量的方法用 **synchronized** 来修饰。这样，当一个客户用 **synchronized** 块或 **synchronized** 方法操作一个共享变量时，其它线程就必须等待，直到该线程执行完该方法或同步块。下面的例子 3 对例子 2 进行了改进。

例子 3

Example2_3.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<%! Integer number=new Integer(0);

%>

<%

    synchronized(number)

    { int i=number.intValue();

        i++;

        number=new Integer(i);

    }

%>

<P>您是第

    <%=number.intValue()%>

    个访问本站的客户。
```

</BODY>

</HTML>

2.2.2 声明方法

在“<%!”和“%>”之间声明方法，该方法在整个 JSP 页面有效，但是该方法内定义的变量只在该方法内有效。这些方法将在 Java 程序片中被调用，当方法被调用时，方法内定义的变量被分配内存，调用完毕即可释放所占的内存。当多个客户同时请求一个 JSP 页面时，他们可能使用方法操作成员变量，对这种情况应给予注意。在下面的例子 4 中，通过 synchronized 方法操作一个成员变量来实现一个计数器。

例子 4

Example2_4.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<HTML>
```

```
<BODY>
```

```
<%! int number=0;
```

```
    synchronized void countPeople()
```

```
    { number++;
```

```
    }
```

```
%>
```

```
<% countPeople(); //    在程序片中调用方法。
```

```
%>
```

```
<P><P>您是第
```

```
<%=number%>
```


个访问本站的客户。

</BODY></HTML>

在上面的例子 4 中，如果 Tomcat 服务器重新启动就会刷新计数器，因此计数又从 0 开始。在下面的例子 5 中，我们使用 Java 的输入输出流技术，将计数保存到文件。当客户访问该 JSP 页面时，就去读取这个文件，将服务器重新启动之前的计数读入，并在此基础上增 1，然后将新的计数写入到文件；如果这个文件不存在（服务器没有作过重新启动），就将计数增 1，并创建一个文件，然后将计数写入到这个文件（有关文件和输入输出流技术可参见第 4 章）。

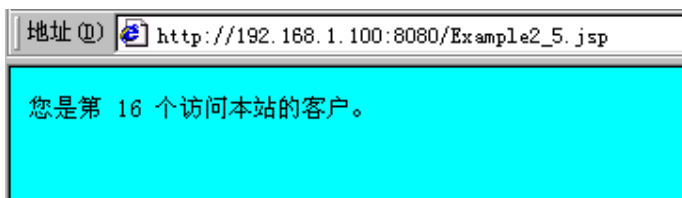


图 2.3 用文件保存计数

例子 5（效果如图 2.3 所示）

Example2_5.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import="java.io.*" %>
```

```
<HTML>
```

```
<BODY BGCOLOR=cyan><FONT Size=1>
```

```

<%! int number=0;

synchronized void countPeople()// 计算访问次数的同步方法
{
    if(number==0)
    {
        try{

            FileInputStream in=new FileInputStream("count.txt");

            DataInputStream dataIn=new DataInputStream(in);

            number=dataIn.readInt();

            number++;

            in.close();

            dataIn.close();

        }

        catch(FileNotFoundException e)

        { number++;

            try {FileOutputStream out=new
FileOutputStream("count.txt");

                DataOutputStream dataOut=new
DataOutputStream(out);

                dataOut.writeInt(number);

                out.close();dataOut.close();

            }

            catch(IOException ee){}

```

```

    }

    catch(IOException ee)

    {

    }

}

else

{number++;

try{

    FileOutputStream out=new FileOutputStream("count.txt");

    DataOutputStream dataOut=new DataOutputStream(out);

    dataOut.writeInt(number);

    out.close();dataOut.close();

}

catch(FileNotFoundException e){}

catch(IOException e){}

}

}

%>

<%

    countPeople();

%>

<P><P>您是第

    <%=number%>

```

个访问本站的客户。

<BODY>

<HTML>

数学上有一个计算p的公式： $p/4=1-1/3+1/5-1/7+1/9-1/11 \dots$ 。

下面的例子 6 中利用成员变量被所有客户共享这一特性实现客户帮助计算p的值，即每当客户访问 Example2_6.jsp 都参与了一次p的计算。

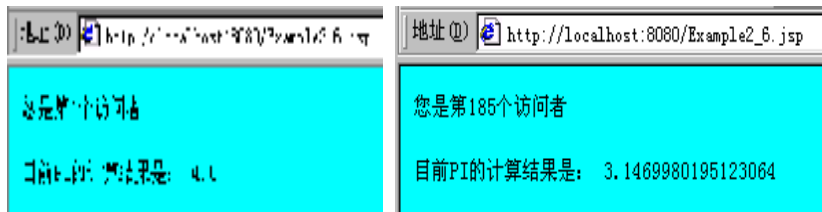


图 2.4 利用页面访问次数计算p的值

例子 6（效果如图 2.4 所示）

Example2_6.jsp:

```
<%@ page contentType="text/html; charset=GB2312" %>
```

```
<HTML>
```

```
<BODY bgcolor=cyan><FONT Size=1>
```

```
<%! double sum=0,i=1,j=1;
```

```
long number=0;
```

```
synchronized void 帮助计算 PI()
```

```

        { number++;

          sum=sum+i/j;

          j=j+2;

          i=-i;

        }

    %>

<%

    帮助计算 PI();

    %>

<P>您是第<%=number%>个访问者

<BR>

<P>目前 PI 的计算结果是:

    <%=sum*4%>

</BODY>

</HTML>

```

2.2.3 声明类

可以在“<%!”和“%>”之间声明一个类，该类在 JSP 页面内有效，即在 JSP 页面的 Java 程序片部分可以使用该类创建对象。在下面的例子 7 中，我们定义了一个 Circle 类，该类的对象负责求圆的面积和周长。当客户向服务器提交圆的半径后，该对象负责计算面积和周长。

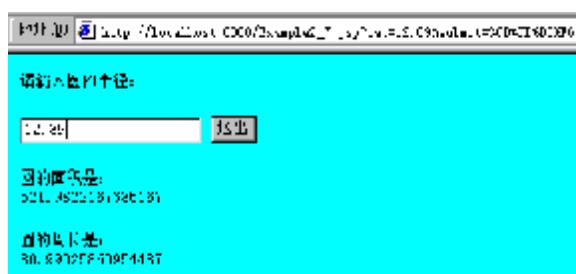


图 2.5 在 JSP 页面中使用自定义的类

例子 7（效果如图 2.5 所示）

Example2_7.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<P> 请输入圆的半径:

<BR>

<FORM action="Example2_7.jsp" method=get name=form>

    <INPUT type="text" name="cat" value="1">

    <INPUT TYPE="submit" value="    送出" name=submit>

</FORM>

<%! public class Circle

    {double r;

        Circle(double r)

        {this.r=r;

        }

        double    求面积()

        {return Math.PI*r*r;
```

```

    }

    double    求周长()

    {return Math.PI*2*r;

    }

}

%>

<% String str=request.getParameter("cat");

    double r;

    if(str!=null)

        {r=Double.valueOf(str).doubleValue();

        }

    else

        {r=1;

        }

    Circle circle=new Circle(r); //    创建对象。

%>

<P> 圆的面积是:

<BR>

    <%=circle. 求面积()%>

<P> 圆的周长是:

<BR>

    <%=circle. 求周长()%>

<HTML>

```

<BODY>

2.3 Java 程序片

可以在“<%”和“%>”之间插入 **Java** 程序片。一个 **JSP** 页面可以有許多程序片，这些程序片将被 **JSP** 引擎按顺序执行。在一个程序片中声明的变量称做 **JSP** 页面的局部变量，它们在 **JSP** 页面内的所有程序片部分以及表达式部分内都有效。这是因为 **JSP** 引擎将 **JSP** 页面转译成 **Java** 文件时，将各个程序片的这些变量作为类中某个方法的变量，即局部变量。利用程序片的这个性质，有时候可以将一个程序片分割成几个更小的程序片，然后在这些小的程序片之间再插入 **JSP** 页面的一些其它标记元素。当程序片被调用执行时，这些变量被分配内存空间，所有的程序片调用完毕，这些变量即可释放所占的内存。当多个客户请求一个 **JSP** 页面时，**JSP** 引擎为每个客户启动一个线程，一个客户的局部变量和另一个客户的局部变量被分配不同的内存空间。因此，一个客户对 **JSP** 页面局部变量操作的结果，不会影响到其它客户的这个局部变量。

下面例子 8 中的程序片负责计算 1 到 100 的连续和。

例子 8（效果如图 2.6 所示）

Example2_8.jsp

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<HTML>
```

```
<BODY bgcolor=cyan><FONT size=1>
```

```
<%!
```

```
    long continueSum(int n)
```

```
    { int sum=0;
```

```
      for(int i=1;i<=n;i++)
```

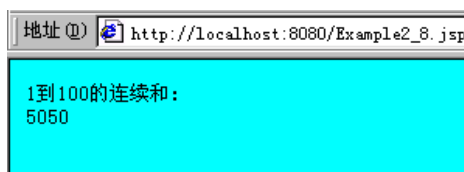


图 2.6 使用程序片计算连续和


```
{ sum=sum+i;  
  
}  
  
return sum;  
  
}
```

```
%>
```

<P>1 到 100 的连续和:


```
<% long sum;
```

```
sum=continueSum(100);
```

```
out.print(''+sum);
```

```
%>
```

</BODY>

</HTML>

下面的例子 9 将例子 8 中的程序片分割成几部分。

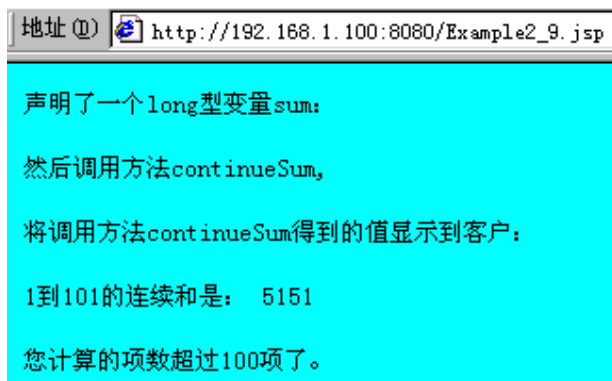


图 2.7 程序片分割

例子 9（效果如图 2.7 所示）

Example2_9.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<HTML>
```

```
<BODY bgcolor=cyan><Font size=1>
```

```
<%!
```

```
    long continueSum(int n)
```

```
    { int sum=0;
```

```
      for(int i=1;i<=n;i++)
```

```
      { sum=sum+i;
```

```
      }
```

```
      return sum;
```

```
    }
```

```
%>
```

```
<P> 声明了一个 long 型变量 sum:
```

```
<% long sum;
```

```
%>
```

```
<P> 然后调用方法 continueSum,
```

```
<% sum=continueSum(101);
```

```
%>
```

<P> 将调用方法 `continueSum` 得到的值显示到客户：

<P>1 到 101 的连续和是：

```
<%= sum %>
```

```
<% if(sum>=5050)
```

```
{%>
```

<P> 您计算的项数超过 100 项了。

```
<%}
```

```
else
```

```
{%>
```

<P> 您计算的项数没有超过 100 项。

```
<%
```

```
}
```

```
%>
```

```
</Font>
```

```
</BODY>
```

```
</HTML>
```

下面例子 10 的程序片负责读取服务器上的一个文件，并将文件的内容显示给客户。

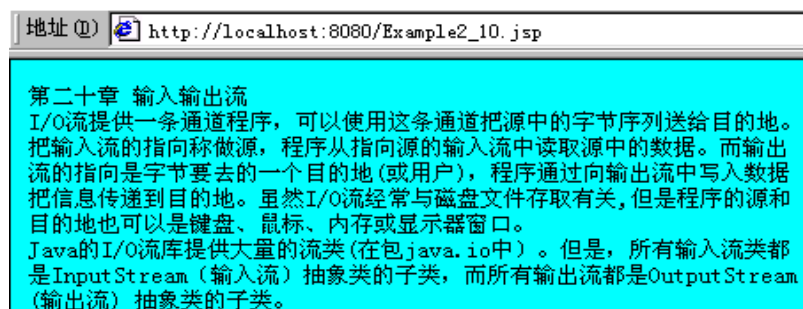


图 2.8 使用程序片读取文件

例子 10（效果如图 2.8 所示）

Example2_10.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.io.*" %>

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.io.*" %>

<HTML>

<BODY bgcolor=cyan><FONT size=1>

    <% try{

        File f=new File("D:/test","A.txt");

        FileReader in=new FileReader(f);

        BufferedReader buffer=new BufferedReader(in);

        String s=null;

        while((s=buffer.readLine())!=null)

            { out.print(s+"<BR>");

              }

        in.close();
```

```

        buffer.close();
    }

    catch(IOException ee)

        {out.print("        文件不存在");}

    %>

</BODY>

</HTML>

```

2.4 表达式

可以在“<%=”和“%>”之间插入一个表达式（注意：不可插入语句，“<%=”是一个完整的符号，“<%=”和“=”之间不要有空格），这个表达式必须能求值。表达式的值由服务器负责计算，并将计算结果用字符串形式发送到客户端显示。

下面的例子 11 计算表达式的值。

例子 11（效果如图 2.9 所示）

Example2_11.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><FONT size=1>

<P> Sin(0.9)除以 3 等于

    <%=Math.sin(0.90)/3%>

<p>3 的平方是:

    <%=Math.pow(3,2)%>

<P>12345679 乘 72 等于

```

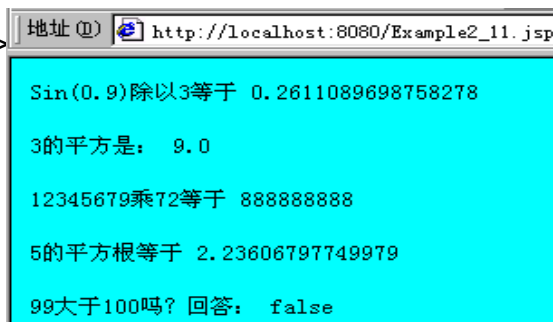


图 2.9 计算表达式的值

`<%=12345679*72%>`

`<P> 5 的平方根等于`

`<%=Math.sqrt(5)%>`

`<P>99 大于 100 吗? 回答:`

`<%=99>100%>`

`</BODY>`

`</HTML>`

2.5 JSP 中的注释

注释可以增强 JSP 文件的可读性,并易于 JSP 文件的维护。JSP 中的注释可分为两种

(1) HTML 注释:在标记符号“`<!--`”和“`-->`”之间加入注释内容:

`<!-- 注释内容 -->`

JSP 引擎把 HTML 注释交给客户,因此客户通过浏览器查看 JSP 的源文件时,能够看到 HTML 注释。

(2) JSP 注释:在标记符号“`<%--`”和“`--%>`”之间加入注释内容:

`<%-- 注释内容 --%>`

JSP 引擎忽略 JSP 注释,即在编译 JSP 页面时忽略 JSP 注释。

例子 12

Example2_12.jsp:

`<%@ page contentType="text/html;charset=GB2312" %>`

`<HTML>`

`<BODY>`

`<P> 请输入三角形的三个边 a,b,c 的长度:`

**
**

<!-- 以下是 HTML 表单，向服务器发送三角形的三个边的长度 -->

<FORM action="Example2_12.jsp" method=post name=form>

<P> 请输入三角形边 a 的长度:

<INPUT type="text" name="a">

**
**

<P> 请输入三角形边 b 的长度:

<INPUT type="text" name="b">

**
**

<P> 请输入三角形边 c 的长度:

<INPUT type="text" name="c">

**
**

<INPUT TYPE="submit" value=" 送出" name=submit>

</FORM>

<%-- 获取客户提交的数据 --%>

<% String string_a=request.getParameter("a"),

string_b=request.getParameter("b"),

string_c=request.getParameter("c");

double a=0,b=0,c=0;

%>

<%-- 判断字符串是否是空对象,如果是空对象就初始化--%>

<% if(string_a==null)

{string_a="0";string_b="0";string_c="0";

```

    }

    %>

<%-- 求出边长，并计算面积--%>

    <% try{ a=Double.valueOf(string_a).doubleValue();

        b=Double.valueOf(string_b).doubleValue();

        c=Double.valueOf(string_c).doubleValue();

        if(a+b>c&&a+c>b&&b+c>a)

            {double p=(a+b+c)/2.0;

                double mianji=Math.sqrt(p*(p-a)*(p-b)*(p-c));

                out.print("<BR>"+ "          三角形面积: "+mianji);

            }

        else

            {out.print("<BR>"+ "          您输入的三边不能构成一个三角形");

            }

        }

    catch(NumberFormatException e)

        {out.print("<BR>"+ "          请输入数字字符");

        }

    %>

</BODY>

</HTML>

```


2.6 JSP 指令标签

2.6.1 page 指令

page 指令用来定义整个 JSP 页面的一些属性和这些属性的值。例如,我们可以用 **page** 指令定义 JSP 页面的 **contentType** 属性的值是 "text/html;charset=GB2312", 这样, 我们的页面就可以显示标准汉语。如,

```
<%@ page contentType="text/html;charset=GB2312" %>
```

page 指令的格式:

```
<%@ page 属性 1=“属性 1 的值” 属性 2=“属性 2 的值” .....%>
```

属性值总是用单引号或引号双号括起来, 例如:

```
<%@ page contentType="text/html;charset=GB2312" import="java.util.*" %>
```

如果为一个属性指定几个值的话, 这些值用逗号分割。**page** 指令只能给 **import** 属性指定多个值; 其它属性只能指定一个值。

例如:

```
<%@ page import="java.util.*", "java.io.*", "java.awt.*" %>
```

当你为 **import** 指定多个属性值时, JSP 引擎把 JSP 页面转译成的 java 文件中会有如下的 **import** 语句:

```
import java.util.*;
```

```
import java.io.*;
```

```
import java.awt.*;
```

在一个 JSP 页面中，也可以使用多个 **page** 指令来指定属性及其值。需要注意的是：可以使用多个 **page** 指令给属性 **import** 几个值，但其它属性只能使用一次 **page** 指令指定该属性一个值。如，

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import="java.util.*" %>
```

```
<%@ page import="java.util.*", "java.awt.*" %>
```

注：下列用法是错误的

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page contentType="text/html;charset=GB2312" %>
```

尽管指定的属性值相同，也不允许 2 次使用 Page 给 contentType 属性指定属性值，

注：page 指令的作用对整个页面有效，与其书写的位置无关，但习惯把 page 指令写在 JSP 页面的最前面。

I language 属性

定义 JSP 页面使用的脚本语言，该属性的值目前只能取 **"java"**。

为 **language** 属性指定值的格式：

```
<%@ page language="java" %>
```

language 属性的默认值是"java"，即如果你在 JSP 页面中没有使用 **page** 指令指定该属性的值的话，那么，JSP 页面默认有如下 **page** 指令：

```
<%@ page language="java" %>
```

I import 属性：

该属性的作用是为 JSP 页面引入 Java 核心包中的类，这样就可以在 JSP 页面的程序片部分、变量及函数声明部分、表达式部分使用包中的类。可以为该属性指定多个值，该属性的值可以是 Java 某包中的所有类或一个具体的类，例如：

```
<%@ page import="java.io.*", "java.util.Date" %>
```

JSP 页面默认 **import** 属性已经有如下的值：

"java.lang.*"、"javax.servlet.*"、"javax.servlet.jsp.*"、"javax.servlet.http.*"。

I contentType 属性：

定义 JSP 页面响应的 MIME(Multipurpose Internet Mail Extention) 类型和 JSP 页面字符的编码。属性值的一般形式是：“MIME 类型”或 “MIME 类型;charset=编码”，如：

<%@ page contentType="text/html;charset=GB2312" %>

contentType 属性的默认值是 "text/html ; charset=ISO-8859-1"。

I session 属性:

用于设置是否需要使用内置的 **session** 对象。

session 的属性值可以是 **true** 或 **false**，**session** 属性默认的属性值是 **true**。

I buffer 属性:

内置输出流对象 **out** 负责将服务器的某些信息或运行结果发送到客户端显示，**buffer** 属性用来指定 **out** 设置的缓冲区的大小或不使用缓冲区。

buffer 属性可以取值 “none”，设置 **out** 不使用缓冲区。**Buffer** 属性的默认值是 **8kb**。

例如:

<%@ page buffer= “24kb” %>

I auotFlush 属性

指定 **out** 的缓冲区被填满时，缓冲区是否自动刷新。

auotFlush 可以取值 **true** 或 **false**。**auotFlush** 属性的默认值是 **true**。当 **auotFlush** 属性取值 **false** 时,如果 **out** 的缓冲区填满时,就会出现缓存溢出异常。当 **buffer** 的值是 “none” 时，**auotFlush** 的值就不能设置成 **false**。

I isThreadSafe 属性

用来设置 **JSP** 页面是否可多线程访问。

isThreadSafe 的属性值取 **true** 或 **false**。当 **isThreadSafe** 属性值设置为 **true** 时，**JSP**

页面能同时响应多个客户的请求；当 **isThreadSafe** 属性值设置成 **false** 时，JSP 页面同一时刻只能处理响应一个客户的请求，其他客户需排队等待。**isThreadSafe** 属性的默认值是 **true**。

I info 属性：

该属性为 JSP 页面准备一个字符串，属性值是某个字符串。例如，

```
<%@ page info= "we are students" %>
```

可以在 JSP 页面中使用方法：

```
getServletInfo();
```

获取 **info** 属性的属性值。

注：当 JSP 页面被转译成 Java 文件时，转译成的类是 **Servlet** 的一个子类，所以在 JSP 页面中可以使用 **Servlet** 类的方法：**getServletInfo()**。

下面的例子 13 使用 **getServletInfo()** 方法获取 **info** 的属性值。

例子 13（效果如图 2.10 所示）

Example2_13.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page info="我！张惠妹" %>
```

```
<HTML>
```

```
<BODY bgcolor=cyan><FONT Si
```

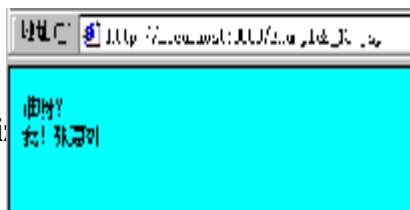


图 2.10 获取 **info** 属性值

<P> 谁呀？

```
<% String s=getServletInfo();
```

```
    out.print("<BR>" + s);
```

```
%>
```

</BODY>

<HTML>

2.6.2 include 指令标签

如果需要在 **JSP** 页面内某处整体嵌入一个文件，就可以考虑使用这个指令标签。该指令标签语法如下：

```
<%@ include file= “文件的名字” %>
```

该指令标签的作用是在 **JSP** 页面出现该指令的位置处，静态插入一个文件。被插入的文件必须是可访问和可使用的，即该文件必须和当前 **JSP** 页面在同一 **Web** 服务目录中。所谓静态插入，就是当前 **JSP** 页面和插入的部分合并成一个新的 **JSP** 页面，然后 **JSP** 引擎再将这个新的 **JSP** 页面转译成 **Java** 类文件。因此，插入文件后，必须保证新合并成的 **JSP** 页面符合 **JSP** 语法规则，即能够成为一个 **JSP** 页面文件。比如，如果一个 **JSP** 页面使用 **include** 指令插入另一个 **JSP** 文件，被插入的这个 **JSP** 页面中有一个设置页面 **contentType** 属性的 **page** 指令：

```
<%@ page contentType="text/html;charset=GB2312" %>,
```

而当前 **JSP** 页面已经使用 **page** 指令设置了 **contentType** 的属性值，那么新合并的 **JSP** 页

面就出现了语法错误，当转译合并的 JSP 页面到 Java 文件时就会失败。

下面的例子 14 在 JSP 页面静态插入一个文本文件：**Hello.txt**，该文本文件的内容是：“你们好,很高兴认识你们呀!”。该文本文件必须和当前 JSP 页面在同一 Web 服务目录中。

例子 14（效果如图 2.11 所示）

Example2_14.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<html>
```

```
<BODY bgcolor=cyan>
```

```
<H3>
```

```
<%@ include file="Hello.tx
```

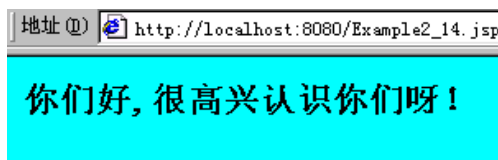


图 2.11 使用 include 指令标签静态嵌入文本文件

```
</H3>
```

```
</BODY>
```

```
</HTML>
```

注：上述 Example2_14.jsp 等价于下面的 JSP 文件：Example2_14_1.jsp。

Example2_14_1.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<html>
```

```
<BODY>
```

```
<H3>
```

你们好，很高兴认识你们呀！

</H3>

</BODY>

</HTML>

注：在 Example2_14.jsp 被转译成 Java 文件后，如果你对插入的文件 Hello.txt 进行了修改，那么必须要重新将 Example2_14.jsp 转译成 java 文件(重新保存页面，然后再访问该页面即可)，否则只能看到修改前的 Hello.txt 的内容。

下面的例子 15 在 JSP 页面 Example2_15.jsp 中静态插入一个 JSP 文件：Computer.jsp。

Computer.jsp:

<FORM action="" method=post name=form>

<INPUT type="text" name="ok">

**
<INPUT TYPE="submit" value="送出" name=submit>**

</FORM>

<% String a=request.getParameter("ok");

if(a==null)

{ a="1";

}

try{

double number=Integer.parseInt(a);

**out.print("
" + Math.sqrt(number));**

}

catch(NumberFormatException e)

{


```

        out.print("<BR>"+"      请输入数字字符");
    }

%>

```

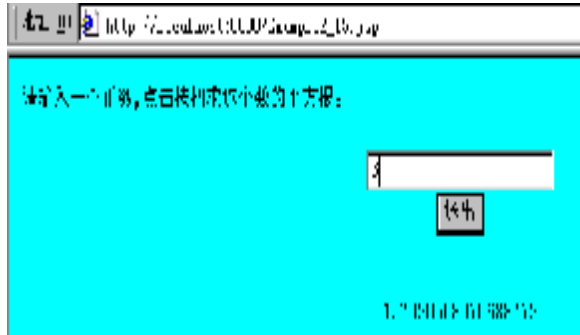


图 2.12 使用 include 指令标签静态嵌入 JSP 文件

例子 15（效果如图 2.12 所示）

Example2_15.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<html>

<BODY Bgcolor=cyan><FONT size=1>

<P>请输入一个正数,点击按钮求这个数的平方根。

<CENTER>

<%@ include file="Computer.jsp"%>

</CENTER>

```

</BODY>

</HTML>

注：Example2_15.jsp 等同于下述 JSP 文件：Example2_15_1.jsp 。

Example2_15_1.jsp:

<%@ page contentType="text/html;charset=GB2312" %>

<html>

<BODY Bgcolor=cyan>

<P>请输入一个正数,点击按钮求这个数的平方根。

<CENTER>

<FORM action="" method=post name=form>

<INPUT type="text" name="ok">

**
<INPUT TYPE="submit" value="送出" name=submit>**

</FORM>

<% String a=request.getParameter("ok");

if(a==null)

{ a="1";

}

try{

double number=Integer.parseInt(a);

**out.print("
" + Math.sqrt(number));**

}

catch(NumberFormatException e)

```

    {
        out.print("<BR>"+ "    请输入数字字符");
    }

%>

</CENTER>

</BODY>

</HTML>

```

2.7 JSP 动作标签

动作标签是一种特殊的标签，它影响 **JSP** 运行时的功能。

2.7.1 include 动作标签

include 动作标签：

```
<jsp:include page= “文件的名字” / >
```

或

```
<jsp:include page= “文件的名字”>
```

```
</jsp:include>
```

该动作标签告诉 **JSP** 页面动态包含一个文件，即 **JSP** 页面运行时才将文件加入。与静态插入文件的 **include** 指令标签不同，当 **JSP** 引擎把 **JSP** 页面转译成 **Java** 文件时，不把 **JSP** 页面中动作指令 **include** 所包含的文件与原 **JSP** 页面合并一个新的 **JSP** 页面，而是告诉 **Java** 解释器，这个文件在 **JSP** 运行时（**Java** 文件的字节码文件被加载执行）才包含进来。如果包含的文件是普通的文本文件，就将文件的内容发送到客户端，由客户

端负责显示；如果包含的文件是 **jsp** 文件，**JSP** 引擎就执行这个文件，然后将执行的结果发送到客户端，并由客户端负责显示这些结果。

注：**include** 动作标签与静态插入文件的 **include** 指令标签有很大的不同，动作标签是在执行时才对包含的文件进行处理，因此 **JSP** 页面和它所包含的文件在逻辑和语法上是独立的；如果你对包含的文件进行了修改，那么运行时看到所包含文件修改后的结果，而静态 **include** 指令包含的文件如果发生了变化，我们必须重新将 **JSP** 页面转译成 **java** 文件（可将该 **JSP** 页面重新保存，然后再访问，就可产生新的转译 **Java** 文件），否则只能看到所包含的修改前的文件内容。

注：书写 **include** 动作标签：`<jsp:include page/>`时要注意：“**jsp**”、“**:**”、“**include**”三者之间不要有空格。

下面的例子 16 动态包含两个文件：**image.html** 和 **Hello.txt**。我们把 **Example2_16.jsp** 页面保存到 **Tomcat\Jakarta-tomcat-4.0\webapps\root**，在 **root** 下又新建了一个文件夹 **Myfile**，**Hello.txt** 存放在 **MyFile** 文件夹中，**image.html** 存放在 **root** 下

Hello.txt:

```
<H4>你好，祝学习进步！  
<BR>学习 JSP 要有 Java 语言的  
<BR>要认真学习 JSP 的基本语  
</H4>
```



图 2.13 用 **include** 动态标签加载文件

image.html:

```
<image src="tom1.jpg">
```

例子 16（效果如图 2.13 所示）

Example2_16.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY BGCOLOR=Cyan><FONT Size=1>

<P>加载的文件:

    <jsp:include page="Myfile/Hello.txt">

    </jsp:include>

<P>加载的图象:

<BR>

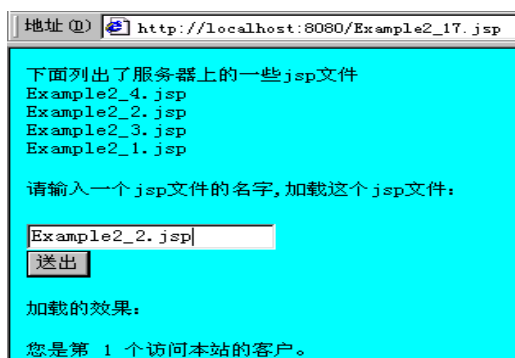
    <jsp:include page="image.html">

    </jsp:include>

</BODY>

</HTML>
```

由于动作指令 **include** 是动态地包含一个文件, 因此客户可以通过 **HTML** 表单提交需要包含的文件的 名字。在下面的例子 17 中, 我们用 **Java** 语言中的 **File** 类列出了 **Tomcat\Jakarta-tomcat-4.0\webapps\root\Myfile** 下的所有的 **jsp** 文件, 客户通过表单, 提交要加载的文件的全名。



例子 17（效果如图 2.14 所示）

Example2_17.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import ="java.io.*" %>
```

```
<%! class FileJSP implements FilenameFilter
```

```
{ String str=null;
```

```
    FileJSP(String s)
```

```
    {str="."+s;
```

```
    }
```

```
    public boolean accept(File dir,String name)
```

```
    { return name.endsWith(str);
```

```
    }
```

```
}
```

```
%>
```

```
<BODY bgcolor=cyan><FONT Size=1>
```

<P>下面列出了服务器上的一些 jsp 文件

```
<% File dir=new File("d:/Tomcat/Jakarta-tomcat-4.0/webapps/root/Myfile");  
  
FileJSP file_jsp=new FileJSP("jsp");  
  
String file_name[]=dir.list(file_jsp);  
  
for(int i=0;i<file_name.length;i++)  
  
    {out.print("<BR>" +file_name[i]);  
  
    }  
  
%>
```

<P> 请输入一个 jsp 文件的 名字,加载这个 jsp 文件:

```
<FORM action="Example2_17.jsp" method=post name=form>  
  
<INPUT type="text" name="ok">  
  
    <BR>  
  
    <INPUT TYPE="submit" value="    送出" name=submit>  
  
</FORM>
```

```
<% String fileName="/Myfile/";  
  
    fileName=fileName+request.getParameter("ok");  
  
%>
```

<P>加载的效果:

```
<jsp:include page="<%= fileName %>" >  
  
</jsp:include>  
  
</BODY>
```

2.7.2 param 动作标签

param 标签以“名字—值”对的形式为其它标签提供附加信息，这个标签与 **jsp:include**、**jsp:forward**、**jsp:plugin** 标签一起使用。

param 动作标签：

```
<jsp:param name= “名字” value= “指定给 param 的值”>
```

当该标签与 **jsp:include** 标签一起使用时，可以将 **param** 标签中的值传递到 **include** 指令要加载的文件中去，因此 **include** 动作标签如果结合 **param** 标签，可以在加载文件的过程中向该文件提供信息。下面例子 18 动态包含文件：**tom.jsp**，当该文件被加载时获取 **param** 标签中 **computer** 的值(获取 **computer** 的值由 JSP 的内置对象 **request** 调用 **getParameter** 方法完成)。

tom.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

    <% String str=request.getParameter("computer");//    获取值。

        int n=Integer.parseInt(str);

        int sum=0;

        for(int i=1;i<=n;i++)

            { sum=sum+i;

              }

    %>
```


<P>

从 1 到<%=n%>的连续和是：

**
**

<%=sum%>

</BODY>

</HTML>

例子 18（效果如图 2.15 所示）

Example2_18.jsp:

<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<P>加载文件效果：

<jsp:include page="tom.jsp">

<jsp:param name="computer" value="300" />

</jsp:include>

</BODY>

</HTML>

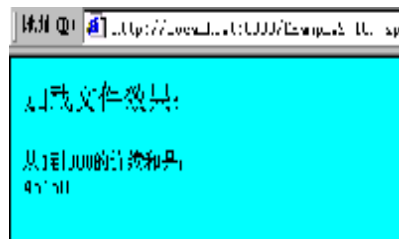


图 2.15 用 param 标签传递参数

2.7.3 forward 动作标签

forward 指令：

<jsp:forward page="要转向的页面" >

</jsp:forward>

或

```
<jsp:forward page="要转向的页面" />
```

该指令的作用是：从该指令处停止当前页面的继续执行，而转向其它的一个 JSP 页面。

假设 Example2_11.jsp 存放在 d:/Tomcat/Jakarta-tomcat-4.0/webapps/root/Myfile 下；
Example2_2 存放在 d:/Tomcat/Jakarta-tomcat-4.0/webapps/root/ 下。下面的
Example2_19.jsp 存放在 d:/Tomcat/Jakarta-tomcat-4.0/webapps/root/下。

在下面的 JSP 页面中，首先随机获取一个数，如果该数大于 0.5 就转向页面：
Example2_11.jsp；否则转向页面：Example2_2.jsp。下图 2.16 是 Example2_19.jsp 运行
几次后，随机出现的两种运行效果。

例子 19（效果如图 2.16 所示）

Example2_19.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<HTML>
```

```
<BODY>
```

```
<% double i=Math.random();
```

```
    if(i>0.5)
```

```
    {
```

```
%>
```

```
<jsp:forward page="/Myfile/Exa
```

```
</jsp:forward>
```

```
<%
```

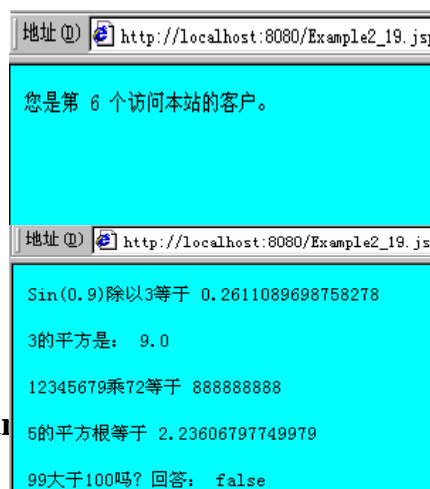


图 2.16 使用 forward 指令进行页面转向

```

    }

    else

    {

%>

    <jsp:forward page="Example2_2.jsp" >

    </jsp:forward>

<%

    }

%>

```

<P> 这句话和下面的表达式的值能输出吗？

```

<%=i%>

</BODY>

</HTML>

```

该指令也可以结合 **param** 指令，向要转到的页面传送信息。下面的 **Example2_20.jsp** 页面转到 **come.jsp**，并向转到的 **come.jsp** 页面传递一个数值，如图 2.17 所示。

come.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><FONT Size=1>

<%String str=request.getParameter("number");

    double n=Double.parseDouble(str);

%>

```

```

<P> 您传过来的数值是:<BR>

<%=n%>

</BODY>

</HTML>

```



图 2.17 用 param 标签传值到转向页面

例子 20（效果如图 2.17 所示）

Example2_20.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<% double i=Math.random();

%>

<jsp:forward page="come.jsp" >

    <jsp:param name="number" value="<%=i%>" />

</jsp:forward>

</BODY>

</HTML>

```

2.7.4 plugin 动作标签

在页面中使用普通的 HTML 标记<apple></apple>可以让客户下载运行一个 java applet 小应用程序，但并不是所有的客户的浏览器都支持 java apple 小程序，如果你的小应用程序使用了 JDK1.2 以后的类，那么，目前的 IE5.5 并不支持这个 Java 小应用程序。而使用 plugin 动作标签可以保证客户能执行你的小应用程序。

该动作标签指示 JSP 页面加载 java plugin，该插件由客户负责下载，并使用该插件来运行 Java applet。

Plugin 动作标签：

```
<jsp:plugin type="applet" code="小应用程序的字节码文件"
```

```
    jreversion="java 虚拟机版本号" width="小程序宽度值" height="小程序高度值" >
```

```
<jsp:fallback>
```

提示信息：用来提示用户的浏览器是否支持插件下载

```
</jsp:fallback>
```

```
</jsp:plugin>
```

假设有一个 java applet 小程序，主类字节码文件是：B.class，该文件存放在 Root 下。含有 plugin 标签的 JSP 文件 Example2_21.jsp 也存放在 Root 下。

例子 21

Example2_21.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

<HTML>

<BODY>

```
<jsp:plugin type="applet" code="B.class" jreversion="1.2" width="200"
height="260" >
```

```
<jsp:fallback>
```

Plugin tag OBJECT or EMBED not supported by browser.

```
</jsp:fallback>
```

```
</jsp:plugin>
```

```
</body></html>
```

当客户访问上述 JSP 页面时，将导致登录 sun 公司的网站下载 Java plugin，出现客户选择是否下载插件的界面，如图 2.18 所示。



图 2.18 客户选择是否下载插件的界面

选择“是”，再选择同意下载，出现下载插件的界面，如图 2.19 所示。

加载执行 **java applet** 小应用程序了，如图 2.21 所示（以后客户再访问带有 **plugin** 标签的 **JSP** 页就能直接执行该页面中包含的 **java applet** 小应用程序了）。



图 2.21 用插件运行的 **java applet** 效果

上面 **JSP** 页面中包含的 **java** 小应用程序是完全用 **java swing** 编写的，使用表格 **JTable** 组件技术，来计算 2 阶行列式的小程序。如果不使用插件标签，目前的浏览器，比如 **IE5.5**，都无法运行。以下是 **java applet** 小应用程序的源代码：**B.java**，该代码摘自《**Java 2 实用教程**》清华大学出版社（耿祥义等编写）。

B.java:

```
import javax.swing.*;import java.awt.*;  
  
import java.awt.event.*;  
  
public class B extends JApplet implements ActionListener
```



```

{  JTable table;Object a[][];

JPanel p1,p2;

    Object name[]={ "   第 1 列","第 2 列"};

    JButton button;JTextField text;

public void init()

{  a=new Object[2][2];

    button=new JButton("   确定");text=new JTextField(8);

    p1=new JPanel();p2=new JPanel();

    p1.setLayout(new GridLayout(2,1));

    p1.add(new Label("   输入 2 阶行列式的元素"));

    p1.add(new Label("   输入或修改数据后，用鼠标点击每个格，使数据生效"));

    p2.add(button);

    p2.add(new JLabel("   结果:"));

    p2.add(text);

    table=new JTable(a,name);

    button.addActionListener(this);

    getContentPane().add(new JScrollPane(table),BorderLayout.CENTER);

    getContentPane().add(p1,BorderLayout.NORTH);

    getContentPane().add(p2,BorderLayout.SOUTH);

}

public void actionPerformed(ActionEvent e)

{if(e.getSource()==button)

    { double d[][]=new double[2][2];

```

```

    double result;

    for(int i=0;i<2;i++)
    {for(int j=0;j<2;j++)
        {d[i][j]=Double.valueOf(a[i][j].toString()).doubleValue();
        }
    }

    result=d[1][1]*d[0][0]-d[0][1]*d[1][0];

    text.setText(String.valueOf(result));

}

}
}

```

注：如果 JSP 页面和 java applet 小程序不在同一文件下，plugin 标签中应增加 codebase 选项，指出小程序所在的位置。比如，如果上述例子的 B.class 存放在 Root 的 Myfile 文件下。那么，Plugin 标签：

```

<jsp:plugin type="applet" code="B.class" codebase="/Myfile"
    jreversion="1.2" width="300" height="260" >
    <jsp:fallback>
        Plugin tag OBJECT or EMBED not supported by browser.
    </jsp:fallback>
</jsp:plugin>

```

注：另外，在 plugin 标签中还可以增加如下的选项，以便控制 java applet 的位置

align: 取值是 “bottom”、“top”、“middle”、“left” 或 “right”。例如, Align 取值是 “left” 时, 小程序在页面的左面, 页面的其它部分在右面。Hspace: 指定 java applet 与左右对象, 比如文字图象等之间的距离。Vspace: 指定 java applet 与上下对象, 比如文字图象等之间的距离。

比如:

```
<jsp:plugin type="applet" code="B.class"
  jreversion="1.2" width="300" height="260" align="left" vspace="60" >
  <jsp:fallback>
    Plugin tag OBJECT or EMBED not supported by browser.
  </jsp:fallback>
</jsp:plugin>
```

2.7.5 useBean 动作标签

该标签用来创建并使用一个 java beans, 是非常重要的一个动作标签, 我们将在第 6 章详细讨论。Sun 公司的倡导是: 用 HTML 完成 JSP 页面的静态部分, 用 javabeans 完成动态部分, 实现真正意义上的静态和动态分割。

第3章 JSP 内置对象

有些对象不用声明就可以在 JSP 页面的脚本部分使用，这就是 JSP 的内置对象。

JSP 的内置对象有：**request**、**response**、**session**、**application**、**out**。以下我们将一一介绍。

response 和 **request** 对象是 JSP 的内置对象较重要的两个，这两个对象提供了对服务器和浏览器通信方法的控制。直接讨论这两个对象前，要先对 HTTP 协议----Word Wide Web 底层协议作简单介绍。

Word Wide Web 是怎样运行的呢？在浏览器上键入一个正确的网址后，若一切顺利，网页就出现了。例如，在浏览器输入栏中键入 <http://www.yahoo.com>，Yahoo 网站的主页就出现在浏览器窗口。这背后是什么在起作用？

使用浏览器从网站获取 HTML 页面时，实际在使用 Hypertext Transfer Protocol(HTTP)。HTTP 协议规定了信息在 Internet 上的传输方法，特别规定了浏览器与服务器的交互方法。

从网站获取页面时，浏览器在网站上打开了一个对网络服务器的连接，并发出请求。服务器收到请求后回应，所以 HTTP 协议被称作“请求和响应”协议。

浏览器请求有某种结构，HTTP 请求包括一个请求行、头域和可能的信息体。最普通的请求类型是对页面的一个简单请求，如下例：

GET/hello.htm HTTP/1.1

Host:www.sina.com.cn

这是对网站：www.sina.com.cn 上页面 **hello.htm** 的 HTTP 请求的例子。首行是请求行，规定了请求的方法、请求的资源及使用的 HTTP 协议的版本。

上例中，请求的方法是“GET”方法，此方法获取特定的资源。上例中 GET 方法用来获取名为 **hello.htm** 的网页。其它请求方法包括 **POST**，**HEAD**，**DELETE**，**TRACE** 及 **PUT** 方法等。

此例中的第二行是头(header)。**Host** 头规定了网站上 **hello.htm** 文件的 Internet 地址。此例中，主机是：www.sina.com.cn。

一个典型请求通常包含许多头，称做请求的 **HTTP** 头。头提供了关于信息体的附加信息及请求的来源。其中有些头是标准的，有些和特定的浏览器有关。

一个请求还可能包含信息体，例如，信息体可包含 **HTML** 表单的内容。在 **HTML** 表单上单击 **Submit** 键时，该表单使用 **ACTION=“POST”**或 **ACTION=“GET”**特征，输入表单的内容都被发送到服务器上。该表单内容就由 **POST** 方法或 **GET** 方法在请求的信息体中发送。

服务器在收到请求时，返回 **HTTP** 响应。响应也有某种结构，每个响应都由状态行开始，可以包含几个头及可能的信息体，称做响应的 **HTTP** 头和响应信息体，这些头和信息体由服务器发送给客户的浏览器，信息体就是客户请求的网页的运行结果，对于 **JSP** 页面，就是网页的静态信息。你可能已经熟悉状态行，状态行说明了正在使用的协议、状态代码及文本信息。例如，若服务器请求出错，则状态行返回错误及对错误的描述，比如“**HTTP/1.1 404 Object Not Found**”。若服务器成功地响应了对网页的请求，返回包含“**200 OK**”的状态行。

3.1 request 对象

HTTP 通信协议是客户与服务器之间一种提交（请求）信息与响应信息（**request/response**）的通信协议。在 **JSP** 中，内置对象 **request** 封装了用户提交的信息，那么该对象调用相应的方法可以获取封装的信息，即使用该对象可以获取用户提交的信息。

客户通常使用 **HTML** 表单向服务器的某个 **JSP** 页面提交信息，表单的一般格式是：

```
<FORM method= get | post action=    “提交信息的目的地页面”>  
    提交手段  
</FORM>.....
```

其中<Form>是表单标签，method 取值 get 或 post。Get 方法和 post 方法的主要区别是：使用 get 方法提交的信息会在提交的过程中显示在浏览器的地址栏中，而 post 方法提交的信息不会显示在地址栏中。提交手段包括：通过文本框、列表、文本区等，例如：

```
<FORM action="tom.jsp" method= “post” >  
    <INPUT type="text" name="boy" value=    “ok” >  
    <INPUT TYPE="submit" value="    送出" name= “submit”>  
</FORM>
```

该表单使用 post 方法向页面 tom.jsp 提交信息，提交信息的手段是：在文本框输入信息，其中默认信息是“ok”；然后点击“送出”按钮向服务器的 JSP 页面 tom.jsp 提交信息。

request 对象可以使用 getParameter(String s) 方法获取该表单通过 text 提交的信息，比如：

```
request.getParameter(“boy”);
```

3.1.1 获取客户提交的信息

request 对象获取客户提交信息的最常用的方法是 getParameter(String s) 。在下面的

例子 1 中, **Example3_1.jsp** 通过表单向 **tree.jsp** 提交信息: **I am a student**。 **tree.jsp** 通过 **request** 对象获取表单提交的信息: 包括 **text** 的值以及按钮的值。

在本章中, 例子中所涉及到的 JSP 页面都保存在 Web 服务目录的根目录: **Root** 中, 即 **D:\tomcat\jakarta-tomcat-4.0\webapps\Root** 中。

例子 1(如图 3.1 所示)

Example3_1.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=green><FONT size=1>

  <FORM action="tree.jsp" method=post name=form>

    <INPUT type="text" name="boy">

    <INPUT TYPE="submit" value="Enter" name="submit">

  </FORM>

</FONT>

</BODY>

</HTML>
```

tree.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=green><FONT size=1>

  <P>获取文本框提交的信息:
```

```

<%String textContent=request.getParameter("boy");

%>

<BR>

<%=textContent%>

<P> 获取按钮的名字:

<%String buttonName=request.getParameter("submit");

%>

<BR>

<%=buttonName%>

</FONT>

</BODY>

</HTML>

```

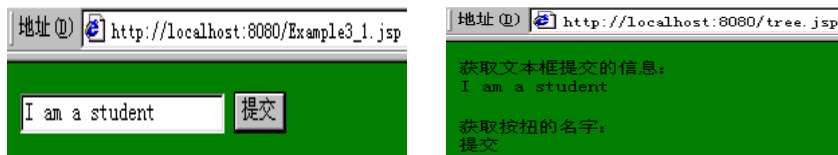


图 3.1 提交信息与获取信息

在下面的例子 2 中，Example3_2.jsp 通过表单向自己提交一个正数，然后计算这个数的平方根

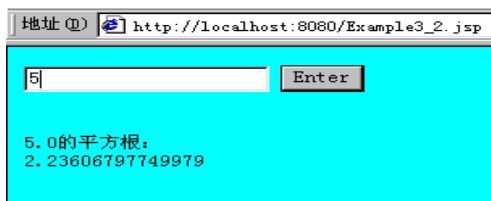


图 3.2 计算数的平方根

例子 2(如图 3.2 所示)

Example3_2.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><FONT size=1>

  <FORM action="Example3_2.jsp" method=post name=form>

    <INPUT type="text" name="girl">

    <INPUT TYPE="submit" value="Enter" name="submit">

  </FORM>

  <%String textContent=request.getParameter("girl");

  double number=0,r=0;

  if(textContent==null)

    {textContent="";

    }

  try{ number=Double.parseDouble(textContent);
```

```

    if(number>=0)

        {r=Math.sqrt(number) ;

        out.print("<BR>" +String.valueOf(number)+"          的平方根: ");

        out.print("<BR>" +String.valueOf(r));

        }

    else

        {out.print("<BR>" + "          请输入一个正数");

        }

    }

    catch(NumberFormatException e)

        {out.print("<BR>" + "          请输入数字字符");

        }

    %>

</FONT>

</BODY>

</HTML>

```

注：使用 request 对象获取信息时要格外小心，在上面的例子中：

```
String textContent =request.getParameter("girl");
```

获取提交的字符串信息，并且在下面的代码中使用了这个字符串对象：

```
number=Doule.parseDoubel(textContent);
```

那么，JSP 引擎在运行这个 JSP 页面生成的字节码文件时，会认为你使用了空对象，因为在这个字节码被执行时（客户请求页面时），客户可能还没有提交数据，textContent 还没有被创建。如果你使用了空对象，即还没有创建对象，就使用了该对象，Java 解释器

就会提示出现了 `NullPointerException` 异常,当然如果你不使用空对象就不会出现异常。

因此,我们可以象上述例子那样,为了避免在运行时 Java 认为我们使用了空对象,使用如下代码:

```
String textContent=request.getParameter("girl");

if(textContent==null)

    {textContent="";

    }
```

3.1.2 处理汉字信息

当用 `request` 对象获取客户提交的汉字字符时,会出现乱码问题,所以对含有汉字字符的信息必须进行特殊的处理方式。首先,将获取的字符串用 **ISO-8859-1** 进行编码,并将编码存放到一个字节数组中,然后再将这个数组转化为字符串对象即可。如下列所示:

```
String str=request.getParameter("girl");

byte b[]=str.getBytes( "ISO-8859-1");

str=new String(b);
```

通过上述过程,提交的任何信息(无论是汉字字符或西欧字符)都能正确的显示。

下面的例子 3 对例子 1 按上述办法做了改动,并将按钮上的字变成汉语,在文本框里输入:“苹果: apple:12 斤 5\$”,然后提交给 `tree.jsp`。

例子 3 (如图 3.3 所示)

Example3_3.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=green><FONT size=1>

    <FORM action="tree.jsp" method=post name=form>

        <INPUT type="text" name="boy">

        <INPUT TYPE="submit" value="    提交" name="submit">

    </FORM>

</FONT>

</BODY>

</HTML>

```

tree.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<MHML>

<BODY>

<P>获取文本框提交的信息:

    <%String textContent=request.getParameter("boy");

        byte b[]=textContent.getBytes("ISO-8859-1");

        textContent=new String(b);

    %>

<BR>

    <%=textContent%>

<P> 获取按钮的名字:

```

```

<%String buttonName=request.getParameter("submit");

byte c[]=buttonName.getBytes("ISO-8859-1");

buttonName=new String(c);

%>

<BR>

<%=buttonName%>

</BODY>

</HTML>

```

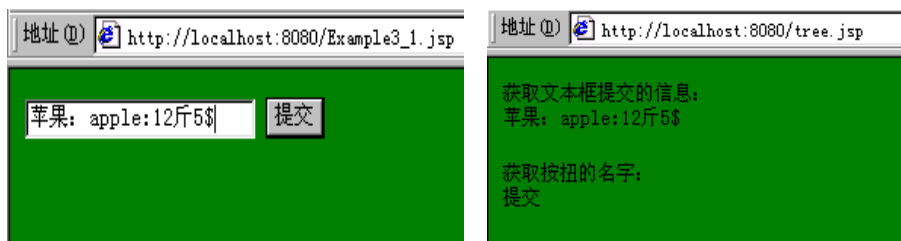


图 3.3 提交和获取含有汉字的信息

3.1.3 常用方法举例

当客户访问一个页面时，会提交一个 **HTTP** 请求给服务器的 **JSP** 引擎，这个请求包括一个请求行、**http** 头和信息体，如下列：

```

post/tree2.jsp/HTTP.1.1

host: localhost    : 8080

```

accept-encoding : gzip, deflate

其中首行叫请求行，规定了向访问的页面请求提交信息的方式，如，**post**、**get** 等方式，以及请求的页面的文件名字和使用的通信协议。

第 2、3 行分别是两个头(Header)，称 **host**、**accept-encoding** 是头名字，而 **localhost:8080** 以及 **gzip,deflate** 分别是它们的值。这里 **host** 的值是 **tree2.jsp** 的地址。上面的请求有 2 个头：**host** 和 **accept-encoding**，一个典型的请求通常包含很多的头，有些头是标准的，有些和特定的浏览器有关。

一个请求还包含信息体，即 **HTML** 标记组成的部分，可能包括各式各样用于提交信息的表单等，如：

<BODY>

<FORM action="tree2.jsp" method=post name=form>

<INPUT type="text" name="boy">

<INPUT TYPE="submit" value="" name="submit">

</FORM>

</BODY>

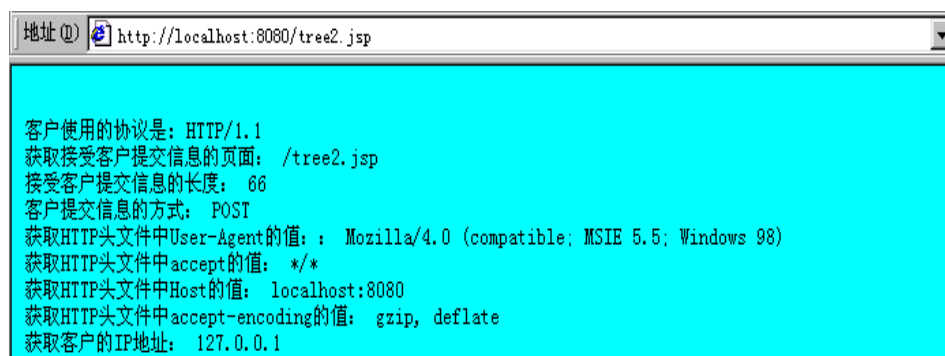
我们可以使用 **JSP** 引擎的内置对象 **request** 对象来获取客户提交的信息，说明如下：

1. **getProtocol()** 获取客户向服务器提交信息所使用的通信协议，比如 **http/1.1** 等。
2. **getServletPath()** 获取客户请求的 **JSP** 页面文件的目录。
3. **getContentLength()** 获取客户提交的整个信息的长度。
4. **getMethod()** 获取客户提交信息的方式，比如：**post** 或 **get**。

5. `getHeader(String s)` 获取 HTTP 头文件中由参数 `s` 指定的头名字的值,一般来说 `s` 参数可取的头名有: `accept`、`referer`、`accept-language`、`content-type`、`accept-encoding`、`user-agent`、`host`、`content-length`、`connection`、`cookie` 等,比如, `s` 取值 `user-agent` 将获取客户的浏览器的版本号等信息。
6. `getHeaderNames()` 获取头名字的一个枚举
7. `getHeaders(String s)` 获取头文件中指定头名字的全部值的一个枚举
8. `getRemoteAddr()` 获取客户的 IP 地址。
9. `getRemoteHost()` 获取客户机的名称 (如果获取不到, 就获取 IP 地址)。
10. `getServerName()` 获取服务器的名称。
11. `getServerPort()` 获取服务器的端口号。
12. `getParameterNames()` 获取客户提交的信息体部分中 `name` 参数值的一个枚举。



图 3.4 提交信息



下面的例子 4 使用了 **request** 的一些常用方法。

例子 4(如图 3.4、3.5 所示)

Example3_4.jsp:

```
<HTML>
```

```
<BODY bgcolor=cyan><FONT size=1>
```



```

<%@ page contentType="text/html;charset=GB2312" %>

<FORM action="tree2.jsp" method=post name=form>

    <INPUT type="text" name="boy">

    <INPUT TYPE="submit" value="enter" name="submit">

</FORM>

</FONT>

</BODY>

</HTML>

```

tree2.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.util.*" %>

<MHML>

<BODY bgcolor=cyan>

<Font size=1 >

<BR>客户使用的协议是:

    <% String protocol=request.getProtocol();

        out.println(protocol);

    %>

<BR>获取接受客户提交信息的页面:

    <% String path=request.getServletPath();

        out.println(path);

    %>

```


接受客户提交信息的长度:

```
<% int length=request.getContentLength();  
    out.println(length);  
%>
```


客户提交信息的方式:

```
<% String method=request.getMethod();  
    out.println(method);  
%>
```


获取 HTTP 头文件中 User-Agent 的值: :

```
<% String header1=request.getHeader("User-Agent");  
    out.println(header1);  
%>
```


获取 HTTP 头文件中 accept 的值:

```
<% String header2=request.getHeader("accept");  
    out.println(header2);  
%>
```


获取 HTTP 头文件中 Host 的值:

```
<% String header3=request.getHeader("Host");  
    out.println(header3);  
%>
```


获取 HTTP 头文件中 accept-encoding 的值:

```
<% String header4=request.getHeader("accept-encoding");  
    out.println(header4);
```

```
%>
```


获取客户的 IP 地址:

```
<% String IP=request.getRemoteAddr();
```

```
    out.println(IP);
```

```
%>
```


获取客户机的名称:

```
<% String clientName=request.getRemoteHost();
```

```
    out.println(clientName);
```

```
%>
```


获取服务器的名称:

```
<% String serverName=request.getServerName();
```

```
    out.println(serverName);
```

```
%>
```


获取服务器的端口号:

```
<% int serverPort=request.getServerPort();
```

```
    out.println(serverPort);
```

```
%>
```


获取客户端提交的所有参数的名字:

```
<% Enumeration enum=request.getParameterNames();
```

```
    while(enum.hasMoreElements())
```

```
    {String s=(String)enum.nextElement();
```

```
        out.println(s);
```

```
    }
```

%>

**
**获取头名字的一个枚举:

```
<% Enumeration enum_headed=request.getHeaderNames();
```

```
while(enum_headed.hasMoreElements())
```

```
{String s=(String)enum_headed.nextElement();
```

```
out.println(s);
```

```
}
```

%>

**
**获取头文件中指定头名字的全部值的一个枚举:

```
<% Enumeration enum_headedValues=request.getHeaders("cookie");
```

```
while(enum_headedValues.hasMoreElements())
```

```
{String s=(String)enum_headedValues.nextElement();
```

```
out.println(s);
```

```
}
```

%>

**
**

<P> 文本框 text 提交的信息:

```
<%String str=request.getParameter("boy");
```

```
byte b[]=str.getBytes("ISO-8859-1");
```

```
str=new String(b);
```

%>

**
**

```
<%=str%>
```


 按钮的名字:

```
<%String buttonName=request.getParameter("submit");  
    byte c[]=buttonName.getBytes("ISO-8859-1");  
    buttonName=new String(c);  
%>  
<BR>  
    <%=buttonName%>  
</Font>  
</BODY>  
</HTML>
```

3.1.4 用户注册

在下面的例子 5 中，用户通过提交姓名和 Email 地址实现注册。当 request 对象获取这些信息后，首先检查散列表对象中是否已经存在这个名字，该散列表存储了已经注册的用户的名字。如果目前准备注册的用户提交的名字在散列表中已经存在，就提示客户更换名字，否则将检查客户是否提供了书写正确的 Email 地址，如果提供了书写正确 Email 地址将允许注册。

在下面的例子 5 中，使用了散列表。散列表是使用相关关键字查找被存储的数据项的一种数据结构，关键字不可以发生逻辑冲突，即不要两个数据项使用相同的關鍵字。散列表在它需要更多的存储空间时会自动增大容量。例如，如果散列表的装载因子是 0.75，那么当散列表的容量被使用了 75% 时，它就把容量增加到原始容量的 2 倍。对于数组和链表这两种数据结构，如果要查找它们存储的某个特定的元素却不知道它的位置，就需要从头开始访问元素直到找到匹配的为止，如果数据结构中包含很多的元素，就会浪费时间。这时最好使用散列表来存储要查找的数据。

我们将例子 5 中的 login1.jsp 和 login2.jsp 保存在 Web 服务目录 root 中，用户首先访问 login1.jsp，输入名字和 Email 提交给 login2.jsp 实现注册。

例子 5(如图 3.6 所示)

Login1.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><Font size=1 >

<FORM action="login2.jsp" method=post >

    <P>    输入你的姓名:

    <INPUT type="text" name="name" value="abc">

    <BR>

    <P>    输入你的 e-mail 地址:

    <INPUT type="text" name="address" value="    ookk@sina.com">

    <P>    点击送出按钮:

    <BR>

    <INPUT TYPE="submit" value="    送出" name=submit>

</FORM>

</FONT>

</BODY>

</HTML>
```

login2.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.util.*" %>

<HTML>

<BODY bgcolor=cyan><Font size=1 >

<%!Hashtable hashtable=new Hashtable();

    public synchronized void putString(String s)

    { hashtable.put(s,s);

    }

%>

<% String person_name=request.getParameter("name"),

    name_found=null;

    if(person_name==null)

    {person_name="";

    }

    byte b[]=person_name.getBytes("ISO-8859-1");

    person_name=new String(b);

    name_found=(String)hashtable.get(person_name);

    if(name_found==null)

    { String person_email=request.getParameter("address");

    if(person_email==null)

    {person_email="";

    }

    StringTokenizer fenxi=new StringTokenizer(person_email," @");

```

```

int n=fenxi.countTokens();

if(n>=3)

    {out.print("<BR>"+ "          你输入的 Email 有不合法字符");

    }

else

    { putString(person_name);

    out.print("<BR>"+ "          您已经注册成功");

    out.print("<BR>"+ "          您注册的名字是"+person_name);

    }

}

}

else

    {out.print("<BR>"+ "          该名字已经存在，请您换个名字");

    }

%>

</FONT>

</BODY>

</HTML>

```

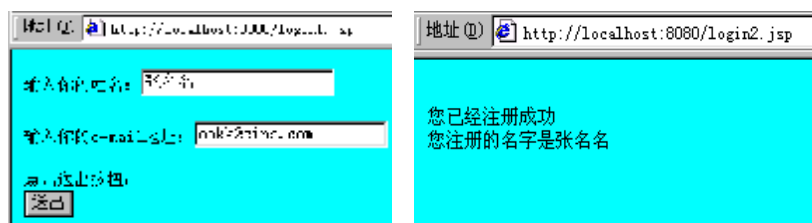


图 3.6 用户注册

注：在上面的例子中，如果服务器重新启动将会刷新散列表。所以，我们应当将散列表存放到文件中，当客户访问服务器时首先从文件中读出散列表，在这个散列表中查找已经注册的名字，如果文件不存在，那么该客户就是第一个注册的人，就负责将散列表写入到文件。我们将 login2.jsp 改进如下：

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.util.*" %>

<%@ page import="java.io.*" %>

<HTML>

<BODY>

<%!Hashtable hashtable=new Hashtable();

    public synchronized void putString(String s)

    { hashtable.put(s,s);

    }

%>

<% String person_name=request.getParameter("name"),name_found=null;

    if(person_name==null)

        {person_name="";

        }

    byte c[]=person_name.getBytes("ISO-8859-1");
```

```
person_name=new String(c);
```

```
%>
```

<%--从文件中读散列表，如果文件不存在，你就是第一个访问本站的人，负责写散列表到文件--%>

```
<%try{File f=new File("name.txt");
```

```
FileInputStream in=new FileInputStream(f);
```

```
ObjectInputStream object_in=new ObjectInputStream(in);
```

```
hashtable=(Hashtable)object_in.readObject();
```

```
object_in.close();
```

```
in.close();
```

```
name_found=(String)hashtable.get(person_name);
```

```
if(name_found==null)
```

```
{ String person_email=request.getParameter("address");
```

```
if(person_email==null)
```

```
{person_email="";
```

```
}
```

```
StringTokenizer fenxi=new StringTokenizer(person_email,"
```

```
@");
```

```
int n=fenxi.countTokens();
```

```
if(n>=3)
```

```
{out.print("<BR>"+"你输入的 Email 有不合法字符");
```

```
}
```

```
else
```

```

        { putString(person_name);

        try{FileOutputStream o=new FileOutputStream(f);

            ObjectOutputStream object_out=new
ObjectOutputStream(o);

            object_out.writeObject(hashtable);

            object_out.close();

            o.close();

        }

        catch(Exception eee)

        {

        }

        out.print("<BR>"+ "                您已经注册成功");

        out.print("<BR>"+ "                您注册的名字是"+person_name);


        }

    }

else

    {out.print("<BR>"+ "                该名字已经存在，请您换个名字");

    }

}

catch(Exception e)

    { String person_email=request.getParameter("address");

    if(person_email==null)

```

```

        {person_email="";

    }

    StringTokenizer fenxi=new StringTokenizer(person_email," @");

    int n=fenxi.countTokens();

    if(n>=3)

        {out.print("<BR>"+"                你输入的 Email 有不合法字符");

        }

    else

        { putString(person_name);

          try{File f=new File("name.txt");

              FileOutputStream o=new FileOutputStream(f);

              ObjectOutputStream object_out=new

ObjectOutputStream(o);

              object_out.writeObject(hashtable);

              object_out.close();

              o.close();

          }

          catch(Exception eee)

          {

          }

          out.print("<BR>"+"                恭喜!, 您是第一个注册成功的人");

          out.print("<BR>"+"                您注册的名字是"+person_name);

        }

```

```
    }  
    %>  
</BODY>  
</HTML>
```

3.1.5 获取 HTML 表单提交的数据

由于客户经常需要使用表单提交数据，所以有必要对表单做一个简明的介绍，如果您对 **HTML** 语言比较陌生，建议补充这方面的知识。

表单的一般格式是：

```
<FORM method= get| post action=    ”提交信息的目的地页面” name= ”表单的名字”>
```

数据提交手段部分

```
</FORM>.....
```

其中<Form>是表单标签，method 取值 get 或 post。get 方法和 post 方法的主要区别是：使用 **get** 方法提交的信息会在提交的过程中显示在浏览器的地址栏中，而 **post** 方法提交的信息不会显示在地址栏中。提交手段包括：通过文本框、列表、文本区等，例如：

```
<FORM action="tom.jsp" method= “post” >  
    <INPUT type="text" name="boy" value=    “ok” >  
    <INPUT TYPE="submit" value="    送出" name= “submit”>  
</FORM>
```

一个表单的数据提交手段部分经常包括如下的标记符号：

l <INPUT>

l <Select ... ></Select>

l <Option> </Option>

l <TextArea> </TextArea>

1. <Input> 的基本格式

在表单中用 **Input** 标记来指定表单中数据的输入方式以及表单的提交键。**Input** 标记中的 **type** 属性可以指定输入方式的 GUI 对象, **name** 属性用来指定这个 GUI 对象的名称。基本格式：

```
<input type= "输入对象的 GUI 类型" name= "名字">
```

服务器通过属性 **name** 指定的名字来获取“输入对象的 GUI 类型”中提交的数据。“输入对象的 GUI 类型”可以是：**text**（文本框）、**checkbox**（检查框）、**submit**（提交键）等。

（1）文本框：text

当输入对象的 GUI 类型是 **text** 时，除了用 **name** 为 **text** 指定名字外，还可以为 **text** 指定其它的一些值。比如：

```
<input type= "text" name= "me" value= "hi" size= "12 " algin= "left"
maxlength= "30">
```

其中，**value** 的值是 **text** 的初始值；**size** 是 **text** 对象的长度（单位是字符）；**align** 是 **text** 在浏览器窗体中的对齐方式；**maxlength** 指定 **text** 可输入字符的最大长度。

（2）单选框：radio

当输入对象的 GUI 类型是 **radio** 时，除了用 **name** 为 **radio** 指定名字外，还可以为 **radio** 指定其它的一些值。比如：

```
<input type= "radio" name= "rad" value= "red" align= "top" checked= "java" >
```

其中，**value** 指定 **radio** 的值；**align** 是 **radio** 在浏览器窗体中的对齐方式；如果几个单选键的 **name** 取值相同，那么同一时刻只能有一个被选中。服务器通过 **name** 指定的名字来获取被选中的 **radio** 提交的由 **value** 指定的值。**checked** 如果取值是一个非空的字符串，那么该单选框的初始状态就是选中状态。

在下面的例子 6 中，我们用单选框来实现一个网上小测试。客户在 **radio.jsp** 页面中选中几个单选框，将选择提交给 **answer.jsp** 页面。

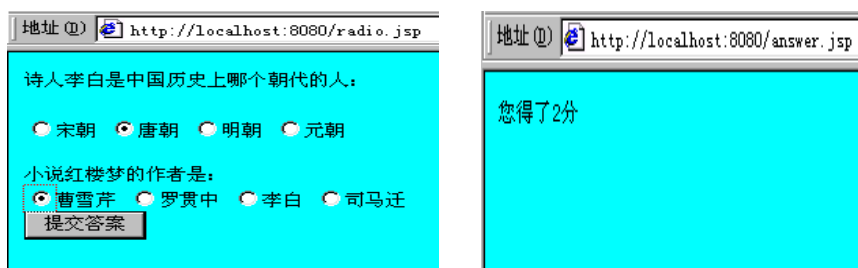


图 3.7 获取单选框的值

例子 6(如图 3.7 所示)

radio.jsp:

<HTML>

<%@ page contentType="text/html;charset=GB2312" %>

<BODY bgcolor=cyan>

<P> 诗人李白是中国历史上哪个朝代的人:

<FORM action="answer.jsp" method=post name=form>

<INPUT type="radio" name="R" value="a"> 宋朝

<INPUT type="radio" name="R" value="b"> 唐朝

<INPUT type="radio" name="R" value="c"> 明朝

<INPUT type="radio" name="R" value="d" checked="ok"> 元朝

**
**

<P> 小说红楼梦的作者是:

**
**

<INPUT type="radio" name="P" value="a"> 曹雪芹

<INPUT type="radio" name="P" value="b"> 罗贯中

<INPUT type="radio" name="P" value="c"> 李白

<INPUT type="radio" name="P" value="d"> 司马迁

**
**

<INPUT TYPE="submit" value=" 提交答案" name="submit">

</FORM>

</BODY>

</HTML>

answer.jsp:

<HTML>

<%@ page contentType="text/html;charset=GB2312" %>

<BODY bgcolor=cyan>

<% int n=0;

String s1=request.getParameter("R");

String s2=request.getParameter("P");

if(s1==null)

{s1="";}

if(s2==null)

{s2="";}

if(s1.equals("b"))

{ n++;}

if(s2.equals("a"))

{ n++;}

%>

<P>您得了<%=n%>分

</BODY>

</HTML>

(3)复选框: **checkbox**:

当输入对象的 GUI 类型是 **checkbox** 时,除了用 **name** 为 **checkbox** 指定名字外,还可以为 **checkbox** 指定其它的一些值。比如:

```
<input type= "checkbox" name= "ch" value= "pink" align= "top" checked=
"java" >
```

其中, **value** 指定 **checkbox** 的值;复选框与单选框的区别就是可以多选。服务器通过 **name** 指定的名字来获取被选中的 **checkbox** 提交的由 **value** 指定的值,为了使服务器能获取提交的值,复选框 **name** 的值应互不相同。**Checked** 如果取值是一个非空的字符串,那么该复选框的初始状态就是选中状态。

(4)口令框: **password**

它是输入口令用的特殊文本框,输入的信息用 “*” 回显,防止他人偷看口令。

```
<input type= "password" name= "me" size= "12 " maxlength= "30">
```

服务器通过 **name** 指定的字符串获取 **password** 提交的值,你在口令框中输入: **bird88_1**,那么 **bird88_1** 将被提交给服务器,口令框仅仅起着不让别人偷看的作用,不提供保密措施。

(5)提交键: **submit**

为了能把表单的数据提交给服务器,一个表单至少要包含一个提交键。

<input type= “submit ” name= “me” value= “ok” size= “12 ” >

点击提交键后，服务器就可以获取表单提交的各个数据。当然服务器也可以获取提交键的值，服务器通过 **name** 指定的名字来获取提交键提交的由 **value** 指定的值。

(6)重置键：reset

重置键将表单中输入的数据清空，以便重新输入数据。

<input type= “reset” >

2. <Select> 、 <Option>格式

下拉式列表和滚动列表通过<Select>和<Option>标记来定义，基本格式为：

<Selection>

<Option>

<Option>

... ...

</Selection>

(1) 下拉列表

<Select name="shulie" >

<Option value="cat"> 你选了小猫

`<Option value="dog">` 你选了小狗

....

`<Option value="600">n=600`

`</Select>`

服务器通过 **name** 获取下拉列表中被选中的 **option** 的值（参数 **value** 指定的值）。

（2） 滚动列表

在 **select** 中增加 **size** 属性的值就变成滚动列表，**size** 的值是滚动列表的可见行的个数。

`<Select name="shulie" size=2>`

`<Option value="1">` 计算 1 到 n 的连续和

`<Option value="2">` 计算 1 到 n 的平方和

`<Option value="3">` 计算 1 到 n 的立方和

`</Select>`

服务器通过 **name** 获取滚动列表中被选中的 **option** 的值（参数 **value** 指定的值）。

在下面的例子 7 中，客户通过滚动列表选择计算求和的方式，通过下拉列表选择计算求和的项数。

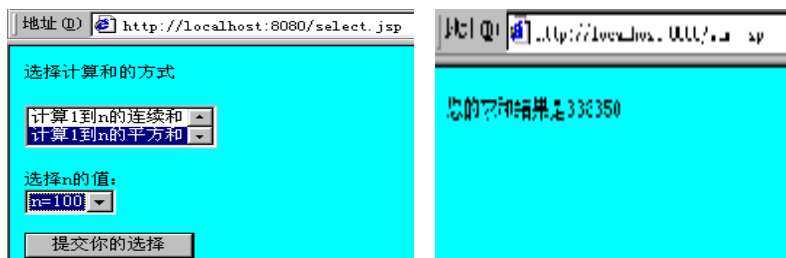


图 3.8 获取 select 提交的值，计算数列和

例子 7(如图 3.8 所示)

select.jsp:

<HTML>

<%@ page contentType="text/html;charset=GB2312" %>

<BODY bgcolor=cyan>

<P> 选择计算和的方式

<FORM action="sum.jsp" method=post name=form>

<Select name="sum" size=2>

<Option Selected value="1"> 计算 1 到 n 的连续和

<Option value="2"> 计算 1 到 n 的平方和

<Option value="3"> 计算 1 到 n 的立方和

</Select>

**<P> 选择 n 的值:
**

<Select name="n" >

<Option value="10">n=10

<Option value="20">n=20

<Option value="30">n=30

<Option value="40">n=40

<Option value="50">n=50

<Option value="100">n=100

```

</Select>

<BR><BR>

<INPUT TYPE="submit" value="    提交你的选择" name="submit">

</FORM>

</FONT>

</BODY>

</HTML>

```

sum.jsp:

```

<HTML>

<%@ page contentType="text/html;charset=GB2312" %>

<BODY bgcolor=cyan><Font size=1 >

<% long sum=0;

String s1=request.getParameter("sum");

String s2=request.getParameter("n");

if(s1==null)

    {s1="";}

if(s2==null)

    {s2="0";}

if(s1.equals("1"))

    {int n=Integer.parseInt(s2);

    for(int i=1;i<=n;i++)

        {sum=sum+i;

```

```

    }
}
else if(s1.equals("2"))
{int n=Integer.parseInt(s2);
  for(int i=1;i<=n;i++)
    {sum=sum+i*i;
    }
}
else if(s1.equals("3"))
{int n=Integer.parseInt(s2);
  for(int i=1;i<=n;i++)
    {sum=sum+i*i*i;
    }
}
%>
<P>您的求和结果是<%=sum%>
</FONT>
</BODY>
</HTML>

```

3. <TextArea> 格式

<TextArea> 标记在表单中指定一个能输入多行文本的文本区域。

```
<TextArea name= "ilovethisgame" Rows= "4" Cols= "20" >
</TextArea>
```

3.1.6 表格

表格由<Table>、</Table>标记定义，一般格式：

```
<Table >
    <TR width=    "该行的宽度" >
        <TH width=    "单元格的宽度">单元格中的数据</TH>
        ...
        <TD width=    "单元格的宽度">单元格中的数据</TD> ...
    </TR>
    ... ....
</Table>
```

其中

```
<TR>
....
</TR>
```

定义表格的一个行，<TH>和<TD>标记定义这一行中的表格单元，二者的区别是<TH>定义的单元着重显示，<TD>称做普通单元，不着重显示；一行中的着重单元和普通的单元可以交替出现，也可以全是着重单元或普通单元。

<table>中增加选项 **Border** 可指明该表格是否带有边框。

在下面的例子 8 中，用一个 4 行的表格显示数据。第一行有 3 个着重显示的单元。第 2 行有 3 个单元，其中第一个单元着重显示。第 3 行有 3 个普通的单元，如图 3.9 所示。

例子 8 (如图 3.9 所示)

table.jsp:

```
<HTML>

<%@ page contentType="text/html;charset=GB2312" %>

<BODY>

<Table align="Center" Border>

    <TR width=400>

        <TH Align="Center">        中间</TH>

        <TH Align="Right">        右</TH>

        <TH Align="LEFT">        左</TH>

        <TD></TD>

        <TD></TD>

    </TR>

    <TR >

        <TH Valign="Top">        数据靠向上沿</TH>

        <TD Valign="Bottom">        数据靠向下沿</TD>

        <TD Valign="Bottom" Align="Center" >        数据居中靠向下沿</TD>

    </TR>

    <TR >

        <TD Valign="Top">        你好</TD>

        <TD Valign="Bottom">hello</TD>

        <TD Valign="Bottom" Align="Center" >112334</TD>

    </TR>
```

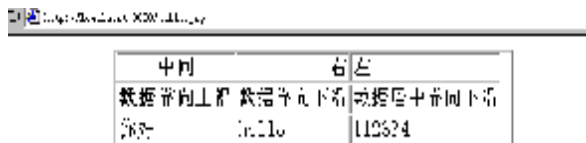
```

</Table>

</BODY>

</HTML>

```



中间	右左
数据带向上和	数据带向下和
数据带向上和	数据带向下和
100%	112524

图 3.9 用表格显示数据

在下面的例子 9 中，把一个表单显示在表格的一个单元格中，如图 3.10 所示。



李白是哪个朝代的人？：	<input type="radio"/> 宋朝 <input type="radio"/> 唐朝 <input type="button" value="送出"/>
输入数据：	<input type="text" value="a"/> <input type="button" value="送出"/>

图 3.10 表格中存放表单

例子 9（如图 3.10 所示）

tableform.jsp:

```

<HTML>

<%@ page contentType="text/html;charset=GB2312" %>

<BODY bgcolor=cyan>

<Table align="left" Border>

```

```

<TR>

    <TH width=250><FONT size=1>        李白是哪个朝代的人? : </TH>

    <TD width=220>

        <FORM action="answer.jsp" method=post name=form>

            <INPUT type="radio" name="R" value="a" ><FONT size=1>        宋
朝

            <INPUT type="radio" name="R" value="b" ><FONT size=1>        唐
朝

            <INPUT type="submit" name="g" value="        送出">

        </FORM>

    </TD>

</TR>

<TR >

    <TH><FONT size=1>        输入数据: </TH>

    <FORM action="answer.jsp" method=post name=form>

        <TD> <INPUT type="text" name="R" value="a" size=20 >

        </TD>

        <TD> <INPUT type="submit" name="f" value="        送出" >

        </TD>

    </Form>

</TR>

</Table>

</BODY>

```

</HTML>

3.2 response 对象

当客户访问一个服务器的页面时，会提交一个 **HTTP** 请求，服务器收到请求时，返回 **HTTP** 响应。响应和请求类似，也有某种结构，每个响应都由状态行开始，可以包含几个头及可能的信息体（网页的结果输出部分）。

上一节学习了用 **request** 对象获取客户请求提交的信息，与 **request** 对象相对应的对象是 **response** 对象。我们可以用 **response** 对象对客户请求作出动态响应，向客户端发送数据。比如，当一个客户请求访问一个 **JSP** 页面时，该页面用 **page** 指令设置页面的 **contentType** 属性的值是 **text/html**，那么 **JSP** 引擎将按着这种属性值响应客户对页面的请求，将页面的静态部分返回给客户。如果想动态地改变 **contentType** 的属性值就需要用 **response** 对象改变页面的这个属性的值，作出动态的响应。

3.2.1 动态响应 contentType 属性

当一个客户请求访问一个 **JSP** 页面时，如果该页面用 **page** 指令设置页面的 **contentType** 属性的值是 **text/html**，那么 **JSP** 引擎将按着这种属性值作出响应，将页面的静态部分返回给客户。由于 **page** 指令只能为 **contentType** 指定一个值，来决定响应的 **MIME** 类型，如果想动态的改变这个属性的值来响应客户，就需要使用 **response** 对象的 **setContentType(String s)**方法来改变 **contentType** 的属性值：

```
public void setContentType(String s);
```

该方法动态设置响应的 **MIME** 类型，参数 **s** 可取：**text/html**、**text/plain**、**application/x-msexcel**、**application/msword** 等

当服务器用 `setContentType` 方法动态改变了 `contentType` 的属性值，即响应的 MIME 类型，并将 JSP 页面的输出结果按着新的 MIME 类型返回给客户时，客户端要保证支持这种新的 MIME 类型。客户如果想知道自己的浏览器能支持哪些 MIME 类型，可以点击资源管理器→工具→文件夹选项→文件类型。如图 3.11 所示。

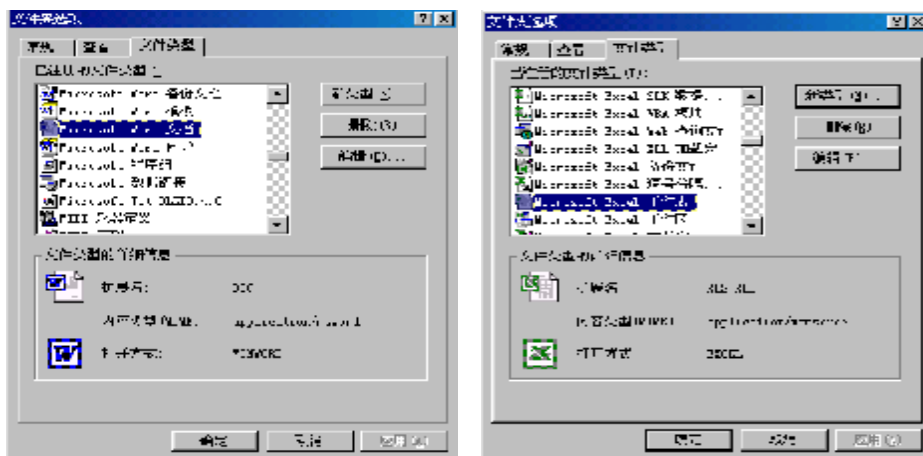


图 3.11 查看 MIME 类型

在下面的例子 10 中，当客户点击按钮，选择将当前页面保存为一个 Word 文档时，JSP 页面动态地改变 `contentType` 的属性值为 `application/msword`。这时，客户的浏览器会提示客户用 Ms-Word 格式来显示当前页面，如图 3.12 所示。

例子 10 (如图 3.12 所示)

Example3_10.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><Font size=1 >

<P> 我正在学习 response 对象的

<BR>setContentType 方法

<P> 将当前页面保存为 word 文档吗?

<FORM action="" method="get" name=form>

    <INPUT TYPE="submit" value="yes" name="submit">

</FORM>

<% String str=request.getParameter("submit");

    if(str==null)

        {str="";

        }

    if(str.equals("yes"))

        {response.setContentType("application/msword;charset=GB2312");

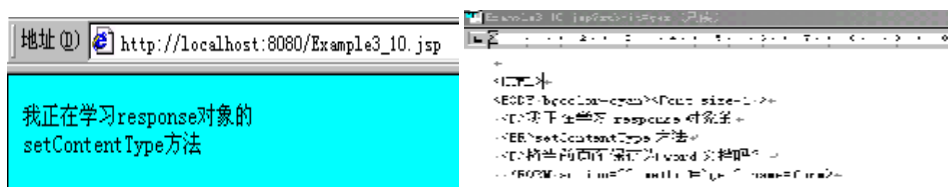
        }

%>

</FONT>

</BODY>

</HTML>
```



在下面的例子中，当客户选择用 Excel 表格显示 JSP 页面中的一个 A.txt 文件时，我们用 response 对象将 contentType 的属性值设为 "application/x-msexcel"。需要注意的是：在编辑文本文件 A.txt 时，回车要用
 来表示，输入空格时要将输入法切换到全角（因为半角输入的多个空格被浏览器认为是一个空格）。为了能用 Excel 显示该文件，数据列之间要有 4 个空格（必须在全角状态下编辑空格）。A.txt 和 JSP 页面保存在同一目录中。

A.txt:

34	79	51	99
40	89	92	99
64	99	30	99
74	56	80	99
87	97	88	99
74	65	56	99
67	75	67	66
89	77	88	99

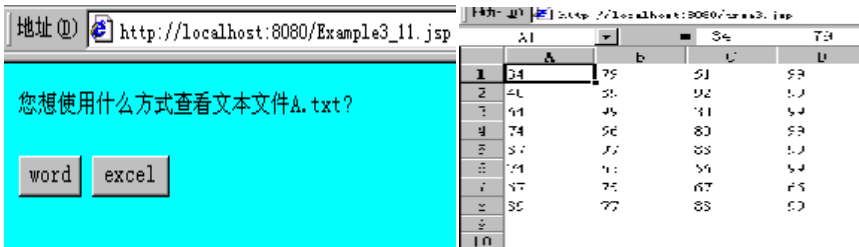


图 3.13 用指定的 MIME 类型查看文本

例子 11（如图 3.13 所示）

Example3_11.jsp:

```
<%@ page contentType="text/html; charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><Font size=1 >

<P> 您想使用什么方式查看文本文件 A.txt?

<FORM action="tree3.jsp" method="post" name=form>

    <INPUT TYPE="submit" value="word" name="submit1">

    <INPUT TYPE="submit" value="excel" name="submit2">

</FORM>

</FONT>

</BODY>

</HTML>
```


tree3.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<% String str1=request.getParameter("submit1");

String str2=request.getParameter("submit2");

if(str1==null)

    {str1="";

    }

if(str2==null)

    {str2="";

    }

if(str1.startsWith("word"))

    {response.setContentType("application/msword;charset=GB2312");

    out.print(str1);

    }

if(str2.startsWith("excel"))

    {response.setContentType("application/x-msexcel;charset=GB2312");

    }

%>

<jsp:include page="A.txt">

</jsp:include>
```

</BODY>

</HTML>

注：当把 `contentType` 的属性值设为 `text/plain`(纯文本)时，如果客户使用的是 Netscape 浏览器，那么 HTML 标记将不被解释，客户以纯文本的形式观看当前网页的输出结果，但 Microsoft 的 Internet Explorer 将解释 HTML 标记。

3.2.2 response 的 HTTP 文件头

我们已经知道，当客户访问一个页面时，会提交一个 **HTTP** 头给服务器，这个请求包括一个请求行、**http** 头和信息体，如下列：

```
post/tree3.jsp/HTTP.1.1
```

```
host: localhost : 8080
```

```
accept-encoding : gzip, deflate
```

第 2、3 行分别是两个头，称 **host**、**accept-encoding** 是头名字，而 **localhost:8080** 以及 **gzip,deflate** 分别是它们的值。这里规定了 **host** 的值是 **tree3.jsp** 的地址。上面的请求有 2 个头：**host** 和 **accept-encoding**，一个典型的请求通常包含很多的头，有些头是标准的，有些和特定的浏览器有关。

同样，响应也包括一些头。**response** 对象可以使用方法

```
addHeader(String head,String value);
```

或方法

```
setHeader(String head ,String value)
```

动态添加新的响应头和头的值，将这些头发送给客户的浏览器。如果添加的头已经存在，则先前的头被覆盖。

在下面的例子 12 中，`response` 对象添加一个响应头：`“refresh”`，其头值是 `“5”`。那么客户收到这个头之后，5 秒钟后将再次刷新该页面，导致该网页每 5 秒刷新一次，如图 3.14 所示。

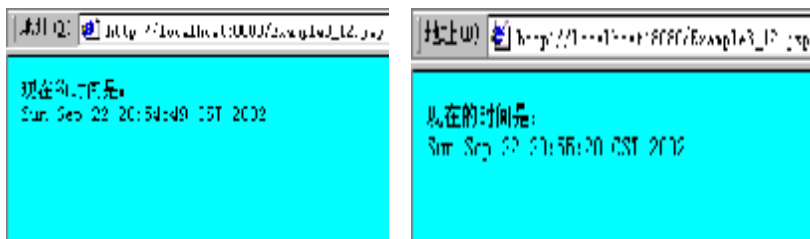


图 3.14 设置响应头

例子 12（如图 3.14 所示）

Example3_12.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.util.*" %>

<HTML>

<BODY bgcolor=cyan><Font size=1 >

<P>现在的时间是: <BR>

<% out.println("'" + new Date());

    response.setHeader("Refresh","5");

%>
```

</BODY>

</HTML>

3.2.3 response 重定向

在某些情况下，当响应客户时，需要将客户重新引导至另一个页面。例如，如果客户输入的表单信息不完整，就会再被引导到该表单的输入页面。

可以使用 **response** 的 **sendRedirect(URL url)** 方法实现客户的重定向。

在下面的例子 13 中，客户在 **Example3_13.jsp** 页面填写表单提交给 **tree4.jsp** 页面，如果填写的表单不完整就会重新定向到 **Example3_13.jsp** 页面。

例子 13

Example3_13.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<HTML>
```

```
<BODY>
```

```
<P>填写姓名: <BR>
```

```
<FORM action="tree4.jsp" method="get" name=form>
```

```
<INPUT TYPE="text" name="boy">
```

```
<INPUT TYPE="submit" value="Enter">
```

```
</FORM>
```

```
</BODY>
```

```
</HTML>
```

tree4.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

    <%    String str=null;

            str=request.getParameter("boy");

            if(str==null)

                {str="";

                }

            byte b[]=str.getBytes("ISO-8859-1");

            str=new String(b);

            if(str.equals(""))

                {response.sendRedirect("Example3_13.jsp");

                }

            else

                {out.print("欢迎您来到本网页！");

                out.print(str);

                }

    %>

</BODY>

</HTML>
```

3.2.4 response 的状态行

当服务器对客户请求进行响应时，它发送的首行称做状态行。

状态行包括 3 位数字的状态代码和对状态代码的描述（称做原因短语）。下面列出了对 5 类状态的代码的大概描述：

1yy(1 开头的 3 位数)：主要是实验性质的。

2yy：用来表明请求成功的，例如，状态代码 **200** 可以表明已成功取得了请求的页面。

3yy：用来表明在请求满足之前应采取进一步的行动。

4yy：当浏览器作出无法满足的请求时，返回该状态代码，例如 **404** 表示请求的页面不存在

5yy：用来表示服务器出现问题。例如，**500** 说明服务器内部发生错误。

我们一般不需要修改状态行，在出现问题时，服务器会自动响应，发送相应的状态代码。我们也可以使用 `response` 对象的 `setStatus(int n)` 方法来增加状态行的内容。在下面的例子 14 中，使用 `setStatus(int n)`方法设置响应的状态行。

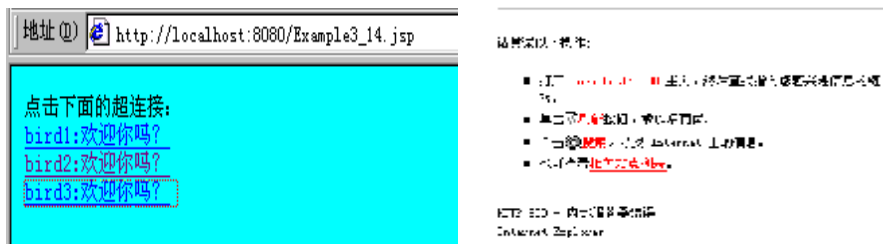


图 3.15 设置响应状态行

例子 14（如图 3.15 所示）

Example3_14.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<HTML>

<BODY bgcolor=cyan><Font size=1>

<P>点击下面的超链接: <BR>

    <A HREF="bird1.jsp"> bird1: 欢迎你吗?

<BR>

    <A HREF="bird2.jsp"> bird2: 欢迎你吗?

<BR>

    <A HREF="bird3.jsp"> bird3: 欢迎你吗?

</FONT>

</BODY>

</HTML>
```

bird1.jsp:

```
<HTML>

<BODY>

    <% response.setStatus(408);

        out.print("    不显示了");

    %>

</BODY>

</HTML>
```

bird2.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

<HTML>

<BODY>

<% response.setStatus(200);

out.println("ok");

%>

</BODY>

</HTML>

bird3.jsp:

<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<%

response.setStatus(500);

%>

</BODY>

</HTML>

下列表 3.1 是状态代码表

表 3.1 状态代码表

状 态 代码	代码说明
	服务器正在升级协议

101	
100	客户可以继续
201	请求成功且在服务器上创建了新的资源
202	请求已被接受但还没有处理完毕
200	请求成功
203	客户端给出的元信息不是发自服务器的
204	请求成功，但没有新信息
205	客户必须重置文档视图
206	服务器执行了部分 get 请求
300	请求的资源有多种表示法
301	资源已经被永久移动到新位置
302	资源已经被临时移动到新位置

303	应答可以在另外一个 URL 中找到
304	Get 方式请求不可用
305	请求必须通过代理来访问
400	请求有语法错误
401	请求需要 HTTP 认证
403	取得了请求但拒绝服务
404	请求的资源不可用
405	请求所用的方法是不允许的
406	请求的资源只能用请求不能接受的内容特性来响应
407	客户必须得到认证
408	请求超时
409	发生冲突，请求不能完成

9	
410	请求的资源已经不可用
411	请求需要一个定义的内容长度才能处理
413	请求太大，被拒绝
414	请求的 URL 太大
415	请求的格式被拒绝
500	服务器发生内部错误，不能服务
501	不支持请求的部分功能
502	从代理和网关接受了不合法的字符
503	HTTP 服务暂时不可用
504	服务器在等待代理服务器应答时发生超时
505	不支持请求的 HTTP 版本

3.3 session 对象

HTTP 协议是一种无状态协议。一个客户向服务器发出请求 (**request**) 然后服务器返回响应 (**respons**), 连接就被关闭了。在服务器端不保留连接的有关信息, 因此当下一次连接时, 服务器已没有以前的连接信息了, 无法判断这一次连接和以前的连接是否属于同一客户。因此, 必须使用会话记录有关连接的信息。

从一个客户打开浏览器连接到服务器, 到客户关闭浏览器离开这个服务器称做一个会话。当一个客户访问一个服务器时, 可能会在这个服务器的几个页面反复连接、反复刷新一个页面或不断地向一个页面提交信息等, 服务器应当通过某种办法知道这是同一个客户, 这就需要 **session**(会话)对象。

3.3.1 session 对象的 Id

当一个客户首次访问服务器上的一个 **JSP** 页面时, **JSP** 引擎产生一个 **secssion** 对象, 这个 **session** 对象调用相应的方法可以存储客户在访问各个页面期间提交的各种信息, 比如, 姓名、号码等信息。这个 **session** 对象被分配了一个 **String** 类型的 **Id** 号, **JSP** 引擎同时将这个 **Id** 号发送到客户端, 存放在客户的 **Cookie** 中。这样, **session** 对象和客户之间就建立起一一对应的关系, 即每个客户都对应着一个 **session** 对象 (该客户的会话), 这些 **session** 对象互不相同, 具有不同的 **Id** 号码。我们已经知道, **JSP** 引擎为每个客户启动一个线程, 也就是说, **JSP** 为每个线程分配不同的 **session** 对象。当客户再访问连接该服务器的其它页面时, 或从该服务器连接到其它服务器再回到该服务器时, **JSP** 引擎不再分配给客户的新 **session** 对象, 而是使用完全相同的一个, 直到客户关闭浏览器后, 服务器端该客户的 **session** 对象被取消, 和客户的会话对应关系消失。当客户重新打开浏览器再连接到该服务器时, 服务器为该客户再创建一个新的 **session** 对象。

在下面的例子 15 中，客户在服务器的三个页面之间进行连接，只要不关闭浏览器，三个页面的 session 对象是完全相同的。客户首先访问 session.jsp 页面，从这个页面再连接到 tom.jsp 页面，然后从 tom.jsp 再连接到 jerry.jsp 页面。

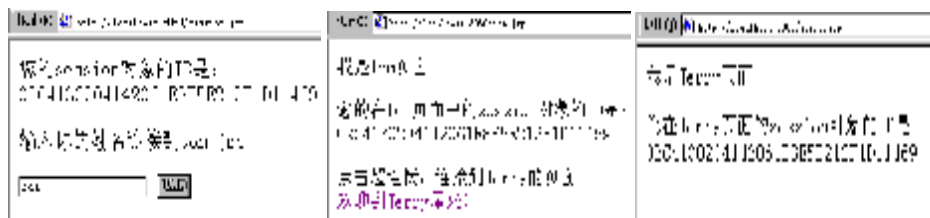


图 3.16 session 对象的 Id

例子 15（如图 3.16 所示）

session.jsp:

```
<%@ page contentType="text/html; charset=GB2312" %>

<HTML>

<BODY>

<P>

    <% String s=session.getId();

    %>

<P> 您的 session 对象的 ID 是:

    <BR>

    <%=s%>
```

<P>输入你的姓名连接到 tom.jsp

<FORM action="tom.jsp" method=post name=form>

<INPUT type="text" name="boy">

<INPUT TYPE="submit" value="送出" name=submit>

</FORM>

</BODY>

</HTML>

tom.jsp:

<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<P>我是 Tom 页面

<% String s=session.getId();

%>

<P> 您的在 Tom 页面中的 session 对象的 ID 是:

<%=s%>

<P> 点击超链接, 连接到 Jerry 的页面。

**
 欢迎到 Jerry 屋来!**

</BODY>

</HTML>

jerry.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<P>我是 Jerry 页面

    <% String s=session.getId();

    %>

<P> 您在 Jerry 页面中的 session 对象的 ID 是:

    <%=s%>

<P> 点击超链接, 连接到 session 的页面。

<A HREF="session.jsp">

<BR>  欢迎到 session 屋来!

</A>

</BODY>

</HTML>
```

3.3.2 session 对象与 URL 重写

session 对象能和客户建立起一一对应关系依赖于客户的浏览器是否支持 Cookie。如果客户端不支持 Cookie, 那么客户在不同网页之间的 session 对象可能是互不相同的, 因为服务器无法将 Id 存放到客户端, 就不能建立 session 对象和客户的一一对应关系。我们将浏览器的 Cookie 设置为禁止后 (选择浏览器菜单→工具→Internet 选项→安全→internet 和本地 intranet→自定义级别→cooker, 将全部选项设置成禁止), 运行上述例子会得到不同的结果。也就是说, “同一客户” 对应了多个 session 对象, 这样服务器就无

法知道在这些页面上访问的实际上是同一个客户。

如果客户的浏览器不支持 **Cookie**，我们可以通过 **URL 重写**来实现 **session** 对象的唯一性。所谓 **URL 重写**，就是当客户从一个页面重新连接到一个页面时，通过向这个新的 **URL** 添加参数，把 **session** 对象的 **Id** 传带过去，这样就可以保障客户在该网站各个页面中的 **session** 对象是 完全相同的。可以使用 **response** 对象 调用 **encodeURL()** 或 **encodeRedirectURL()** 方法实现 **URL 重写**，比如，如果从 **tom.jsp** 页面连接到 **jerry** 页面，首先实现 **URL 重写**：

```
String str=response.encodeRedirectURL("jerry.jsp");
```

然后将连接目标写成`<%=str%>`

如果客户不支持 **Cookie**，在下面的例子 16 中将例子中 15 中的 **session.jsp**、**tom.jsp** 和 **jerry.jsp** 实行 **URL 重写**。

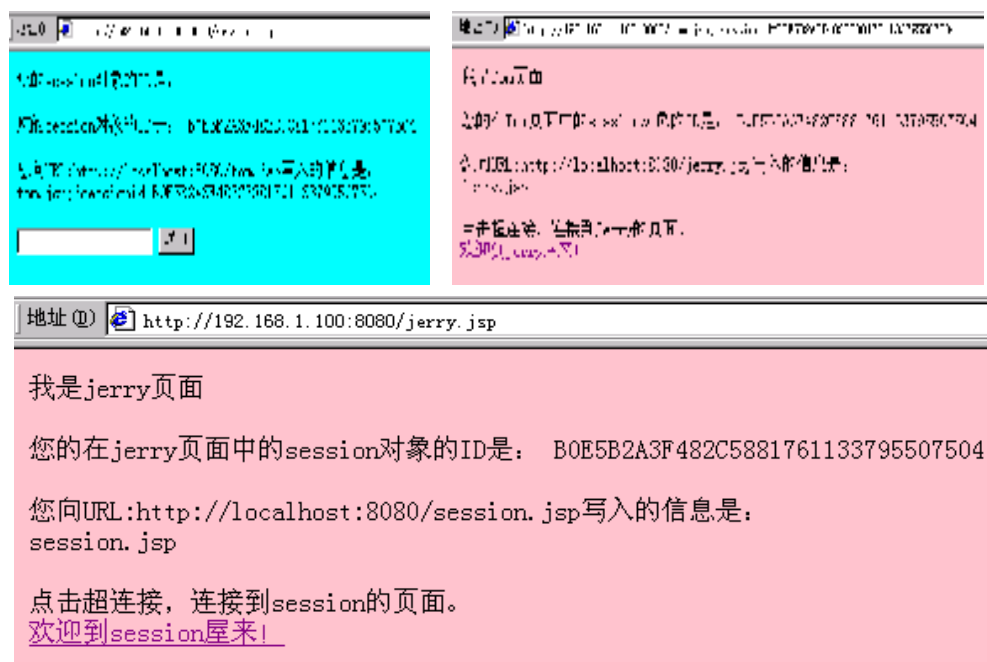


图 3.17 URL 重写

例子 16（如图 3.17 所示）

session.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<HTML>
```

```
<BODY bgcolor=cyan>
```

```
<P> 您的 session 对象的 ID 是:
```

```
<% String s=session.getId();
```

```
String str=response.encodeURL("tom.jsp");
```

```
%>
```

```
<P> 您的 session 对象的 ID 是:
```

```
<%=s%>
```

```
<BR>
```

```
<P>您向 URL:http://localhost:8080/tom.jsp 写入的信息是:
```

```

<%=str%>

<FORM action="<%=str%>" method=post name=form>

    <INPUT type="text" name="boy">

    <INPUT TYPE="submit" value="    送出" name=submit>

</FORM>

</BODY>

</HTML>

```

tom.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=pink>

<P>我是 Tom 页面

    <% String s=session.getId();

        String str=response.encodeRedirectURL("jerry.jsp");

    %>

<P> 您在 Tom 页面中的 session 对象的 ID 是:

    <%=s%>

<P>您向 URL:http://localhost:8080/jerry.jsp 写入的信息是:

<BR>

<%=str%>

<P> 点击超链接, 连接到 Jerry 的页面。

<A HREF="<%=str%>">

```


 欢迎到 Jerry 屋来!

</BODY>

</HTML>

jerry.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<HTML>
```

```
<BODY bgcolor=pink>
```

```
<P>我是 jerry 页面
```

```
    <% String s=session.getId();
```

```
        String str=response.encodeRedirectURL("session.jsp");
```

```
    %>
```

```
<P> 您的在 jerry 页面中的 session 对象的 ID 是:
```

```
    <%=s%>
```

```
<P>您向 URL:http://localhost:8080/session.jsp 写入的信息是:
```

```
<BR>
```

```
    <%=str%>
```

```
<P> 点击超链接, 连接到 session 的页面。
```

```
<A HREF="<%=str%>">
```

```
    <BR> 欢迎到 session 屋来!
```

```
</A>
```

```
</BODY>
```

</HTML>

3.3.3 session 对象的常用方法:

(1) **public void setAttribute(String key, Object obj)**

session 对象类似于散列表, session 对象可以调用该方法将参数 **Object** 指定的对象 **obj** 添加到 session 对象中, 并为添加的对象指定了一个索引关键字, 如果添加的两个对象的关键字相同, 则先前添加的对象被清除。

(2) **public Object getAttribute(String key)**

获取 session 对象含有的关键字是 **key** 的对象。由于任何对象都可以添加到 session 对象中, 因此用该方法取回对象时, 应强制转化为原来的类型。

(3) **public Enumeration getAttributeNames()**

session 对象调用该方法产生一个枚举对象, 该枚举对象使用 **nextElements()** 遍历 session 对象所含有的全部对象。

(4) **public long getCreationTime()**

session 对象调用该方法可以获取该对象创建的时间, 单位是毫秒 (从 1970 年 7 月 1 日午夜起至该对象创建时刻所走过的毫秒数)。

(5) **public long getLastAccessedTime()**

获取当前 session 对象最后一次被操作的时间, 单位是毫秒。

(6) **public int getMaxInactiveInterval()**

获取 session 对象的生存时间。

(7) **public void setMaxInactiveInterval(int n)**

设置 session 对象的生存时间 (单位是秒)

(8) **public void removeAttribute(String key)**

从当前 session 对象中删除关键字是 **key** 的对象。

(9) public String getId()

获取 session 对象的编号。

(10) invalidate

使得 session 无效。

下面的例子 17 中涉及 3 个页面,我们使用 session 对象存储顾客的姓名和购买的商品。

例子 17 (如图 3.18 所示)

Example3_17.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><FONT Size=1>

  <% session.setAttribute("customer"," 顾客");

  %>

  <P> 输入你的姓名连接到第一百货: first.jsp

  <FORM action="first.jsp" method=post name=form>

    <INPUT type="text" name="boy">

    <INPUT TYPE="submit" value="    送出" name=submit>

  </FORM>

</FONT>

</BODY>

</HTML>
```

first.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><FONT Size=1>

    <% String s=request.getParameter("boy");

        session.setAttribute("name",s);

    %>

<P>这里是第一百货

<P>输入你想购买的商品连接到结帐: account.jsp

    <FORM action="account.jsp" method=post name=form>

        <INPUT type="text" name="buy">

        <INPUT TYPE="submit" value="    送出" name=submit>

    </FORM>

</FONT>

</BODY>

</HTML>

```

account.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%! //处理字符串的方法:

    public String getString(String s)

    { if(s==null)

        {s="";

        }

    }

```

```

    try {byte b[]=s.getBytes("ISO-8859-1");

        s=new String(b);

    }

    catch(Exception e)

    {

    }

    return s;

}

%>

<HTML>

<BODY bgcolor=cyan><FONT Size=1>

    <% String s=request.getParameter("buy");

        session.setAttribute("goods",s);

    %>

<BR>

<% String 顾客=(String)session.getAttribute("customer");

    String 姓名=(String)session.getAttribute("name");

    String 商品=(String)session.getAttribute("goods");

    姓名=getString(姓名);

    商品=getString(商品);

%>

<P>这里是结帐处

<P><%=顾客%>的姓名是:

```

```
<%= 姓名%>

<P>您选择购买的商品是:

<%= 商品 %>

</FONT>

</BODY>

</HTML>
```

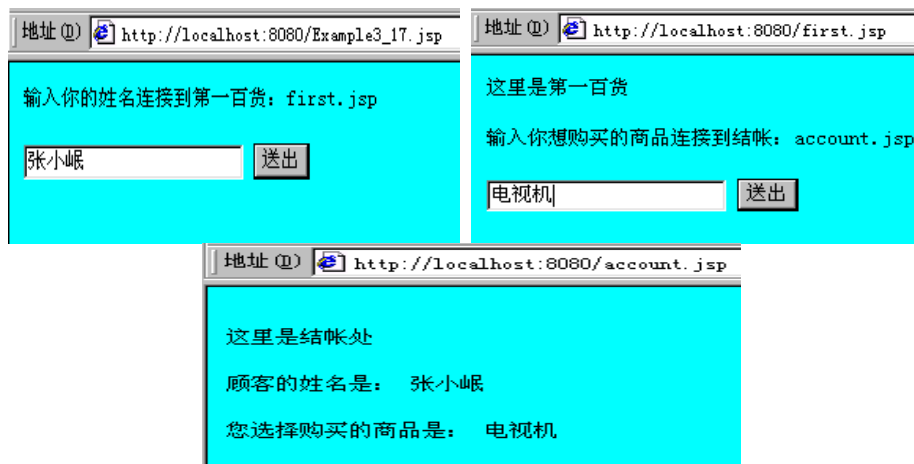


图 3.18 用 session 对象存放信息

下面的例子 18 是个猜数字的小游戏。当客户访问服务器上的 `Example3_18.jsp` 时，随机分配给客户一个 1 到 100 之间的数，然后将这个数字存在客户的 session 对象中。客户

在表单里输入一个数，来猜测分配给自己的那个数字。客户输入一个数字后，提交给 `result.jsp`，该页面负责判断这个数是否和客户的 `session` 对象中存放的那个数字相同，如果相同就连接到 `success.jsp`；如果不相同就连接到 `large.jsp` 或 `small.jsp`，然后，客户在这些页面再重新提交数字到 `result` 页面。

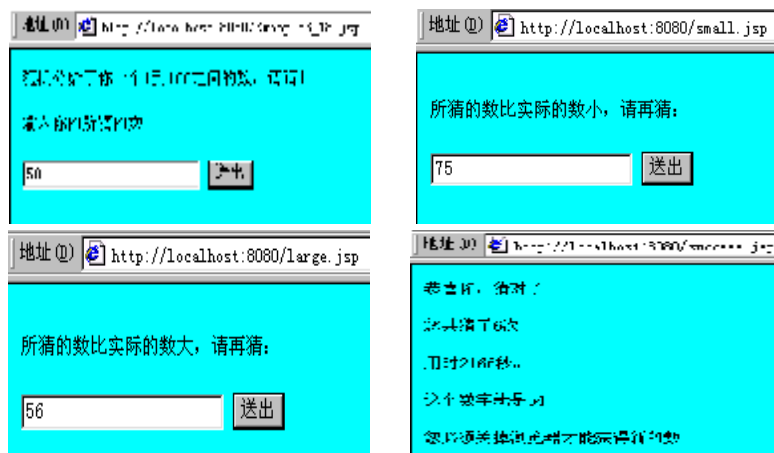


图 3.19 猜数字游戏

例子 18（如图 3.19 所示）

Example3_18.jsp:

```
<%@ page contentType="text/html; charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><FONT Size=1>

<P>随机分给了你一个 1 到 100 之间的数，请猜！
```

```

<%
    int number=(int)(Math.random()*100)+1;
    session.setAttribute("count",new Integer(0));
    session.setAttribute("save",new Integer(number));
%>
<BR>
<P>输入你的所猜的数
    <FORM action="result.jsp" method="post" name=form>
        <INPUT type="text" name="boy" >
        <INPUT TYPE="submit" value="    送出" name="submit">
    </FORM>
</FONT>
</BODY>
</HTML>

```

result.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY bgcolor=cyan><FONT Size=1>
<BR>
<% String str=request.getParameter("boy");
    if(str==null)
        {str="0";

```

```

    }

    int guessNumber=Integer.parseInt(str);

    Integer integer=(Integer)session.getAttribute("save");

    int realnumber=integer.intValue();

    if(guessNumber==realnumber)

        { int n=((Integer)session.getAttribute("count")).intValue();

          n=n+1;

          session.setAttribute("count",new Integer(n));

          response.sendRedirect("success.jsp");

        }

    else if(guessNumber>realnumber)

        { int n=((Integer)session.getAttribute("count")).intValue();

          n=n+1;

          session.setAttribute("count",new Integer(n));

          response.sendRedirect("large.jsp");

        }

    else if(guessNumber<realnumber)

        { int n=((Integer)session.getAttribute("count")).intValue();

          n=n+1;

          session.setAttribute("count",new Integer(n));

          response.sendRedirect("small.jsp");

        }

```

```
%>

</FONT>

</BODY>

</HTML>
```

large.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><FONT Size=1>

<BR>

<P>所猜的数比实际的数大，请再猜：

<FORM action="result.jsp" method="get" name=form >

    <INPUT type="text" name="boy" >

    <INPUT TYPE="submit" value="    送出" name="submit">

</FORM>

</FONT>

</BODY>

</HTML>
```

small.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><FONT Size=1>
```


<P>所猜的数比实际的数小，请再猜：

<FORM action="result.jsp" method="post" name=form>

<INPUT type="text" name="boy" >

<INPUT TYPE="submit" value="送出" name="submit">

</FORM>

</BODY>

</HTML>

success.jsp:

<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan>

<% int count=((Integer)session.getAttribute("count")).intValue();

int num=((Integer)session.getAttribute("save")).intValue();

long startTime=session.getCreationTime();

long endTime=session.getLastAccessedTime();

%>

<P>恭喜你，猜对了

<P>您共猜了<%=count%>次

<P>用时<%=(endTime-startTime)/1000%>秒。

<P>这个数字就是<%=num%>

<P>您必须关掉浏览器才能获得新的数。

</BODY>

</HTML>

3.3.4 计数器

在第 2 章讲述过一个计数器的例子，但那个不断的刷新页面来增加计数器的计数，在下面的例子 19 中，用 `session` 对象禁止客户通过刷新页面增加计例子并不能限制客户通过数。当客户刷新页面时，我们可以使用 `session` 的 `public boolean isNew()` 方法判断是否是一个新的客户，因为客户刷新页面不会改变服务器分配给该客户的 `session` 对象。

例子 19

Examle3_19.jsp:

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.io.*" %>

<HTML>

<BODY>

<%! int number=0;

synchronized void countPeople()

{

if(number==0)

{

try{File f=new File("D:/tomcat","countPeople.txt");

```

        FileInputStream in=new FileInputStream(f);

        DataInputStream dataIn=new DataInputStream(in);

        number=dataIn.readInt();

        number++;

        in.close();dataIn.close();

    }

    catch(FileNotFoundException e)

    { number++;

        try {File f=new File("D:/tomcat","countPeople.txt");

            FileOutputStream out=new FileOutputStream(f);

            DataOutputStream dataOut=new

DataOutputStream(out);

            dataOut.writeInt(number);

            out.close();dataOut.close();

        }

        catch(IOException ee){}

    }

    catch(IOException ee)

    {

    }

}

else

{number++;

```

```

        try{File f=new File("D:/tomcat","countPeople.txt");

            FileOutputStream out=new FileOutputStream(f);

            DataOutputStream dataOut=new DataOutputStream(out);

            dataOut.writeInt(number);

            out.close();dataOut.close();

        }

        catch(FileNotFoundException e){}

        catch(IOException e){}

    }

}

%>

<%

    if(session.isNew())

        {countPeople();

            String str=String.valueOf(number);

            session.setAttribute("count",str);

        }

    %>

<P>您是第<%= (String)session.getAttribute("count") %>个访问本站的人。

<BODY>

<HTML>

```


3.4 application 对象

我们已经知道，当一个客户第一次访问服务器上的一个 JSP 页面时，JSP 引擎创建一个和该客户相对应的 session 对象，**当客户在所访问的网站的各个页面之间浏览时，这个 session 对象都是同一个，直到客户关闭浏览器**，这个 session 对象才被取消；而且不同客户的 session 对象是互不相同的。与 session 对象不同的是 application 对象。服务器启动后，就产生了这个 application 对象。当一个客户访问服务器上的一个 JSP 页面时，JSP 引擎为该客户分配这个 application 对象，**当客户在所访问的网站的各个页面之间浏览时，这个 application 对象都是同一个，直到服务器关闭**，这个 application 对象才被取消。与 session 对象不同的是，所有客户的 application 对象是相同的一个，即所有的客户共享这个内置的 application 对象。我们已经知道，JSP 引擎为每个客户启动一个线程，也就是说，这些线程共享这个 application 对象。

3.4.1 application 对象的常用方法

(1) public void setAttribute(String key, Object obj)

application 对象可以调用该方法将参数 Object 指定的对象 obj 添加到 application 对象中，并为添加的对象指定了一个索引关键字，如果添加的两个对象的关键字相同，则先前添加对象被清除。

(2) public Object getAttribute(String key)

获取 application 对象含有的关键字是 key 的对象。由于任何对象都可以添加到 application 对象中，因此用该方法取回对象时，应强制转化为原来的类型。

(3) public Enumeration getAttributeNames()

application 对象调用该方法产生一个枚举对象，该枚举对象使用 nextElements() 遍历 application 对象所含有的全部对象。

(4) **public void removeAttribute(String key)**

从当前 **application** 对象中删除关键字是 **key** 的对象。

(5) **public String getServletInfo()**

获取 **Servlet** 编译器的当前版本的信息。

由于 **application** 对象对所有的客户都是相同的,任何客户对该对象中存储的数据的改变都会影响到其他客户,因此,在某些情况下,对该对象的操作需要实现同步处理。

在下面的例子中,我们用 **application** 对象实现一个计数器,将计数存放在 **application** 对象中,每个客户对该对象中“计数”的改变都会影响到其他客户。

注:有些服务器不直接支持使用 **application** 对象,必须用 **ServletContext** 类声明这个对象,再使用 **getServletContext()** 方法对这个 **application** 对象进行初始化。

例子 20

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<%!

    synchronized void countPeople()

    { ServletContext application=getServletContext();

      Integer number=(Integer)application.getAttribute("Count");

      if(number==null)

      { number=new Integer(1);

        application.setAttribute("Count",number);

      }

    }
```

```

else

    { number=new Integer(number.intValue()+1);

      application.setAttribute("Count",number);

    }

}

%>

<% if(session.isNew())

    { countPeople();

      Integer myNumber=(Integer)application.getAttribute("Count");

      session.setAttribute("MyCount",myNumber);

    }

%>

<P><P>您是第

    <%int a=((Integer)session.getAttribute("MyCount")).intValue();

    %>

    <%=a%>

个访问本站的客户。

</BODY>

</HTML>

```

3.4.2 用 application 制作留言板

在例子 21 中，客户通过 submit.jsp 向 messagePane.jsp 页面提交姓名、留言标题和留言内容，messagePane.jsp 页面获取这些内容后，用同步方法将这些内容添加到一个向量

中，然后将这个向量再添加到 `application` 对象中。当用户点击查看留言版时，`showMessage.jsp` 负责显示所有客户的留言内容，即从 `application` 对象中取出向量，然后遍历向量中存储的信息。

在这里我们使用了向量这种数据结构，Java 的 `java.util` 包中的 `Vector` 类负责创建一个向量对象。如果你已经学会使用数组，那么很容易就会使用向量。当我们创建一个向量时不用象数组那样必须要给出数组的大小。向量创建后，例如，`Vector a=new Vector();` `a` 可以使用 `add(Object o)` 把任何对象添加到向量的末尾，向量的大小会自动的增加。可以使用 `add(int index ,Object o)` 把一个对象追加到该向量的指定位置。向量 `a` 可以使用 `elementAt(int index)` 获取指定索引处的向量的元素（索引初始位置是 0）；`a` 可以使用方法 `size()` 获取向量所含有的元素的个数。另外，与数组不同的是向量的元素类型不要求一致。需要注意的是，虽然你可以把任何一种 Java 的对象放入一个向量，但是，当从向量中取出一个元素时，必须使用强制类型转化运算符将其转化为原来的类型。

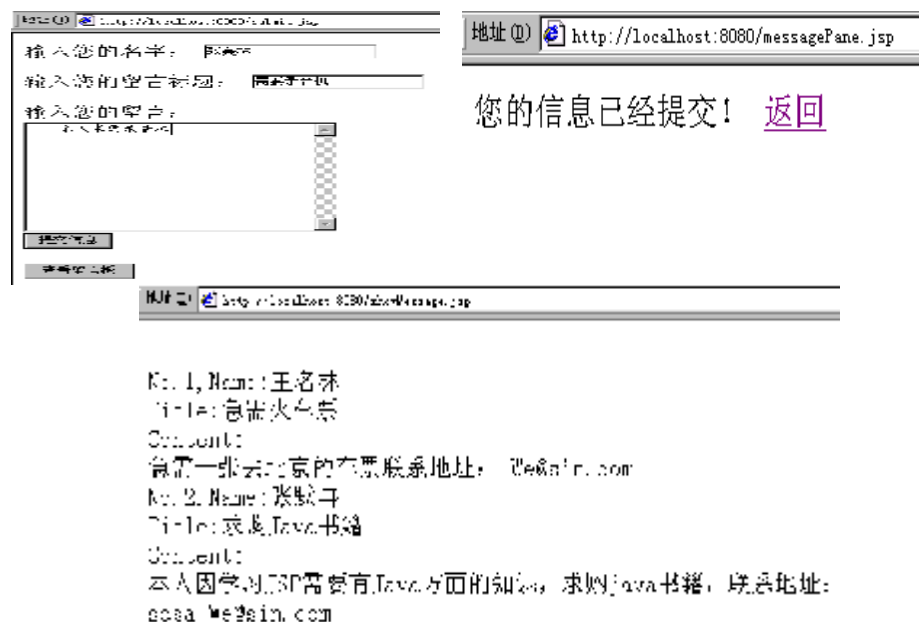


图 3.20 留言板

例子 21 (如图 3.20 所示)

submit.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<FORM action="messagePane.jsp" method="post" name="form">

  <P>  输入您的名字:

  <INPUT type="text" name="peopleName">

  <BR>

  <P>  输入您的留言标题:

  <INPUT type="text" name="Title">

  <BR>

  <P>  输入您的留言:

  <BR>
```

```

<TEXTAREA name="messages" ROWs="10" COLS=36 WRAP="physical">

</TEXTAREA>

<BR>

<INPUT type="submit" value=" 提交信息" name="submit">

</FORM>

<FORM action="showMessage.jsp" method="post" name="form1">

  <INPUT type="submit" value=" 查看留言板" name="look">

</FORM>

</BODY>

</HTML>

```

messagePane.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.util.*" %>

<HTML>

<BODY>

  <%! Vector v=new Vector();

    int i=0; ServletContext application;

    synchronized void sendMessage(String s)

    { application=getServletContext();

      i++;

      v.add("No."+i+", "+s);

      application.setAttribute("Mess",v);

```

```

    }

%>

<% String name=request.getParameter("peopleName");

    String title=request.getParameter("Title");

    String messages=request.getParameter("messages");

    if(name==null)

        {name="guest"+(int)(Math.random()*10000);

        }

    if(title==null)

        {title="        无标题";

        }

    if(messages==null)

        {messages="        无信息";

        }

    String

s="Name:"+name+"#"+"Title:"+title+"#"+"Content:"+"<BR>"+messages;

    sendMessage(s);

    out.print("        您的信息已经提交! ");

%>

<A HREF="submit.jsp" > 返回

</BODY>

</HTML>

```

showMessage.jsp :

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.util.*" %>

<HTML>

<BODY>

    <% Vector v=(Vector)application.getAttribute("Mess");

        for(int i=0;i<v.size();i++)

            { String message=(String)v.elementAt(i);

                StringTokenizer fenxi=new StringTokenizer(message,"#");

                while(fenxi.hasMoreTokens())

                    { String str=fenxi.nextToken();

                        byte a[]=str.getBytes("ISO-8859-1");

                        str=new String(a);

                        out.print("<BR>" +str);

                    }

            }

    %>

</BODY>

</HTML>
```

3.5 out 对象

out 对象是一个输出流，用来向客户端输出数据。在前面的许多例子里曾多次使用 **out** 对象进行数据的输出。**out** 对象可调用如下的方法用于各种数据的输出，例如：

out.print(Boolean), **out.println(boolean)** : 用于输出一个布尔值。

out.print(char), out.println(char)	: 输出一个字符。
out.print(double), out.println(double)	: 输出一个双精度的浮点数。
out.print(float), out.println(float)	: 用于输出一个单精度的浮点数。
out.print(long), out.println(long)	: 输出一个长整型数据。
out.print(String), out.println(String)	: 输出一个字符串对象的内容。
out.newLine()	: 输出一个换行符
out.flush()	: 输出缓冲区里的内容
out.close()	: 关闭流

下面的例子 22 使用 **out** 对象向客户输出包括表格等信息。

例子 22 (如图 3.21 所示)

Example3_22.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.util.*" %>

<HTML>

<BODY>

<% int a=100;long b=300;boolean c=true;

    out.println("<H1> 这是标题 1 字体的大小</HT1>");

    out.println("<H2> 这是标题 2 字体的大小</HT2>");

    out.print("<BR>");

    out.println(a); out.println(b); out.println(c);

%>

```

<Center>

<p> 以下是一个表格

<%out.print("");

out.println("<Table Border >");

out.println("<TR >");

out.println("<TH width=80>"+ " 姓名"+"</TH>");

out.println("<TH width=60>"+ " 性别"+"</TH>");

out.println("<TH width=200>"+ " 出生日期"+"</TH>");

out.println("</TR>");

out.println("<TR >");

out.println("<TD >"+ " 刘甲一"+"</TD>");

out.println("<TD >"+ " 男"+"</TD>");

out.println("<TD >"+ "1978 年 5 月"+"</TD>");

out.println("</TR>");

out.println("<TR>");

out.println("<TD >"+ " 林 霞"+"</TD>");

out.println("<TD >"+ " 女"+"</TD>");

out.println("<TD >"+ "1979 年 8 月"+"</TD>");

out.println("<TD width=100>"+ " 这是表格"+"</TD>");

out.println("</TR>");

out.println("</Table>");

out.print("") ;

%>

</Center>

</BODY>

</HTML>

下面的例子 23 根据当前时间向客户发送提示信息，比如，如果是 26 日，就发送病毒易发作的信息等。

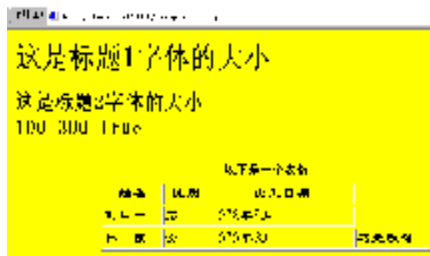


图 3.21 用 out 对象输出各种信息

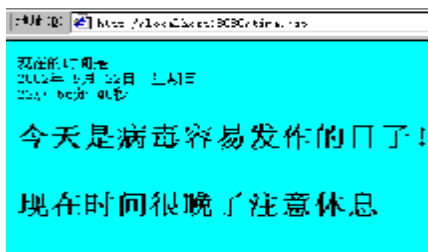


图 3.22 发送提示信息

例子 23（如图 3.22 所示）

time.jsp:

```
<%@ page contentType="text/html; charset=GB2312" %>
```

```
<%@ page import="java.util.*"%>
```

```
<%! public String getDayWeek(int n)
```

```
{ String week[]={ " 星期日", "星期一", "星期二", "星期三", "星期四", "星期五",
```

星期六"};

return week[n];

}

%>

<HTML>

<BODY bgcolor=cyan>

<% Calendar calendar=Calendar.getInstance(); //创建一个日历对象。

calendar.setTime(new Date());// 用当前时间初始化日历时间。

String 年=String.valueOf(calendar.get(Calendar.YEAR)),

月=String.valueOf(calendar.get(Calendar.MONTH)+1),

日=String.valueOf(calendar.get(Calendar.DAY_OF_MONTH)),

星期=getDayWeek(calendar.get(Calendar.DAY_OF_WEEK)-1);

int hour=calendar.get(Calendar.HOUR_OF_DAY),

minute=calendar.get(Calendar.MINUTE),

second=calendar.get(Calendar.SECOND);

%>

<P>现在的时间是

<%=年%>年

<%=月%>月

<%=日%>日

<%=星期%>

<%=hour%>点

<%=minute%>分

<%=second%>秒

<% if(日.equals("26"))

{out.print("
<H2> 今天是病毒容易发作的日子! </H2>");

}

if(hour>=22)

{ out.print("
<H2> 现在时间很晚了注意休息</H2>");

}

%>

</BODY>

</HTML>

第4章 JSP 中的文件操作

有时服务器需要将客户提交的信息保存到文件或根据客户的要求将服务器上的文件的内容显示到客户端。**JSP** 通过 **Java** 的输入输出流来实现文件的读写操作。

4.1 File 类

File 类的对象主要用来获取文件本身的一些信息，例如文件所在的目录、文件的长度、文件读写权限等，不涉及对文件的读写操作。

创建一个 **File** 对象的构造方法有 3 个：

File(String filename);

File(String directoryPath,String filename);

File(File f, String filename);

其中，**filename** 是文件名字，**directoryPath** 是文件的路径，**f** 是指定成一个目录的文件。

使用 **File(String filename)** 创建文件时，该文件被认为是与当前应用程序在同一目录中，由于 **JSP** 引擎是在 **bin** 下启动执行的，所以该文件被认为在下列目录中：

D:\Tomcat\jakarta-tomcat-4.0\bin。

4.1.1 获取文件的属性

经常使用 **File** 类的下列方法获取文件本身的一些信息：

1. **public String getName():** 获取文件的名字。
2. **public boolean canRead():** 判断文件是否是可读的。

3. `public boolean canWrite()`: 判断文件是否可被写入。
4. `public boolean exists()`: 判断文件是否存在。
5. `public long length()`: 获取文件的长度（单位是字节）。
6. `public String getAbsolutePath()`: 获取文件的绝对路径。
7. `public String getParent()`: 获取文件的父目录。
8. `public boolean isFile()`: 判断文件是否是一个正常文件，而不是目录。
9. `public boolean isDirectory()`: 判断文件是否是一个目录。
10. `public boolean isHidden()` : 判断文件是否是隐藏文件。
11. `public long lastModified()` : 获取文件最后修改的时间（时间是从 1970 年午夜至文件最后修改时刻的毫秒数）。

在下面的例子 1 中，我们使用上述的一些方法，获取某些文件的信息。



图 4.1 使用 File 对象获取文件的属性

例子 1（效果如图 4.1 所示）

Example4_1.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.io.*"%>

<HTML>

<BODY bgcolor=cyan><Font Size=1>

    <%File f1=new

        File("D:\\Tomcat\\jakarta-tomcat-4.0\\webapps\\root","Example3_1.jsp");

        File f2=new File("jasper.sh");

    %>

    <P> 文件 Example3_1.jsp 是可读的吗?

    <%=f1.canRead()%>

    <BR>

    <P> 文件 Example3_1.jsp 的长度:

    <%=f1.length()%>  字节

    <BR>

    <P> jasper.sh 是目录吗?

    <%=f2.isDirectory()%>

    <BR>

    <P>Example3_1.jsp 的父目录是:

    <%=f1.getParent()%>
```


<P>jasper.sh 的绝对路径是:

<%=f2.getAbsolutePath()%>

</BODY>

</HTML>

4.1.2 创建目录

(1) 创建目录

File 对象调用方法: **public boolean mkdir()** 创建一个目录, 如果创建成功返回 **true**, 否则返回 **false**(如果该目录已经存在将返回 **false**)。

在下面的例子 2 中, 我们在 **Root** 下创建一个名字是 **Students** 的目录

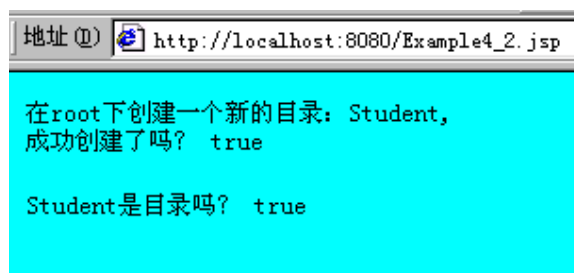


图 4.2 创建目录

例子 2 (效果如图 4.2 所示)

Example4_2.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.io.*"%>

<HTML>

<BODY><Font Size=2>

    <% File dir=new
File("D:/Tomcat/jakarta-tomcat-4.0/webapps/root","Students");

    %>

<P> 在 root 下创建一个新的目录: Student,<BR>成功创建了吗?

    <%=dir.mkdir()%>

<P> Student 是目录吗?

    <%=dir.isDirectory()%>

</Font>

</BODY>

</HTML>

```

(2) 列出目录中的文件

如果 **File** 对象是一个目录, 那么该对象可以调用下述方法列出该目录下的文件和子目录:

public String[] list(): 用字符串形式返回目录下的全部文件,

public File [] listFiles(): 用 **File** 对象形式返回目录下的全部文件。

在下面的例子 3 中, 输出了 **Root** 下的全部文件中的 5 个和全部子目录。



```

地址 http://localhost:8080/Example4_3.jsp

列出root下的5个长度大于1000字节的文件和全部目录:
目录有:
D:\Tomcat\jakarta-tomcat-4.0\webapps\root\WEB-INF
D:\Tomcat\jakarta-tomcat-4.0\webapps\root\Students
D:\Tomcat\jakarta-tomcat-4.0\webapps\root\Myfile

5个长度大于1000 字节的文件名:
D:\Tomcat\jakarta-tomcat-4.0\webapps\root\index.html
D:\Tomcat\jakarta-tomcat-4.0\webapps\root\tomcat.gif
D:\Tomcat\jakarta-tomcat-4.0\webapps\root\jakarta-banner.gif

```

例子 3（效果如图 4.3 所示）

Example4_3.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.io.*"%>

<HTML>

<BODY><Font Size=2>

    <% File dir=new File("D:/Tomcat/jakarta-tomcat-4.0/webapps/root");

        File file[]=dir.listFiles();

    %>

    <P>  列出 root 下的 5 个长度大于 1000 字节的文件和全部目录:

    <BR>  目录有:

    <% for(int i=0;i<file.length;i++)

        {if(file[i].isDirectory())

            out.print("<BR>" +file[i].toString());
```

```

        }

        %>

<P> 5 个长度大于 1000 字节的文件名字:

        <% for(int i=0,number=0;(i<file.length)&&(number<=5);i++)

            {if(file[i].length()>=1000)

                {out.print("<BR>" +file[i].toString());

                    number++;

                }

            }

        %>

</Font>

</BODY>

</HTML>

```

(3) 列出指定类型的文件

我们有时需要列出目录下指定类型的文件，比如.jsp、.txt 等扩展名的文件。可以使用 **File** 类的下述两个方法，列出指定类型的文件，

public String[] list(FilenameFilter obj); 该方法用字符串形式返回目录下的指定类型的所有文件。

public File [] listFiles(FilenameFilter obj); 该方法用 **File** 对象返回目录下的指定类型所有文件。

FilenameFile 是一个接口，该接口有一个方法：

public boolean accept(File dir,String name) ;

当向 **list** 方法传递一个实现该接口的对象时，**list** 方法在列出文件时，将让该文件调用 **accept** 方法检查该文件是否符合 **accept** 方法指定的目录和文件名字要求。

在下面的例子 4 中，列出 **Root** 目录下的部分 **JSP** 文件的名字。

例子 4

Example4_4.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<%! class FileJSP implements FilenameFilter

{ String str=null;

    FileJSP(String s)

    {str="."+s;

    }

    public boolean accept(File dir,String name)

    { return name.endsWith(str);

    }

}

%>

<P>下面列出了服务器上的一些 jsp 文件

<% File dir=new File("d:/Tomcat/Jakarta-tomcat-4.0/webapps/root/");

FileJSP file_jsp=new FileJSP("jsp");

String file_name[]=dir.list(file_jsp);
```

```

for(int i=0;i<5;i++)

    {out.print("<BR>" +file_name[i]);

    }

%>

```

4.1.3 删除文件和目录

File 对象调用方法 **public boolean delete()** 可以删除当前对象代表的文件或目录，如果 **File** 对象表示的是一个目录，则该目录必须是一个空目录，删除成功返回 **true**。

下面的例子 5 删除 **Root** 目录下的 **A.java** 文件和 **Students** 目录。

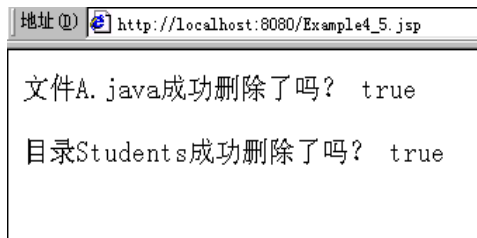


图 4.4 删除目录或文件

例子 5（效果如图 4.4 所示）

Example4_5.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<HTML>

<BODY>

```

```
<%File f=new File("d:/Tomcat/Jakarta-tomcat-4.0/webapps/root/","A.java");  
  
File dir=new File("d:/Tomcat/Jakarta-tomcat-4.0/webapps/root","Students");  
  
boolean b1=f.delete();  
  
boolean b2=dir.delete();  
  
%>  
<P>文件 A.java 成功删除了吗?  
  
<%=b1%>  
  
<P>目录 Students 成功删除了吗?  
  
<%=b2%>  
  
</BODY>  
  
</HTML>
```

4.2 使用字节流读写文件

Java 的 I/O 流提供一条通道程序，可以使用这条通道把源中的数据送给目的地。把输入流的指向称做源，程序从指向源的输入流中读取源中的数据。而输出流的指向是数据要去的一个目的地，程序通过向输出流中写入数据把信息传递到目的地，如下图 4.5、4.6 所示。

java.io 包提供大量的流类。所有字节输入流类都是 InputStream（输入流）抽象类的子类，而所有字节输出流都是 OutputStream（输出流）抽象类的子类。

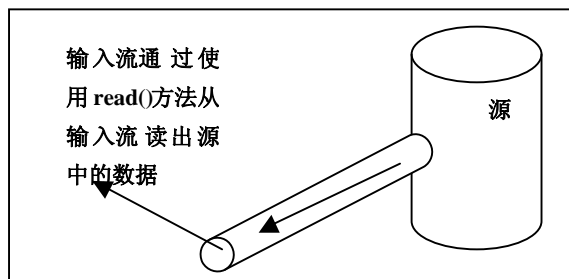


图 4.5 输入流示意图

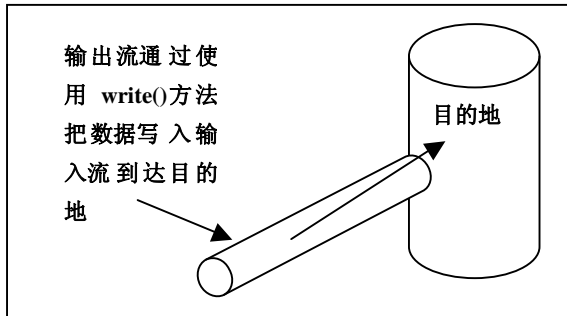


图 4.6 输出流示意图

InputStream 类的常用方法:

- 2 **int read():** 输入流调用该方法从源中读取单个字节的数据, 该方法返回字节值 (0~255 之间的一个整数), 如果未读出字节就返回-1。
 - 2 **int read(byte b[]):** 输入流调用该方法从源中试图读取 b.length 个字节到 b 中, 返回实际读取的字节数目。如果到达文件的末尾, 则返回-1。
 - 2 **int read(byte b[], int off, int len):** 输入流调用该方法从源中试图读取 len 个字节到 b 中, 并返回实际读取的字节数目。如果到达文件的末尾, 则返回-1, 参数 off 指定从字节数组的某个位置开始存放读取的数据。

2 **void close():** 输入流调用该方法关闭输入流。

2 **long skip(long numBytes):** 输入流调用该方法跳过 **numBytes** 个字节，并返回实际跳过的字节数目。

OutputStream 类的常用方法：

2 **void write(int n):** 输出流调用该方法向输出流写入单个字节。

2 **void write(byte b[]):** 输出流调用该方法向输出流写入一个字节数组。

2 **void write(byte b[],int off,int len):** 从给定字节数组中起始于偏移量 **off** 处取 **len** 个字节写到输出流。

2 **void close():** 关闭输出流。

4.2.1 FileInputStream 和 FileOutputStream 类

FileInputStream 该类是从 **InputStream** 中派生出来的简单输入类。该类的所有方法都是从 **InputStream** 类继承来的。为了创建 **FileInputStream** 类的对象，用户可以调用它的构造方法。下面显示了两个构造方法：

```
FileInputStream (String name);
```

```
FileInputStream(File file);
```

第一个构造方法使用给定的文件名 **name** 创建一个 **FileInputStream** 对象，第二个构造方法使用 **File** 对象创建 **FileInputStream** 对象。参数 **name** 和 **file** 指定的文件称做输入流的源，输入流通过调用 **read** 方法读出源中的数据。

FileInputStream 文件输入流打开一个到达文件的输入流（源就是这个文件，输入流指

向这个文件) 例如, 为了读取一个名为 myfile.dat 的文件, 建立一个文件输入流对象, 如下所示:

```
FileInputStream istream = new FileInputStream("myfile.dat");
```

文件输入流构造方法的另一种格式是允许使用文件对象来指定要打开哪个文件, 如下所示:

```
FileInputStream (File file) ;
```

例如, 下面这段代码使用文件输入流构造方法来建立一个文件输入流, 如下所示:

```
File f = new File("myfile.dat");  
FileInputStream istream = new FileInputStream(f);
```

当您使用文件输入流构造方法建立通往文件的输入流时, 可能会出现错误 (也被称为异常)。例如, 您试图要打开的文件可能不存在。当出现 I/O 错误, Java 生成一个出错信号, 它使用一个 `IOException`(IO 异常)对象来表示这个出错信号。程序必须使用一个 `catch` (捕获) 块检测并处理这个异常。例如, 为了把一个文件输入流对象与一个文件关联起来, 使用类似于下面所示的代码:

```
try {FileInputStream ins = new FileInputStream("myfile.dat"); // 读取输入流  
}
```

```
catch (IOException e )  
  
    { // 文件 I/O 错误  
  
        System.out.println("File read error: " +e );  
  
    }
```

与 `FileInputStream` 类相对应的类是 `FileOutputStream` 类。`FileOutputStream` 提供了基本的文件写入能力。除了从 `FileOutputStream` 类继承来的方法以外，`FileOutputStream` 类还有两个常用的构造方法，这两个构造方法如下所示：

```
FileOutputStream (String name);  
  
FileOutputStream(File file);
```

第一个构造方法使用给定的文件名 `name` 创建一个 `FileOutputStream` 对象。第二个构造方法使用 `File` 对象创建 `FileOutputStream` 对象。参数 `name` 和 `file` 指定的文件称做输出流的目的地，通过向输出流中写入数据把信息传递到目的地。创建输出流对象也能发生 `IOException` 异常，必须在 `try`、`catch` 块语句中创建输出流对象。

注：使用 `FileInputStream` 的构造方法：`FileInputStream (String name)` 创建一个输入流时，以及使用 `FileOutputStream` 的构造方法：`FileOutputStream (String name)` 创建一个输出流时要保证文件和当前应用程序在同一目录下。由于 JSP 引擎是在 `bin` 下启动执行的，所以文件必须在 `bin` 目录中。

4.2.2 `BufferedInputStream` 和 `BufferedOutputStream` 类

为了提高读写的效率，`FileInputStream` 流经常和 `BufferedInputStream` 流配合使用，`FileOutputStream` 流经常和 `BufferedOutputStream` 流配合使用。

`BufferedInputStream` 的一个常用的构造方法是：

```
BufferedInputStream(InputStream in);
```

该构造方法创建缓存输入流，该输入流的指向是一个输入流。当我们要读取一个文件，比如 A.txt 时，可以先建立一个指向该文件的文件输入流：

```
FileInputStream in=new FileInputStream( "A.txt");
```

然后再创建一个指向文件输入流 in 的输入缓存流（就好象把两个输入水管接在一起）：

```
BufferedInputStream buffer=new BufferedInputStream(in);
```

这时，我们就可以让 buffer 调用 read 方法读取文件的内容，buffer 在读取文件的过程中，会进行缓存处理，增加读取的效率。

同样，当要向一个文件，比如 B.txt，写入字节时，可以先建立一个指向该文件的文件输出流：

```
FileOutputStream out=new FileOutputStream( "B.txt");
```

然后再创建一个指向输出流 out 的输出缓存流：

```
BufferedOutputStream buffer=new BufferedOutputStream(out);
```

这时，buffer 调用 write 方法向文件写入内容时会进行缓存处理，增加写入的效率。需要注意的是，写入完毕后，须调用 flush 方法将缓存中的数据存入文件。

在下面的例子 6 中，服务器将若干内容写入一个文件，然后读取这个文件，并将文件的内容显示给客户。

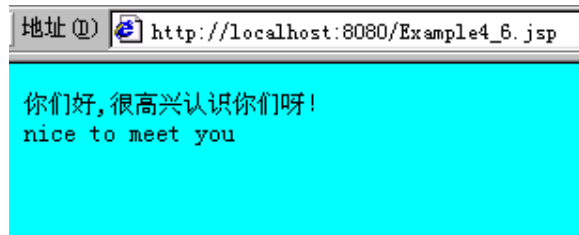


图 4.7 使用字节流读写文件

例子 6（效果如图 4.7 所示）

Example4_6.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<HTML>

<BODY bgcolor=cyan><FONT size=1>

  <%File dir=new

File("d:/Tomcat/Jakarta-tomcat-4.0/webapps/root","Students");

  dir.mkdir();

  File f=new File(dir,"hello.txt");
```

```

try{

    FileOutputStream outfile=new FileOutputStream(f);

    BufferedOutputStream bufferout=new BufferedOutputStream(outfile);

    byte b[]="    你们好,很高兴认识你们呀! <BR>nice to meet you".getBytes();

    bufferout.write(b);

    bufferout.flush();

    bufferout.close();

    outfile.close();

    FileInputStream in=new FileInputStream(f);

    BufferedInputStream bufferin=new BufferedInputStream(in);

    byte c[]=new byte[90];

    int n=0;

    while((n=bufferin.read(c))!=-1)

        { String temp=new String(c,0,n);

            out.print(temp);

        }

    bufferin.close();

    in.close();

    }

catch(IOException e)

    {

    }

%>

```

</BODY>

</HTML>

4.3 使用字符流读写文件

上面我们学习了使用字节流读写文件，字节流不能直接操作 Unicode 字符，所以 Java 提供了字符流，由于汉字在文件中占用 2 个字节，如果使用字节流，读取不当会出现乱码现象，采用字符流就可以避免这个现象，在 Unicode 字符中，一个汉字被看作一个字符。

所有字符输入流类都是 Reader（输入流）抽象类的子类，而所有字符输出流都是 Writer（输出流）抽象类的子类。

Reader 类中常用方法：

2 int read(): 输入流调用该方法从源中读取一个字符，该方法返回一个整数（0~65535 之间的一个整数，Unicode 字符值），如果未读出字符就返回-1。

2 int read(char b[]): 输入流调用该方法从源中读取 b.length 个字符到字符数组 b 中，返回实际读取的字符数目。如果到达文件的末尾，则返回-1。

2 int read(char b[], int off, int len): 输入流调用该方法从源中读取 len 个字符并存放到字符数组 b 中，返回实际读取的字符数目。如果到达文件的末尾，则返回-1。其中，off 参数指定 read 方法从符数组 b 中的什么地方存放数据。

2 void close(): 输入流调用该方法关闭输入流。

2 long skip(long numBytes): 输入流调用该方法跳过 numBytes 个字符，并返回实际跳过的字符数目。

Writer 类中 常用方法：

2 **void write(int n):** 向输入流写入一个字符。

2 **void write(byte b[]):** 向输入流写入一个字符数组。

2 **void write(byte b[],int off,int length):** 从给定字符数组中起始于偏移量 off 处取 len 个字符写到输出流。

2 **void close():** 关闭输出流。

4.3.1 FileReader 和 FileWriter 类

FileReader 该类是从 **Reader** 中派生出来的简单输入类。该类的所有方法都是从 **Reader** 类继承来的。为了创建 **FileReader** 类的对象，用户可以调用它的构造方法。下面显示了两个构造方法：

```
FileReader (String name) ;
```

```
FileReader (File file) ;
```

第一个构造方法使用给定的文件名 **name** 创建一个 **FileReader** 对象，第二个构造方法使用 **File** 对象创建 **FileReader** 对象。参数 **name** 和 **file** 指定的文件称做输入流的源，输入流通过调用 **read** 方法读出源中的数据。

与 **FileReader** 类相对应的类是 **FileWriter** 类。**FileWriter** 提供了基本的文件写入能力。除了从 **FileWriter** 类继承来的方法以外，**FileWriter** 类还有两个常用的构造方法，这两个构造方法如下所示：

```
FileWriter (String name) ;
```

```
FileWriter (File file);
```


第一个构造方法使用给定的文件名 **name** 创建一个 **FileWriter** 对象。第二个构造方法使用 **File** 对象创建 **FileWriter** 对象。参数 **name** 和 **file** 指定的文件称做输出流的目的地，通过向输出流中写入数据把信息传递到目的地。创建输入、输出流对象能发生 **IOException** 异常，必须在 **try**、**catch** 块语句中创建输入、输出流对象。

4.3.2 **BufferedReader** 和 **BufferedWriter** 类

为了提高读写的效率，**FileReader** 流经常和 **BufferedReader** 流配合使用；**FileWriter** 流经常和 **BufferedWriter** 流配合使用。**BufferedReader** 流还可以使用方法 **String readLine()** 读取一行；**BufferedWriter** 流还可以使用方法 **void write(String s,int off,int length)** 将字符串 **s** 的一部分写入文件，使用 **newLine()** 向文件写入一个行分隔符。

在下面的例子 7 中，服务器将若干内容写入一个文件，然后读取这个文件，并将文件的内容显示给客户。

例子 7（效果如图 4.8 所示）

Example4_7.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import ="java.io.*" %>
```

```
<HTML>
```

```
<BODY>
```

```
<%
```

```
File dir=new File("d:/Tomcat/Jakarta-tomcat-4.0/webapps/root","Students");
```

```
dir.mkdir();
```

```
File f=new File(dir,"apple.txt");
```

```

try{FileWriter outfile=new FileWriter(f);

    BufferedWriter bufferout=new BufferedWriter(outfile);

    bufferout.write("    你好: ");

    bufferout.newLine();

    bufferout.write("    好久不见了，近来在忙什么呢? ");

    bufferout.newLine();

    bufferout.write("    欢迎来玩");

    bufferout.newLine();

    bufferout.write("2002    年 8 月 8 日");

    bufferout.flush();

    bufferout.close();

    outfile.close();

    FileReader in=new FileReader(f);

    BufferedReader bufferin=new BufferedReader(in);

    String str=null;

    while((str=bufferin.readLine())!=null)

        {out.print("<BR>" +str);

        }

    bufferin.close();

    in.close();

}

catch(IOException e)

{

```

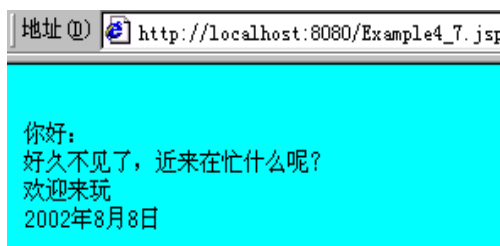


图 4.8 使用字节流读写文件

```
    }  
    %>  
</BODY>  
</HTML>
```

下面的例子 8 是一个网络英语单词测试，客户首先访问 Example4_8.jsp 页面，该页面通过输入流技术将一个文本文件读入，这个文本的每一行是一个英语单词选择练习，如下所示：

English.txt:

```
#Take an umbrella with you in case___#it will rain#it rains#it raining#it  
rained#B#  
  
#He is no longer the honest man___he was#who#whom#which#that#D#  
  
#During the recession, thousands of workers were____#laid on#laid down#laid out#laid  
off#B#  
  
#The building casts a large__on the ground#shadow#shade#shanpoo#shawl#D#  
  
#The driver came close to ___killed for speeding#having been#have  
been#be#being#A#
```

然后把文本的每一行都存入客户的 session 对象中，并用行号做关键字，需要时可通过这个关键字在 session 对象中查找这一行，同时，将一个分数属性也存入客户的 session 对象中。

客户点击“开始练习”按钮进入练习页面 Exercise.jsp，该页面从客户的 session 对象中首先取出第一行，然后通过 StringTokenizer 类将该行的语言符号取出。将第一

个语言符号作为试题，第 2 到第 4 个语言符号作为选择，第 5 个语言符号是该试题的答案，把该题的答案存入客户的 session 对象中，将来根据题号在 session 对象中检索这个答案。当客户点击该页面中的“提交答案”按钮后再从 session 对象取出下一行，依次类推。

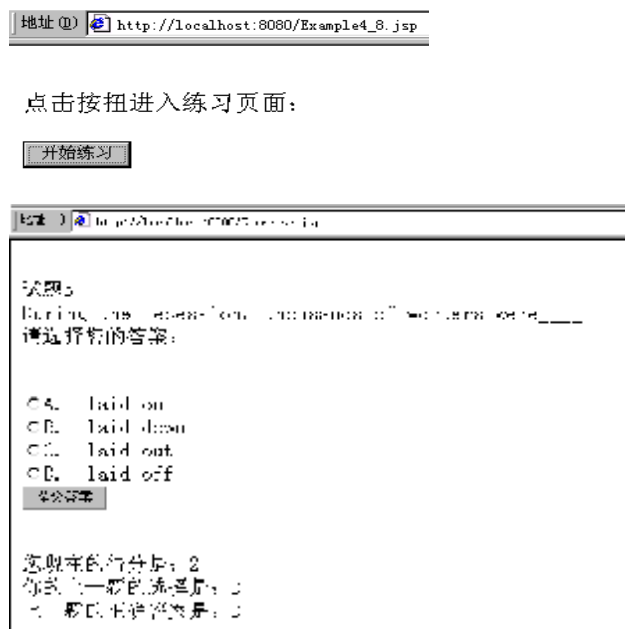


图 4.9 英语单词测试练习

例子 8（效果如图 4.9 所示）

Example4_8.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<%@ page import ="java.util.*" %>

<% int i=0;

    String str=null;

    Integer score=new Integer(0);

    Integer number=new Integer(0);

    session.setAttribute("score",score);

    session.setAttribute("    序号",number);

    try{ //englishi.txt    存放在 f:/2000 目录下。

        File f=new File("f:/2000","English.txt");

        FileReader in=new FileReader(f);

        BufferedReader buffer=new BufferedReader(in);

        while((str=buffer.readLine())!=null)

            {i++;

                session.setAttribute(""+i,str);

            }

    }

    catch(IOException e)

    {
```

```

    }

    %>

<HTML>

<BODY>

<P><BR> 点击按钮进入练习页面:

<FORM action="Exercise.jsp" method="post" name=form>

    <Input type=submit value=" 开始练习">

</FORM>

</BODY>

</HTML>

```

Exercise.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<%@ page import ="java.util.*" %>

<HTML>

<BODY>

<% String option[]=new String[7];

    int    题号=0;

    if(!(session.isNew()))

    { Integer number=(Integer)session.getAttribute("    序号");//获取题号。

        if(number==null)

            {number=new Integer(0);

```

```

    }

    number=new Integer(number.intValue()+1);//      将题号加 1。
session.setAttribute("      序号",number);      //      更新序号

    int i=0;

String str=(String)session.getAttribute(""+number);//      获取行号是 number 的
文本。

    if(str==null)

        {str="#"      练习结束#练习结束#练习结束#练习结束#练习结束#再见#";

        }

StringTokenizer tokenizer=new StringTokenizer(str,"#");//      分析该行文本。

while(tokenizer.hasMoreTokens())

    {option[i]=tokenizer.nextToken();i++;

    }

    题号=number.intValue();

session.setAttribute("      答案"+题号,option[5]); // 将该题答案存入 session。

out.print("<BR>"+ "      试题"+number+"<BR>"+option[0]);

out.print("<BR>      请选择您的答案: ");

out.print("<FORM action=Exercise.jsp method=post name=form>");


out.print("<BR>"+ "<Input type=radio name=R value=A>");

out.print("A.      "+option[1]);

out.print("<BR>"+ "<Input type=radio name=R value=B>");

out.print("B.      "+option[2]);

```

```

        out.print("<BR>"+ "<Input type=radio name=R value=C>");

        out.print("C.          "+option[3]);

        out.print("<BR>"+ "<Input type= radio name=R value=D>");

        out.print("D.          "+option[4]);

        out.print("<BR>"+ "<Input type=submit name=submit value=          提交答案
>");

        out.print("</FORM>");

    }

%>

<% String answer=request.getParameter("R");// 获取客户提交的答案。

    //    获取题目的标准答案，需要注意的是：客户提交答案后，该页面就将题号增
加 1

    //    因此，要给客户的上一题进行评判必须将题号减 1。

String    答案=(String)session.getAttribute("答案"+(题号-1));

if(answer==null)

    {answer="          您没有给出选择呢";

    }

if(answer.equals(    答案))

    { Integer score=(Integer)session.getAttribute("score");

      score=new Integer(score.intValue()+1);

      session.setAttribute("score",score);

    }

    out.print("<BR>"+ "    您现在的得分是: "+session.getAttribute("score"));

```



```

        out.print("<BR>"+ "    你的上一题的选择是: "+answer);

        out.print("<BR>"+ "    上一题的正确答案是: "+答案);

    %>

</BODY>

</HTML>

```

4.4 回压字符流

称 **PushbackReader** 类创建的对象为回压字符流。回压流可以使用 **unread(char ch)** 将一个字符回压到该流中，被回压的字符是该回压流紧接着再调用 **read()** 方法时最先读出的字符。回压流可以用来监视读出的信息，当读出一个不需要的信息时，可以不处理该信息，而将需要的信息回压，然后再读出回压的信息。该类的构造方法是：

```
PushbackReader(Reader in);
```

当我们使用前面讲的字节输入流或字符输入流把 **JSP** 文件或超文本文件发送给客户时，客户的浏览器将解释运行超文本标记，客户无法看见原始的超文本文件和 **JSP** 文件。我们可以使用回压流技术，读取原始的网页文件，当读取到 “<” 符号时，将 “<” 回压、读取到 “>” 时，将 “>” 回压。

下面的例子 9 将 **JSP** 源文件显示给客户。

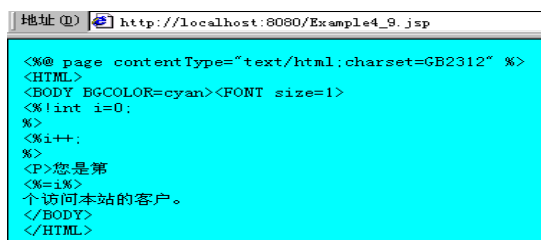


图 4.10 使用回压流显示 JSP 源文件

例子 9 (效果如图 4.10 所示)

Example4_9.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import ="java.io.*" %>
```

```
<HTML>
```

```
<BODY bgcolor=cyan><FONT size=1>
```

```
<%File f=new
```

```
File("d:/Tomcat/Jakarta-tomcat-4.0/webapps/root","Example2_4.jsp");
```

```
try{ FileReader in=new FileReader(f) ;
```

```
    PushbackReader push=new PushbackReader(in);
```

```
    int c;
```

```
    char b[]=new char[1];
```

```
    while ( (c=push.read(b,0,1))!=-1)//      读取 1 个字符放入字符数组 b。
```

```
    { String s=new String(b);
```

```
        if(s.equals("<"))      //      回压的条件
```

```
        { push.unread('&');
```

```
            push.read(b,0,1); //push      读出被回压的字符字节,放入数组 b.
```

```
            out.print(new String(b));
```

```
            push.unread('L');
```

push.read(b,0,1); //push	读出被回压的字符字节,放入数组 b.
out.print(new String(b));	
push.unread('T');	
push.read(b,0,1); //push	读出被回压的字符字节,放入数组 b.
out.print(new String(b));	
}	
else if(s.equals(">")) //	回压的条件
{ push.unread('&');	
push.read(b,0,1); //push	读出被回压的字符字节,放入数组 b.
out.print(new String(b));	
push.unread('G');	
push.read(b,0,1); //push	读出被回压的字符字节,放入数组 b.
out.print(new String(b));	
push.unread('T');	
push.read(b,0,1); //push	读出被回压的字符字节,放入数组 b.
out.print(new String(b));	
}	
else if(s.equals("\n"))	
{ out.print("
");	
}	
else	
{out.print(new String(b));	
}	

```

    }

    push.close();

}

catch(IOException e){}

%>

</BODY>

</HTML>

```

4.5 数据流

`DataInputStream` 类和 `DataOutputStream` 类创建的对象被称为数据输入流和数据输出流。这两个流是很有用的两个流，它们允许程序按着机器无关的风格读取 **Java** 原始数据。也就是说，当我们读取一个数值时，不必再关心这个数值应当是多少个字节。

`DataInputStream` 类和 `DataOutputStream` 的构造方法：

`DataInputStream (InputStream in)`：将创建的数据输入流指向一个由参数 `in` 指定的输入流，以便从后者读取数据（按着机器无关的风格读取）。

`DataOutputStream (OutputStream out)`：将创建的数据输出流指向一个由参数 `out` 指定的输出流，然后通过这个数据输出流把 **Java** 数据类型的数据写到输出流 `out`。

表 4.1 `DataInputStream` 类及 `DataOutputStream` 的部分方法

方法	描述
<code>close()</code>	关闭流
<code>readBoolean()</code>	读取一个布尔值
<code>readByte()</code>	读取一个字节

<code>readChar()</code>	读取一个字符
<code>readDouble()</code>	读取一个双精度浮点值
<code>readFloat()</code>	读取一个单精度浮点值
<code>readInt()</code>	从文件中读取一个 <code>int</code> 值
<code>readLong()</code>	读取一个长型值
<code>readShort()</code>	读取一个短型值
<code>ReadUnsignedByte()</code>	读取一个无符号字节
<code>ReadUnsignedShort()</code>	读取一个无符号短型值
<code>readUTF()</code>	读取一个 UTF 字符串
<code>skipBytes(int n)</code>	跳过给定数量的字节
<code>writeBoolean(boolean v)</code>	把一个布尔值作为单字节值写入
<code>writeBytes(String s)</code>	写入一个字符串
<code>writeChars(String s)</code>	写入字符串
<code>writeDouble(double v)</code>	写入一个双精度浮点值
<code>writeFloat(float v)</code>	写入一个单精度浮点值

v)	
writeInt(int v)	一个 int 值
writeLong(long v)	一个长型值
writeShort(int v)	一个短型值
writeUTF(String s)	写入一个 UTF 字符串

下面的例子 10 使用数据流实现录入成绩单和显示成绩单。在Example4_10.jsp 录入成绩单，连接到 showresult.jsp 页面查询成绩单。

地址: [http://localhost:8080/example4_10.jsp](#)

在下方的表格输入成绩:

姓名	数学	英语
刘永强	80	88
苗小林	90	92
陈芳芳	75	85
周俊杰	84	89
张永光	77	87
陈永成	82	86
陈名平	83	87
录入成绩	Math	English

查看成绩单:
[连接到成绩单页面](#)

图 4.11 成绩录入


地址  http://localhost:8080/showresult.jsp		
成绩单:		
刘小名	90	98
画中林	98	80
细先好	75	89
向前进	86	96
张林关	90	95
微小艳	80	88
荷名金	66	67

图 4.12 成绩单显示

例子 10（效果如图 4.11、4.12 所示）

Example4_10.jsp:

```

<%@ page contentType="text/html; charset=GB2312" %>

<%@ page import="java.io.*" %>

<HTML>

<BODY>

<P> 在下面的表格输入成绩:

<FORM action="Example4_10.jsp" method=post name=form>

<Table align="CENTER" Border>

  <TR>

    <TH width=50>          姓名</TH>

    <TH width=50>          数学</TH>

    <TH width=50>          英语</TH>

```

```

</TR>

<% int i=0;

while(i<=6)

{out.print("<TR>");

    out.print("<TD>");

        out.print("<INPUT type=text name=name value="      姓名>");

    out.print("</TD>");

    out.print("<TD>");

        out.print("<INPUT type=text name=math value=0>");

    out.print("</TD>");

    out.print("<TD>");

        out.print("<INPUT type=text name=english value=0>");

    out.print("</TD>");

    out.print("</TR>");

    i++;

}

%>

<TR>

<TD>

    <INPUT type=submit name="g" value="      写入成绩" >

</TD>

<TD> Math</TD>

<TD> English</TD>

```



```

</TR>

</Table>

</FORM>

<%

    String name[]=request.getParameterValues("name");

    String math[]=request.getParameterValues("math");

    String english[]=request.getParameterValues("english");

    try{ File f=new File("f:/2000","student.txt");

        FileOutputStream o=new FileOutputStream(f);

        DataOutputStream DataOut=new DataOutputStream(o);

        for(int k=0;k<name.length;k++)

            {DataOut.writeUTF(name[k]);

                DataOut.writeUTF(math[k]);

                DataOut.writeUTF(english[k]);

            }

        DataOut.close();

        o.close();

    }

    catch(IOException e)

        {    }

    catch(NullPointerException ee)

        {    }

%>

```

**<P>
 查看成绩单:**

**
 连接到成绩单页面>**

</BODY>

</HTML>

showresult.jsp:

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<HTML>

<BODY>

<P> 成绩单:

<% try{ File f=new File("f:/2000","student.txt");

FileInputStream in=new FileInputStream(f);

DataInputStream DataIn=new DataInputStream(in);

String name="ok";

String math="0",english="0";

out.print("<Table Border>");

out.print("<TR>");

out.print("<TH width=50> 姓名</TH>");

out.print("<TH width=50> 数学</TH>");

out.print("<TH width=50> 英语</TH>");

out.print("</TR>");

while((name=DataIn.readUTF())!=null)

```

    { byte bb[]=name.getBytes("ISO-8859-1");

      name=new String(bb);

      math=DataIn.readUTF();

      english=DataIn.readUTF();

      out.print("<TR>");

        out.print("<TD width=200>");

        out.print(name);

        out.print("</TD>");

        out.print("<TD width=100>");

        out.print(math);

        out.print("</TD>");

        out.print("<TD width=100>");

        out.print(english);

        out.print("</TD>");

      out.print("</TR>");

    }

    out.print("</Table>");

    DataIn.close(); in.close();

  }

  catch(IOException ee)

  {   }

  %>

</BODY>

```

</HTML>

4.6 对象流

`ObjectInputStream` 类和 `ObjectOutputStream` 类分别是 `DataInputStream` 类和 `DataOutputStream` 类的子类。`ObjectInputStream` 类和 `ObjectOutputStream` 类创建的对象被称为对象输入流和对象输出流。对象输出流使用 `writeObject(Object obj)` 方法将一个对象 `obj` 写入到一个文件，对象输入流使用 `readObject()` 读取一个对象到程序中。

`ObjectInputStream` 类和 `ObjectOutputStream` 类的构造方法分别是：

`ObjectInputStream(InputStream in);`

`ObjectOutputStream(OutputStream out);`

`ObjectOutputStream` 的指向应当是一个输出流对象，因此当准备将一个对象写入到文件时，首先用 `FileOutputStream` 创建一个文件输出流，如下列代码所示：

`FileOutputStream file_out=new FileOutputStream("tom.txt");`

`ObjectOutputStream object_out=new ObjectOutputStream(file_out);`

同样 **`ObjectInputStream` 的指向应当是一个输入流对象，因此当准备从文件中读入一个对象到程序中时，首先用 `FileInputStream` 创建一个文件输入流，如下列代码所示：**

`FileInputStream file_in=new FileInputStream("tom.txt");`

`ObjectInputStream object_in=new ObjectInputStream(file_in);`

在下面的例子 11 中我们使用对象流技术实现网上货单录入与查询，客户在 `Example4_11.jsp` 页面输入数据提交给 `input.jsp` 页面，`input.jsp` 页面首先读取文件 `goods_name.txt` 中的散列表对象，如果文件不存在，该客户就是第一个录入货物的客户，就将录入的数据存入散列表，并将散列表写入新创建的文件 `goods_name.txt`。如果该文件已经存在，客户就从文件 `goods_name.txt` 读出散列表，查找客户要存放的货物的货号是否已经存在，如果货号已经存在，客户新录入的数据就替换了旧的库存，然后将散列表写入到文件；如果货号不存在，客户就可将新的货号的库存信息存入到散列表，并将散列表写入到文件。

例子 11 中使用了散列表这种数据结构。使用 `java.util` 包中的 `Hashtable` 类来创建散列表对象，该类的常用方法如下：

`public Hashtable():` 创建具有默认容量和装载因子为 0.75 的散列表。

`public Hashtable(int itialCapacity):` 创建具有指定容量和装载因子为 0.75 的散列表。

`public Hashtable(int initialCapacity,float loadFactor):` 创建具有默认容量和指定装载因子散列表。

`public void clear():` 清空散列表。

`public boolean contains(Object o):` 判断散列表是否含有元素 `o`。

`public Object get(Object key):` 获取散列表中具有关键字 `key` 的数据项。

`public boolean isEmpty():` 判断散列表是否为空。

`public Object put(Object key,Object value):` 向散列表添加数据项 `value` 并把关键字 `key` 关联到数据项 `value`。

`public Object remove(Object key):` 删除关键字是 `key` 的数据项。

`public int size():` 获取散列表中关键字的数目。

使用上述的 `get` 方法可以从散列表中检索某个数据。我们还可以借助 `Enumeration` 对

象实现遍历散列表，一个散列表可以使用 `elements()`方法获取一个 `Enumeration` 对象，后者使用 `nextElement()`方法遍历散列表。

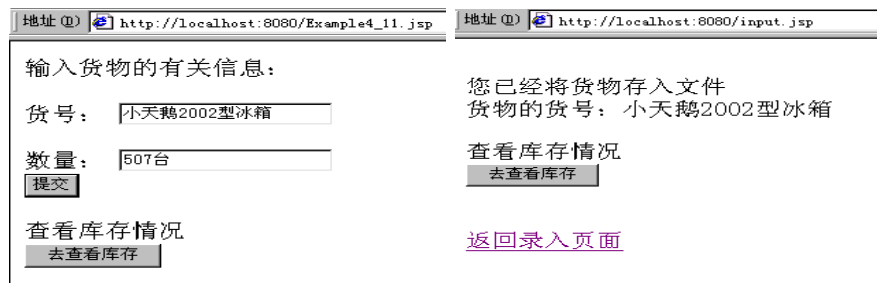


图 4.13 数据输入操作

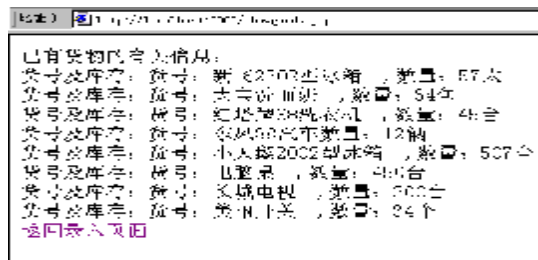


图 4.14 数据显示

例子 11（效果如图 4.13、4.14 所示）

Example4_11.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<%@ page import ="java.util.*" %>

<HTML>

<BODY>

<P> 输入货物有关信息:

<FORM action="input.jsp" method=post >

    <P> 货号:

        <INPUT type=text name="N">

    <P> 数量:

        <INPUT type=text name="M">

    <BR>

    <INPUT type=submit value="    提交">

</FORM>

<FORM action="showgoods.jsp" method=post >

    <P> 查看库存情况

    <BR>

    <INPUT type=submit value="    去查看库存">

</FORM>

</BODY>

</HTML>
```

input.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<%@ page import ="java.util.*" %>

<HTML>

<BODY>

<%! // 声明创建一个散列表对象，被所有的客户共享：

    Hashtable hashtable=new Hashtable();

    //    一个向散列表填加信息的同步方法：

    synchronized void putGoodsToHashtable(String key,String list)

    {hashtable.put(key,list);

    }

    //    从散列表移去信息的同步方法：

    synchronized void removeGoodsToHashtable(String key)

    {hashtable.remove(key);

    }

%>

<%--获取客户提交的书号名字和数量--%>

<%    String name=request.getParameter("N");

    if(name==null)

        {name="have no any goods number";

        }

%>
```



```

byte c[]=name.getBytes("ISO-8859-1");

name=new String(c);

String mount=request.getParameter("M");

if(mount==null)

    {mount="have no any recoder";

    }

byte d[]=mount.getBytes("ISO-8859-1");

mount=new String(d);

```

```
%>
```

<%--从文件中读散列表，如果文件不存在，你就是第一个录入的人，负责写散列表到文件--%>

```

<%File f=new File("F:/2000","goods_name.txt");

if(f.exists())

    { try{FileInputStream in=new FileInputStream(f);

        ObjectInputStream object_in=new ObjectInputStream(in);

        hashtable=(Hashtable)object_in.readObject();

        object_in.close();

        in.close();

        String temp=(String)hashtable.get(name);

        if(temp==null)

            {temp="";

            }

        if((temp.length())!=0)

```

```

{ out.print("<BR>"+ "          该货号已经存在，您已经修改了数量");

String s="          货号: "+name+" ,数量: "+mount;

removeGoodsToHashtable(name); //          首先移去旧的信息。

putGoodsToHashtable(name,s); //          填加新的信息。

//          将新的散列表写入到文件:

try{FileOutputStream o=new FileOutputStream(f);

    ObjectOutputStream object_out=new

ObjectOutputStream(o);

    object_out.writeObject(hashtable);

    object_out.close();

    o.close();

}

catch(Exception eee)

{ }

}

else

{String s="          货号: "+name+" ,数量: "+mount;

putGoodsToHashtable(name,s); //          向散列表填加新的货物信息。

//          再将新的散列表写入到文件:

try{FileOutputStream o=new FileOutputStream(f);

    ObjectOutputStream object_out=new

ObjectOutputStream(o);

    object_out.writeObject(hashtable);

```

```

        object_out.close();

        o.close();

    }

    catch(Exception eee)

    { }

    out.print("<BR>"+ "          您已经将货物存入文件");

    out.print("<BR>"+ "          货物的货号: "+name);

}

}

catch(IOException e) {}

}

else

{ // 如果文件不存在，您就是第 1 个录入信息的人。

    String s="          货号: "+name+"数量: "+mount;

    putGoodsToHashtable(name,s);//          放信息到散列表。

    //          负责创建文件，并将散列表写入到文件：

    try{ FileOutputStream o=new FileOutputStream(f);

        ObjectOutputStream object_out=new

ObjectOutputStream(o);

        object_out.writeObject(hashtable);

        object_out.close();

        o.close();

        out.print("<BR>"+ "          您是第一个录入货物的人");

```

```

        out.print("<BR>"+ "          货物的货号: "+name);

    }

    catch(Exception eee)

    { }

}

%>

<FORM action="showgoods.jsp" method=post >

    <P> 查看库存情况

        <BR> <INPUT type=submit value="    去查看库存">

    </FORM>

<A href="Example4_11.jsp"><BR>返回录入页面

</BODY>

</HTML>

```

showgoods.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<%@ page import ="java.util.*" %>

<HTML>

<BODY>

    <P> 已有货物的有关信息:

    <%

        try{File f=new File("F:/2000","goods_name.txt");

```

```

FileInputStream in=new FileInputStream(f);

ObjectInputStream object_in=new ObjectInputStream(in);

Hashtable hashtable=(Hashtable)object_in.readObject();

object_in.close();

in.close();

Enumeration enum=hashtable.elements();

    while(enum.hasMoreElements()) //          遍历当前散列表。

    { String goods=(String)enum.nextElement();

      out.print("<BR>"+"          货号及库存: "+goods);

    }

hashtable.clear();

}

catch(ClassNotFoundException event)

    { out.println("          无法读出");

    }

catch(IOException event)

    {out.println("          无法读出");

    }

%>

<A href="Example4_11.jsp"><BR>返回录入页面

</BODY>

</HTML>

```

4.7 RandomAccessFile 类

前面我们学习了用来处理文件的几个文件输入流、文件输出流，而且通过一些例子，已经了解了那些流的功能。

`RandomAccessFile` 类创建的流与前面的输入、输出流不同，`RandomAccessFile` 类既不是输入流类 `InputStream` 类的子类，也不是输出流类 `OutputStream` 类的子类流。习惯上，仍然称 `RandomAccessFile` 类创建的对象为一个流，`RandomAccessFile` 流的指向既可以作为源也可以作为目的地，换句话说，当我们想对一个文件进行读写操作时，我们可以创建一个指向该文件的 `RandomAccessFile` 流即可，这样我们既可以从这个流中读取这个文件的数据，也通过这个流写入数据给这个文件。

`RandomAccessFile` 类的两个构造方法：

`RandomAccessFile(String name,String mode)`：参数 `name` 用来确定一个文件名，给出创建的流的源（也是流目的地），参数 `mode` 取“r”（只读）或“rw”（可读写），决定创建的流对文件的访问权利。

`RandomAccessFile(File file,String mode)`：参数 `file` 是一个 `File` 对象，给出创建的流的源（也是流目的地），参数 `mode` 取“r”（只读）或“rw”（可读写），决定创建的流对文件的访问权利。创建对象时应捕获 `FileNotFoundException` 异常，当流进行读写操作时，应捕获 `IOException` 异常。

`RandomAccessFile` 类中有一个方法 `seek(long a)`，用来移动 `RandomAccessFile` 流指向的文件的指针，其中参数 `a` 确定文件指针距离文件开头的字节位置。另外流还可以调用 `getFilePointer()` 方法获取当前文件的指针的位置（`RandomAccessFile` 类的一些方法见下表），`RandomAccessFile` 流对文件的读写比顺序读写的文件输入输出流更为灵活。

表 4.2 `RandomAccessFile` 类的常用方法

方法	描述
<code>close()</code>	关闭文件
<code>getFilePointer()</code>	获取文件指针的位置
<code>length()</code>	获取文件的长度
<code>read()</code>	从文件中读取一个字节的数据
<code>readBoolean()</code>	从文件中读取一个布尔值，0 代表 <code>false</code> ；其他值代表 <code>true</code>
<code>readByte()</code>	从文件中读取一个字节
<code>readChar()</code>	从文件中读取一个字符 (2 个字节)
<code>readDouble()</code>	从文件中读取一个双精度浮点值 (8 个字节)
<code>readFloat()</code>	从文件中读取一个单精度浮点值 (4 个字节)
<code>readFully(byte b[])</code>	读 <code>b.length</code> 字节放入数组 <code>b</code> ，完全填满该数组
<code>readInt()</code>	从文件中读取一个 <code>int</code> 值 (4 个字节)
<code>readLine()</code>	从文件中读取一个文本行
<code>readLong()</code>	从文件中读取一个长型值 (8 个字节)
<code>readShort()</code>	从文件中读取一个短型值 (2 个字节)

	节)
<code>readUTF()</code>	从文件中读取一个 UTF 字符串
<code>seek()</code>	定位文件指针在文件中的位置
<code>setLength(long newlength)</code>	设置文件的长度
<code>skipBytes(int n)</code>	在文件中跳过给定数量的字节
<code>write(byte b[])</code>	写 <code>b.length</code> 个字节到文件
<code>writeBoolean(boolean v)</code>	把一个布尔值作为单字节值写入文件
<code>writeByte(int v)</code>	向文件写入一个字节
<code>writeBytes(String s)</code>	向文件写入一个字符串
<code>writeChar(char c)</code>	向文件写入一个字符
<code>writeChars(String s)</code>	向文件写入一个作为字符数据的字符串
<code>writeDouble(double v)</code>	向文件写入一个双精度浮点值
<code>writeFloat(float v)</code>	向文件写入一个单精度浮点值

<code>writeInt(int v)</code>	向文件写入一个 int 值
<code>writeLong(lon g v)</code>	向文件写入一个长型 int 值
<code>writeShort(in t v)</code>	向文件写入一个短型 int 值
<code>writeUTF(Stri ng s)</code>	写入一个 UTF 字符串

下面的例子实现网上小说创作，每个客户都可以参与一部小说的写作，也就是说一个客户需接着前一个客户的写作内容继续写作。在服务器的某个目录下有 4 部小说，小说的内容完全由客户来决定。客户首先在 **Example4_12.jsp** 页面选择一部小说的名字，然后连接到 **continueWrite.jsp** 页面，该页面显示小说的已有内容，客户可以在该页面输入续写的内容，再提交给 **continue.jsp** 页面，该页面负责将续写的内容存入文件，并通知客户续写是否成功，如果其他用户正在保存续写的内容到该小说，就通知该客户等待。

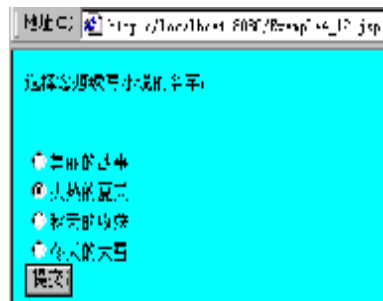


图 4.15 选择续写的小说

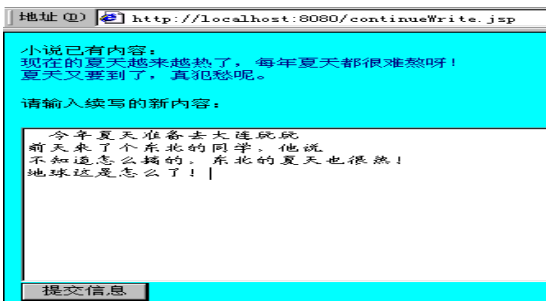


图 4.16 编辑、提交续写的内容



内容已经写入

图 4.17 续写内容成功提示

例子 12（效果如图 4.15、4.16、4.17 所示）

Example4_12.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<HTML>

<BODY bgcolor=cyan><Font size=1>

<% String str=response.encodeURL("continueWrite.jsp");

%>

<P> 选择您想续写小说的名字: <BR>

<FORM action="<%=str%>" method=post name=form>

<BR><INPUT type="radio" name="R" value="spring.doc" > 美丽的故事
```

```

<BR><INPUT type="radio" name="R" value="summer.doc" > 火热的夏天
<BR><INPUT type="radio" name="R" value="autumn.doc" > 秋天的收获
<BR><INPUT type="radio" name="R" value="winter.doc" > 冬天的大雪
<BR> <INPUT type=submit name ="g" value=" 提交">

</FORM>

</BODY>

</HTML>

```

continueWrite.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<HTML>

<BODY bgcolor=cyan><Font size=1>

<P>小说已有内容:

<Font size=1 Color=Navy>

<%String str=response.encodeURL("continue.jsp");

%>

<%--获取客户提交的小说的名字--%>

<%String name=(String)request.getParameter("R");

    if(name==null)

        {name="";

        }

    byte c[]=name.getBytes("ISO-8859-1");

```

```

        name=new String(c);

session.setAttribute("name",name);

File storyFileDir=new File("F:/2000","story");

storyFileDir.mkdir();

File f=new File(storyFileDir,name);

// 列出小说的内容:

try{ RandomAccessFile file=

        new RandomAccessFile(f,"r");

        String temp=null;

        while((temp=file.readUTF())!=null)

            { byte d[]=temp.getBytes("ISO-8859-1");

                temp=new String(d);

                out.print("<BR>" +temp);

            }

        file.close();

    }

    catch(IOException e){}

%>

</Font>

<P> 请输入续写的新内容:

<Form action="<%=str%>" method=post name=form>

    <TEXTAREA name="messages" ROWs="12" COLS=80 WRAP="physical">

    </TEXTAREA>

```

**
**

<INPUT type="submit" value=" 提交信息" name="submit">

</FORM>

continue.jsp:

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<%@ page import ="java.util.*" %>

<HTML>

<BODY>

<%! // 声明一个需同步处理的文件:

File f=null;

String use="yes";

%>

<%--获取客户提交的小说的名字--%>

<% String name=(String)session.getAttribute("name");

byte c[]=name.getBytes("ISO-8859-1");

name=new String(c);

// 获取客户续写的内容:

String content=(String)request.getParameter("messages");

if(content==null)

{content=" ";

}

%>

```
<%File storyFileDir=new File("F:/2000","story");

storyFileDir.mkdir();

f=new File(storyFileDir,name);

// 把对文件的操作放入一个同步块中，并通知
// 其它用户该文件正在被操作中：

if(use.startsWith("yes"))

{ synchronized(f)

{ use="beisusing";

try{

    RandomAccessFile file=new RandomAccessFile(f,"rw");

    file.seek(file.length()); //          定位到文件的末尾。

    file.writeUTF(content);

    file.close();

    use="yes";

    out.print("<BR>"+          内容已经写入");

}

catch(IOException e){}

}

}

// 如果该小说正在被续写，就通知客户等待：

else

{out.print("          该小说正在被续写，请等待");
```

```

    }

%>

</BODY>

</HTML>

```

4.8 文件上传

客户通过一个 JSP 页面，上传文件给服务器时，该 JSP 页面必须含有 File 类型的表单，并且表单必须将 ENCTYPE 的属性值设成 “multipart/form-data”，File 类型表单如下所示：

```

<Form action=      “ 接 受 上 传 文 件 的 页      面 ” method=      “post”
ENCTYPE=“ multipart/form-data”

<Input Type= “File” name= “picture” >

</Form>

```

JSP 引擎可以让内置对象 request 调用方法 `getInputStream()` 获得一个输入流，通过这个输入流读入客户上传的全部信息，包括文件的内容以及表单域的信息。

下面的例子 13 中，客户通过 `Example4_13.jsp` 页面上传如下的文本文件 `A.txt`。

`A.txt`:

你好，我们正在学习文件的上传，request 调用 `getInputStream()` 可以获得一个输入流，通过这个输入流可以读取客户上传的全部信息，包括表单的头信息以及上传文件的内容。以后将讨论如何去掉表单的信息，获取文件的内容。

在 `accept.jsp` 页面，内置对象 request 调用方法 `getInputStream()` 获得一个输入流 `in`、用 `FileOutputStream` 创建一个输出流 `o`。输入流 `in` 读取客户上传的信息，输出流 `o` 将读取

的信息写入文件 B.txt，该文件 B.txt 被存放于服务器的 F:/2000 中。B.txt 的内容如图 4.20 所示。

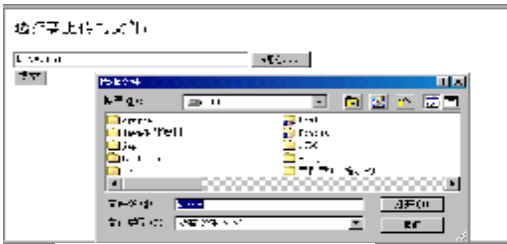


图 4.18 选择上传的文件



图 4.19 接收上传

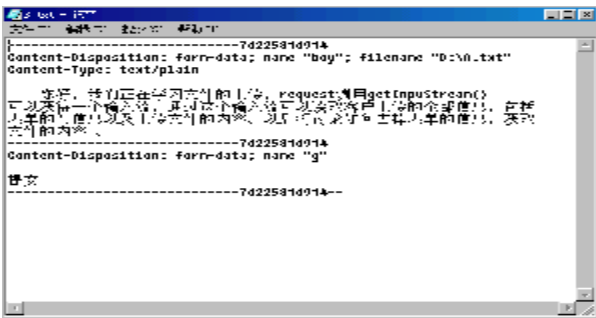


图 4.20 收到的上传信息

文件 B.txt 的前 4 行（包括一个空行）以及倒数 5 行（包括一个空行）是表单域的内容，中间部分是上传文件 A.txt 的内容。

例子 13（效果如图 4.18、4.19、4.20 所示）

Example4_13.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<P> 选择要上传的文件: <BR>

<FORM action="accept.jsp" method="post" ENCTYPE="multipart/form-data">

    <INPUT type=FILE name="boy" size="38">

    <BR>

    <INPUT type="submit" name ="g" value="    提交">

</BODY>

</HTML>
```

accept.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<HTML>

<BODY>

<%try{  InputStream in=request.getInputStream();

        File f=new File("F:/2000","B.txt");

        FileOutputStream o=new FileOutputStream(f);

        byte b[]=new byte[1000];

        int n;
```

```

        while((n=in.read(b))!=-1)

            {o.write(b,0,n);

            }

        o.close();

        in.close();

    }

    catch(IOException ee){}

    out.print("  文件已上传");

    %>

</BODY>

</HTML>

```

通过上面的讨论我们知道，按着 HTTP 协议，文件表单提交的信息中，前 4 行和后面的 5 行是表单本身的信息，中间部分才是客户提交的文件的内容。在下面的例子中我们通过输入输出流技术获取文件的内容，即去掉表单的信息。

首先，我们将客户提交的全部信息保存为一个临时文件，该文件的 名字是客户的 session 对象的 Id，不同客户的这个 Id 是不同的。然后读取该文件的第 2 行，这一行中含有客户上传的文件的名字，获取这个名字。再获取第 4 行结束的位置，以及倒数第 6 行的结束位置，因为这两个位置之间的内容是上传文件的内容。然后将这部分内容存入文件，该文件的 名字和客户上传的文件的名字保持一致。最后删除临时文件。

在下面的例子 14 中，客户上传一个图象文件，还可以连接到 `showImage.jsp` 页面查看这个上传图象的效果。我们可以允许客户将文件上传到服务器的任何一个目录，为了让客户能查看上传图象的效果，下面的例子 14 将上传文件保存到一个 web 服务目录 `D:/tomcat/jakarta-tomcat-4.0/webapps/examples` 中，假设服务器的 IP 是：192.168.0.100。

例子 14（效果如图 4.21、4.22、4.23 所示）

Example4_14.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<HTML>
```

```
<BODY>
```

```
<% String str=response.encodeURL("acceptFile.jsp");
```

```
%>
```

```
<P> 选择要上传的文件: <BR>
```

```
<FORM action="<%=str %>" method="post"
```

```
ENCTYPE="multipart/form-data">
```

```
<INPUT type=FILE name="boy" size="45">
```

```
<BR> <INPUT type="submit" name="g" value="    提交">
```

```
</FORM>
```

```
</BODY>
```

```
</HTML>
```

acceptFile.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import ="java.io.*" %>
```

```
<HTML>
```

```
<BODY>
```

```
<%try{ // 用客户的 session 的 id 建立一个临时文件:
```

```
String tempFileName=(String)session.getId();
```

```

//      建立临时文件 f1:

File f1=new

File("D:/Tomcat/jakarta-tomcat-4.0/webapps/examples/",tempFileName);

FileOutputStream o=new FileOutputStream(f1);

//      将客户上传的全部信息存入 f1:

InputStream in=request.getInputStream();

byte b[]=new byte[10000];

int n;

while( (n=in.read(b))!=-1)

    {o.write(b,0,n);

    }

o.close();in.close();

//      读取临时文件 f1，从中获取上传文件的 名字，和上传的文件的内容:

RandomAccessFile random=new RandomAccessFile(f1,"r");

//      读出 f1 的第 2 行，析取出上传文件的 名字:

int second=1;

String secondLine=null;

while(second<=2)

    {secondLine=random.readLine();

    second++;

    }

//      获取第 2 行中目录符号'\'最后出现的位置

int position=secondLine.lastIndexOf('\\');

```

```

//      客户上传的文件的名字是:

String
fileName=secondLine.substring(position+1,secondLine.length()-1);

random.seek(0); //      再定位到文件 f1 的开头。

//      获取第 4 行回车符号的位置:

long forthEndPosition=0;

int forth=1;

while((n=random.readByte())!=-1&&(forth<=4))

    { if(n=='\n')

        { forthEndPosition=random.getFilePointer();

            forth++;

        }

    }

//      根据客户上传文件的名字，将该文件存入磁盘:

File f2=new

File("D:/Tomcat/jakarta-tomcat-4.0/webapps/examples/",fileName);

session.setAttribute("Name",fileName);//      供 showImage.jsp 页面使用。

RandomAccessFile random2=new RandomAccessFile(f2,"rw");

//      确定出文件 f1 中包含客户上传的文件的内容的最后位置，即倒数第
6 行。

random.seek(random.length());

long endPosition=random.getFilePointer();

long mark=endPosition;

```

```

int j=1;

while((mark>=0)&&(j<=6))
{
    mark--;

    random.seek(mark);

    n=random.readByte();

    if(n=='\n')
    {
        endPosition=random.getFilePointer();

        j++;
    }
}

//      将 random 流指向文件 f1 的第 4 行结束的位置：

random.seek(forthEndPosition);

long startPoint=random.getFilePointer();

//      从 f1 读出客户上传的文件存入 f2（读取从第 4 行结束位置和倒数第 6 行
之间的内容）。

while(startPoint<endPosition-1)
{
    n=random.readByte();

    random2.write(n);

    startPoint=random.getFilePointer();
}

random2.close();random.close();

f1.delete(); //      删除临时文件

}

```

```
catch(IOException ee){}
```

```
out.print(" 文件已上传");
```

```
%>
```

<P> 查看上传的图象效果

```
<%String str=response.encodeURL("showImage.jsp");
```

```
%>
```

```
<FORM action="<%=str%>">
```

```
<Input type=submit value=" 查
```

```
</Form >
```

```
</BODY>
```

```
</HTML>
```

showImage.jsp:

```
<HTML>
```

```
<BODY>
```

```
<% String name=(String)session.getAttribute("Name");
```

```
if(name==null)
```

```
{name="";
```

```
}
```

```
out.print("<image src= http://192.168.1.100:8080/examples/"+name);
```

```
%>
```

```
</BODY>
```

```
</HTML>
```

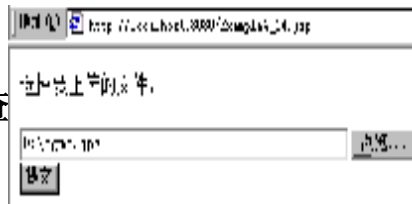


图 4.21 选择上传的图象文件

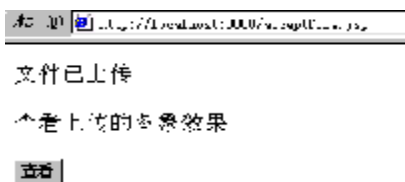


图 4.22 接收上传的图象文件



图 4.23 显示上传的图象文件

4.9 文件下载

JSP 内置对象 **response** 调用方法 **getOutputStream()** 可以获取一个指向客户的输出流，服务器将文件写入这个流，客户就可以下载这个文件了。

当 JSP 页面提供下载功能时，应当使用 **response** 对象向客户发送 HTTP 头信息，说明文件的 **MIME** 类型，这样客户的浏览器就会调用相应的外部程序打开下载的文件。例如，Ms-Word 文件的 **MIME** 类型是 **application/msword**，pdf 文件的 **MIME** 类型是 **application/pdf**。点击资源管理器→工具→文件夹选项→文件类型可以查看文件的相应的 **MIME** 类型。

在下面的例子 15 中，客户在 **Example4_15** 页面点击超链接下载一个 **zip** 文档。



图 4.24 选择下载文件



图 4.25 下载文件

例子 15（效果如图 4.24、4.25 所示）

Example4_15.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<P>点击超链接下载 Zip 文档 book.Zip

<BR> <A href="loadFile.jsp"> 下载 book.zip

</Body>

</HTML>
```

loadFile.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.io.*" %>

<HTML>

<BODY>

<% //获得响应客户的输出流:

    OutputStream o=response.getOutputStream();

    // 输出文件用的字节数组,每次发送 500 个字节到输出流:
```

```

byte b[]=new byte[500];

// 下载的文件：

File fileLoad=new File("f:/2000","book.zip");

// 客户使用保存文件的对话框：

response.setHeader("Content-disposition","attachment;filename="+ "book.zip");

// 通知客户文件的 MIME 类型：

response.setContentType("application/x-tar");

// 通知客户文件的长度：

long fileLength=fileLoad.length();

String length=String.valueOf(fileLength);

response.setHeader("Content_Length",length);

// 读取文件 book.zip,并发送给客户下载：

FileInputStream in=new FileInputStream(fileLoad);

int n=0;

while((n=in.read(b))!=-1)

    { o.write(b,0,n);

    }

%>

</BODY>

</HTML>

```

注：如果在 fileLoad.jsp 中取消下列语句：

```
response.setHeader("Content-disposition","attachment;filename="+ "book.zip");
```


例子 16（效果如图 4.26 所示）

readFileByLine.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<%! //对字符串进行回压流处理的方法:

public String getString(String content)

{ try{ StringReader in=new StringReader(content) ;// 指向字符串的字符流。

    PushbackReader push=new PushbackReader(in); // 回压流

    StringBuffer stringbuffer=new StringBuffer(); // 缓冲字符串对象。

    int c;

    char b[]=new char[1];

    while ( (c=push.read(b,0,1))!=-1)// 读取 1 个字符放入字符数组 b。

    { String s=new String(b);

        if(s.equals("<")) // 回压的条件

        { push.unread('&');

            push.read(b,0,1); //push 读出被回压的字符字节,放入数组 b.
```

```

    stringBuffer.append(new String(b));

    push.unread('L');

    push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.

    stringBuffer.append(new String(b));

    push.unread('T');

    push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.

    stringBuffer.append(new String(b));

}

else if(s.equals(">"))    //          回压的条件

{
    push.unread('&');

    push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.

    stringBuffer.append(new String(b));

    push.unread('G');

    push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.

    stringBuffer.append(new String(b));

    push.unread('T');

    push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.

    stringBuffer.append(new String(b));

}

else if(s.equals("\n"))

{
    stringBuffer.append("<BR>");

}

else

```

```

        { stringBuffer.append(s);
        }
    }

    push.close();

    in.close();

    return new String(stringbuffer); //      返回处理后的字符串。
}

```

```

catch(IOException e)

```

```

    {return content=new String("      不能读取内容");
    }
}

```

```

}

```

```

%>

```

```

<% String s=request.getParameter("g"); //      获取客户提交的信息（是否重新读取文件）

```

```

    if(s==null)

```

```

        {s=""; }

```

```

    byte b[]=s.getBytes("ISO-8859-1");

```

```

    s=new String(b);

```

```

    File f=null;

```

```

    FileReader in=null;

```

```

    BufferedReader buffer=null;

```

```

    if(session.isNew()) //      当第一次请求该页面时，建立和文件的输入流连接。

```

```

    { f=new File("f:/javabook","JSP      教程.txt");

```

```

in=new FileReader(f);

buffer=new BufferedReader(in);

session.setAttribute("file",f); session.setAttribute("in",in);

session.setAttribute("buffer",buffer);

}

```

if(s.equals(" 重新读取文件")) // 当请求重新读取文件时，建立和文件的输入流连接。

```

{ f=new File("f:/javabook","JSP      教程.txt");

in=new FileReader(f);

buffer=new BufferedReader(in);

session.setAttribute("file",f); session.setAttribute("in",in);

session.setAttribute("buffer",buffer);

}

```

// 将上述对象保存到用户的 session 对象中：

```

try{ String str=null; int i=1;

    f=(File)session.getAttribute("file");

    in=(FileReader)session.getAttribute("in");

    buffer=(BufferedReader)session.getAttribute("buffer");

    while( ((str=buffer.readLine())!=null)&&(i<=5))

        { //          为了能显示原始的 HTML 文件或 JSP 文件需使用回压流技术。

            str=getString(str);

            out.print("<BR>" +str);

            i++;

```

```

        }
    }
    catch(IOException e)
    { out.print("        文件不存在，或读取出现问题");
    }
%>

<%String code=response.encodeURL("readFileByLine.jsp");
%>

<HTML>

<BODY>

<P><BR> 点击按钮读取下 5 行：

<FORM action="<%=code%>" method="post" name=form>

    <Input type=submit value="    读取文件的下 5 行">

</FORM>

<P><BR> 点击按钮读重新读取文件：

<FORM action=""" method="post" name=form>

    <Input type=submit name="g" value="    重新读取文件">

</FORM>

</BODY>

</HTML>

```

4.11 标准化考试

大部分网络上的标准化 考试试题都是使用数据库技术 实现的，使用数据库 易编写代码，但降低了效率，因为打开一个数据库连接的时间要远远慢于打开一个文件。这一节，

我们结合 Java 的流技术实现一个标准化网络 考试。

用一个文件输入流每次读取一道试题。

试题文件的书写格式是：

- 1 第一行必须是全部试题的答案（用来判定考试者的分数）。
- 1 每道题目之间用一行：*****分割(至少含有两个**）。

试题可以是一套标准的英语测试题，包括单词测试，阅读理解等（也可以在文件的最后给出整套试题的全部答案）。所用试题 English.txt 如图 4.27 所示.。下述的 ttt.jsp 存放在 Root 目录中，English.txt 存放在 F:/2000 中。

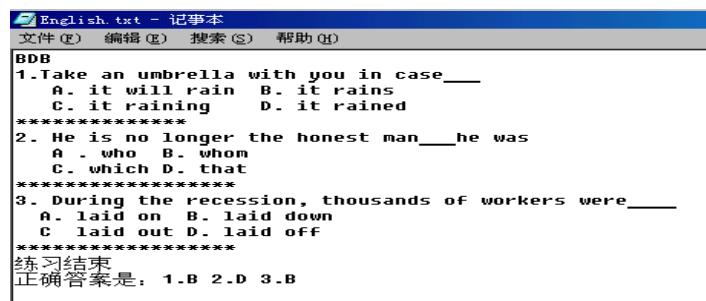


图 4.27 试题文件的书写格式

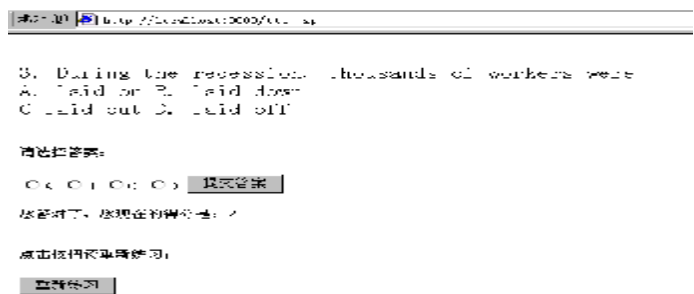


图 4.28 标准化网络考试

标准化考试（效果如图 4.28 所示）

ttt.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<%@ page import ="java.util.*" %>

<%!

    String answer=null;// 存放答案用的字符串。

    // 对字符串进行回压流处理的方法:

    public String getString(String content)

    { try{ StringReader in=new StringReader(content) ;// 指向字符串的字符流。

        PushbackReader push=new PushbackReader(in); // 回压流

        StringBuffer stringbuffer=new StringBuffer(); // 缓冲字符串对象。

        int c;

        char b[]=new char[1];

        while ( (c=push.read(b,0,1))!=-1)// 读取 1 个字符放入字符数组 b。

        { String s=new String(b);

            if(s.equals("<")) // 回压的条件
```

```

{ push.unread('&');

  push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
  stringBuffer.append(new String(b));

  push.unread('L');

  push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
  stringBuffer.append(new String(b));

  push.unread('T');

  push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
  stringBuffer.append(new String(b));

}

else if(s.equals(">"))    //          回压的条件

{ push.unread('&');

  push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
  stringBuffer.append(new String(b));

  push.unread('G');

  push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
  stringBuffer.append(new String(b));

  push.unread('T');

  push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
  stringBuffer.append(new String(b));

}

else if(s.equals("\n"))

{ stringBuffer.append("<BR>");

```

```

    }

    else

        { stringBuffer.append(s);

        }

    }

    push.close();

    in.close();

    return new String(stringbuffer); //      返回处理后的字符串。
}

catch(IOException e)

    {return content=new String("    不能读取内容");

    }

}

%>

<% String s=request.getParameter("g"); //      获取客户提交的信息（是否重新读取文件）

    if(s==null)

        {s="";

        }

    byte b[]=s.getBytes("ISO-8859-1");

    s=new String(b);

    File f=null;

    FileReader in=null;

```

BufferedReader buffer=null;

Integer number=new Integer(0); // 初始题号。

Integer score=new Integer(0);// 初始分数。

if(session.isNew()) // 当第一次请求该页面时，建立和文件的输入流连接。

{ f=new File("f:/2000","English.txt");

in=new FileReader(f);

buffer=new BufferedReader(in);

// 读入文件的第 1 行：答案

answer=buffer.readLine().trim();;

// 将上述 f、in、buffer 对象保存到用户的 session 对象中：

session.setAttribute("file",f);

session.setAttribute("in",in);

session.setAttribute("buffer",buffer);

// 再将初始题号保存到 session 对象中：

session.setAttribute("number",number);

// 再将用户的初始得分保存到 session 对象中：

session.setAttribute("score",score);

}

if(s.equals(" 重新练习")) // 当请求重新读取文件时，建立和文件的输入流连接。

{ f=new File("f:/2000","English.txt");

in=new FileReader(f);

buffer=new BufferedReader(in);

```

//      读入文件的第 1 行：答案

answer=buffer.readLine().trim();

//      将上述 f、in、buffer 对象保存到用户的 session 对象中：

session.setAttribute("file",f);

session.setAttribute("in",in);

session.setAttribute("buffer",buffer);

//      再将初始题号保存到 session 对象中：

session.setAttribute("number",number);

//      再将用户的初始得分保存到 session 对象中：

session.setAttribute("score",score);

}

//      读取试题：

try{ String str=null;

    f=(File)session.getAttribute("file");

    in=(FileReader)session.getAttribute("in");

    buffer=(BufferedReader)session.getAttribute("buffer");

    while((str=buffer.readLine())!=null)

    { //          为了能显示原始的 HTML 文件或 JSP 文件需使用回压流技术。

        str=getString(str);

        if(str.startsWith("***")) //          每个试题的结束标志。

        {break;

        }

        out.print("<BR>" +str);

```

```

        }

    }

    catch(IOException e)

    { out.print(""+e);

    }

%>

<%String code=response.encodeURL("ttt.jsp");

%>

<HTML>

<BODY><Font size=1>

<P><BR> 请选择答案:

<FORM action="<%=code%>" method="post" name=form>

    <Input type=radio name="r" value="A">A

    <Input type=radio name="r" value="B">B

    <Input type=radio name="r" value="C">C

    <Input type=radio name="r" value="D">D

    <Input type=submit name="submit" value=" 提交答案">

</FORM>

<% //当用户提交表单时，获取提交的答案:

// 判断用户是否提交了答案表单:

String select=request.getParameter("submit"); // 获取客户提交的答案选择表单

if(select==null)

    {select="";

```

```

    }

    byte c[]=select.getBytes("ISO-8859-1");

    select=new String(c);

    if(select.equals(" 提交答案"))

    { String userAnswer=request.getParameter("r");

        if(userAnswer==null)

            { userAnswer="#";

                }

        //  将提交的答案与正确答案进行比较:

        //  首先获取题号:

        Integer num=(Integer)session.getAttribute("number");

        int tihao=num.intValue();

        //  获取相应题号的标准答案:

        char correctAnswer='\0';

        try{ correctAnswer=answer.charAt(tihao);

            }

        catch(StringIndexOutOfBoundsException ee)

            { tihao=0;

                }

        //  然后再将题号重新存入 session 对象:

        tihao=tihao+1;

        Integer newNumber=new Integer(tihao);

        session.setAttribute("number",newNumber);

```



```

//    将用户提交的答案与标准答案比较

char user=userAnswer.charAt(0);

if(user==correctAnswer)

    { //        给用户增加分值：

        Integer newScore=(Integer)session.getAttribute("score");

        int fenshu=newScore.intValue();

        fenshu=fenshu+1;

        newScore=new Integer(fenshu);

        session.setAttribute("score",newScore);

        out.print("        您答对了，您现在的得分是： ");

        out.print(""+fenshu);

    }

else

    { out.print("        您没有答对，继续努力！ ");

      out.print("        您现在的得分是： "+session.getAttribute("score"));

    }

}

%>

<P><BR>点击按钮重新练习：

<FORM action="" method="post" name=form>

    <Input type=submit name="g" value="    重新练习">

</FORM>

</FONT>

```

</BODY>

</HTML>

在 JSP 中可以使用 Java 的 JDBC 技术,实现对数据库中表记录的查询、修改和删除等操作。

JDBC 技术在 JSP 开发中占有很重要的地位。

JDBC(Java DataBase Connectivity)是 Java 数据库连接 API。简单来说, JDBC 能完成三件事:

- (1) 与一个数据库建立连接,
- (2) 向数据库发送 SQL 语句,
- (3) 处理数据库返回的结果。

JDBC 在设计上和 **ODBC** 很相似。**JDBC** 和数据库建立连接的一种常见方式是建立一个 **JDBC—ODBC** 桥接器。由于 **ODBC** 驱动程序被广泛的使用，建立这种桥接器后，使得 **JDBC** 有能力访问几乎所有类型的数据库。**JDBC** 也可以直接加载数据库驱动程序访问数据库，我们将在 2.8 节讨论。

如果使用 JDBC—ODBC 桥接器访问数据库，事先必须设置数据源。

5.1 数据源

假设要访问 SQL Server 服务器上的 pubs 数据库，该库有一个表 students，如图 5.1、

5.2 所示。

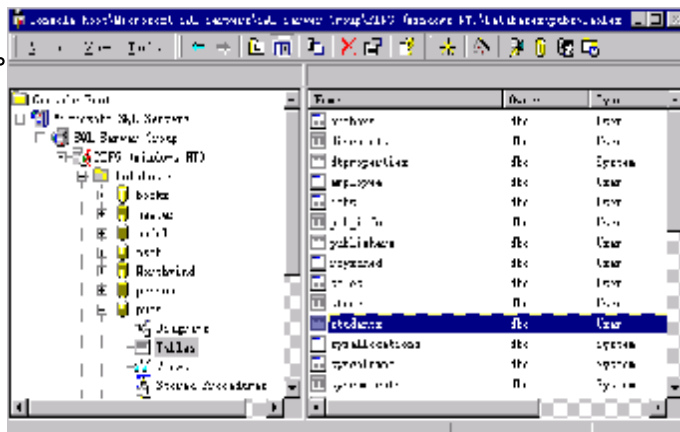
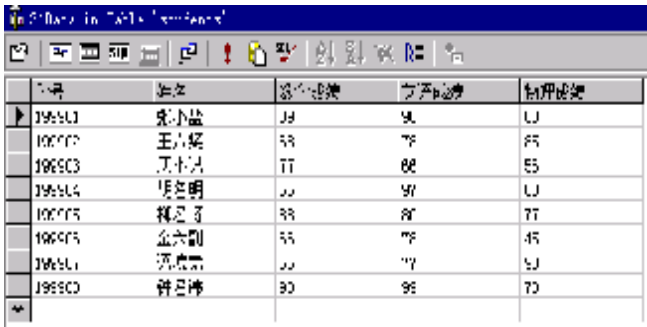


图 5.1 SQL Server 数据库 pubs

为连接一个 SQL-Server 数据库，我们需设置一个数据源。在控制面板选择 ODBC 数据源，如图 5.3 所示。

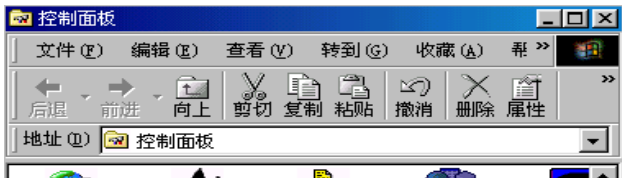
双击 ODBC 数据源图标。出现如图 5.4 所示界面，图 5.4 中显示了用户已有的数据源的名称。



The screenshot shows a web browser window with a table titled 'students'. The table has five columns: '学号' (Student ID), '姓名' (Name), '语文成绩' (Chinese Score), '数学成绩' (Math Score), and '物理成绩' (Physics Score). There are eight rows of data.

学号	姓名	语文成绩	数学成绩	物理成绩
198901	王小强	88	90	85
198902	王小明	85	88	82
198903	王小红	77	80	75
198904	张晓明	80	85	80
198905	杨志军	88	88	85
198906	金大明	85	88	85
198907	陈晓明	80	85	80
198908	孙晓明	80	85	80

图 5.2 表 students



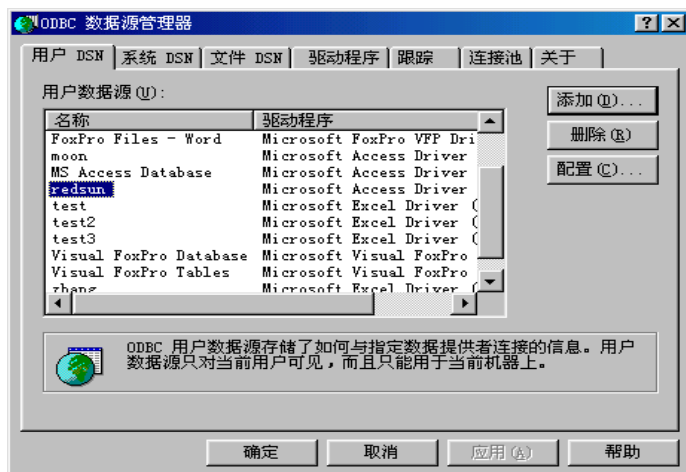


图 5.4 用户已有的数据源

选择“用户 DSN”，点击 **add** 按钮，增加新的数据源。如图 5.5 所示：



图 5.5 为新增的数据源选择驱动程序

为新增的数据源选择驱动程序，因为要访问 **SQL Server** 数据库，选择 **SQL Server**，点击完成按钮（为数据源选择了驱动程序），出现设置数据源具体项目的对话框，如图 5.6 所示。在名称栏里为数据源起一个你自己喜欢的名字，这里我们起的名字是 **sun**(当然，如果你喜欢的话，可以把名字叫做 **moon**.)。这个数据源就是指某个数据库（将来随着计

算机的进步，我们也可能有能力把数据源设成是一个卫星上来的信号)。在“你想连接哪个 SQL Server？”栏中选择或输入一个数据库服务器，这里我们选择了网络上的另一台机器：Ping。

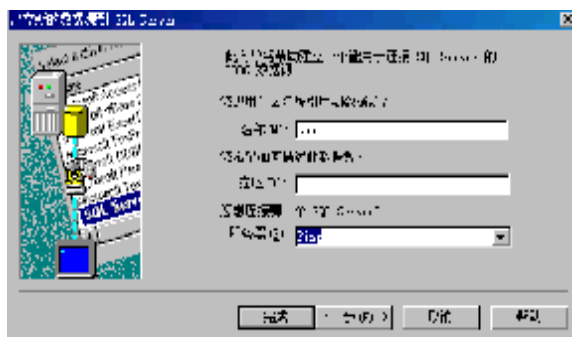
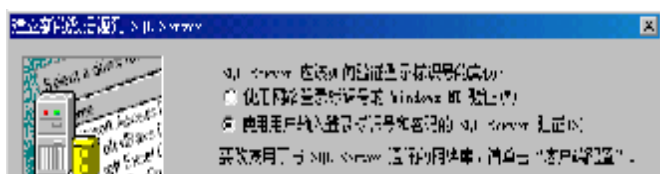


图 5.6 设置数据源的名字和所在服务器

单击“下一步”出现图 5.7 画面，选择连接 SQL Server 的 ID。

在图 5.7 的对话框中，选择“使用用户输入登录标识号和密码的 SQL Server 验证”选项，在这里我们选择用户名为 sa (不需要密码)，单击“下一步”出现如图 5.8 所示的选择数据库的对话框。

选中“改变默认的数据库为”复选框，在下拉菜单里，我们选择用户 sa 有权限操作的数据库 pubs。单击“下一步”出现完成数据源设置的对话框如图 5.9。



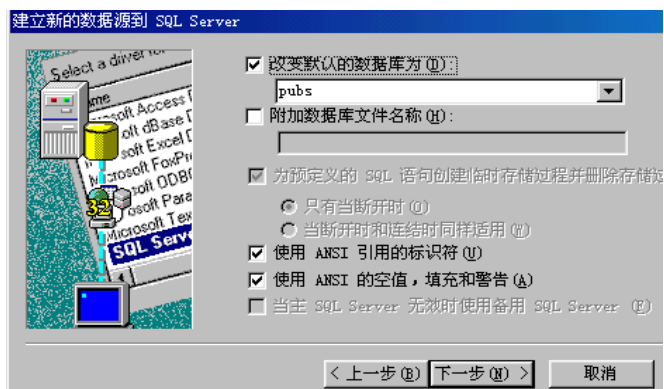
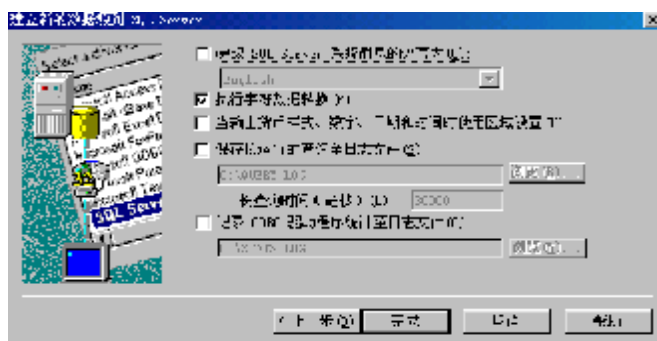


图 5.8 选择数据库



在图 5.9 中，单击“完成”出现你所配置的数据源的信息窗口，如图 5.10 所示。

点击“测试数据源”按钮，如果正常就会出现数据源设置成功的窗口，如图 5.11 所示。



图 5.10 数据源信息

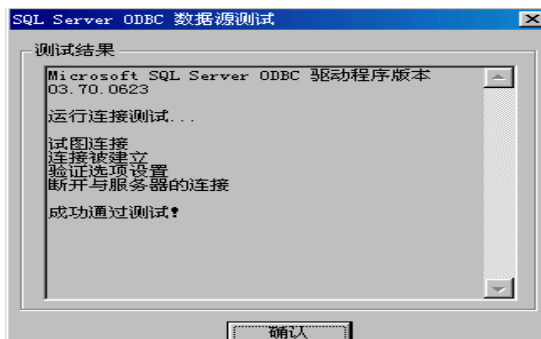


图 5.11 测试数据源

5.2 JDBC-ODBC 桥接器

现在你可以这样的直观理解：我们有了一个数据源，这个数据源就是一个数据库。为了要连接到这个数据库，需要建立一个 JDBC—ODBC 桥接器，即加载桥接器驱动程序。

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

这里，**Class** 是包 **java.lang** 中的一个类，该类通过调用它的静态方法 **forName** 就可以建立 **JDBC-ODBC** 桥接器。建立桥接器时可能发生异常，所以建立桥接器的标准是：

```
try { Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
catch(ClassNotFoundException e)
    {}
```

5.3 查询记录

要查询数据库中的记录，必须和数据库建立连接，由于使用的是 **JDBC-ODBC** 方式访问数据库，那么就要与数据源建立连接。

(1) 连接到数据库

首先使用包 **java.sql** 中的 **Connection** 类声明一个对象，然后再使用类 **DriverManager** 调用它的静态方法 **getConnection** 创建这个连接对象：

```
Connection con = DriverManager.getConnection("jdbc:odbc:    数据 源名字 ","login
name", "password ");
```

假如您没有为数据源设置 **login name** 和 **password**，那么连接形式是：

```
Connection con = DriverManager. getConnection("jdbc:odbc:    数据 源名字 ", "",
    "");
```

与数据库 **pubs**(它就是数据源 **sun**)建立连接，格式如下：

```
try{ Connection con = DriverManager. getConnection("jdbc:odbc:sun", "sa", "") ;
    }
```

```
catch(SQLException e)
```

```
{}
```

这样就建立了到数据库 **pubs** 的连接。

(2) 向数据库发送 SQL 语句。

首先使用 **Statement** 声明一个 SQL 语句对象, 然后通过刚才创建的连接数据库的对象 **con** 调用方法 **createStatment()** 创建这个 SQL 语句对象。

```
try {Statement sql=con.createStatement();}
```

```
catch(SQLException e){}
```

(3) 处理查询结果

有了 SQL 语句对象后, 这个对象就可以调用相应的方法实现对数据库中表的查询和修改。并将查询结果存放在一个 **ResultSet** 类声明的对象中, 也就是说 SQL 语句对数据库的查询操作将返回一个 **ResultSet** 对象:

```
ResultSet rs=sql.executeQuery("SELECT * FROM 成绩表");
```

ResultSet 对象是以统一形式的列组织的数据行组成。**ResultSet** 对象一次只能看到一个数据行, 使用 **next()** 方法走到下一数据行, 获得一行数据后, **ResultSet** 对象可以使用 **getxxxx** 方法获得字段值, 将位置索引 (第一列使用 1, 第二列使用 2 等等) 或字段名传递给 **getxxxx** 方法的参数即可。表 5.1 给出了出了 **ResultSet** 对象的若干方法。

表 5.1: ResultSet 类的若干方法

返回类型	方法名称
boolean	next()
byte	getBytes(int columnIndex)
Date	getDate(int columnIndex)
double	getDouble(int columnIndex)
float	getFloat(int columnIndex)
int	getInt(int columnIndex)
long	getLong(int columnIndex)
String	getString(int columnIndex)
byte	getBytes(String columnName)
Date	getDate(String columnName)
double	getDouble(String columnName)
float	getFloat(String columnName)
int	getInt(String columnName)
long	getLong(String columnName)
String	getString(String

	columnName)
--	-------------

5.3.1 顺序查询

使用结果集 **Result** 的 **next()**方法，可以顺序的查询。一个结果集将游标最初定位在第一行的前面，第一次调用 **next()**方法使游标移动到第一行。**next()**方法返回一个 **boolean** 型数据，当游标移动到最后一行之后返回 **false**。

在下面的例子 1 中，我们查询数据库 **pubs**（数据源 **sun**）中 **students** 表里的包含全部字段的记录。



The screenshot shows a web browser window with the address bar displaying 'http://21.222.222.222/Example5_1.jsp'. The main content area displays a table with 5 columns: '学号' (Student ID), '姓名' (Name), '数学成绩' (Math Score), '英语成绩' (English Score), and '物理成绩' (Physics Score). The table contains 8 rows of data.

学号	姓名	数学成绩	英语成绩	物理成绩
199901	王小兰	89	90	88
199902	王小明	88	88	85
199903	李小明	77	86	86
199904	胡志明	85	87	85
199905	柳志明	88	80	77
199906	孙大明	66	78	45
199907	张向东	50	77	80
199908	钟志明	80	85	78

图 5.12 顺序查询

例子 1（效果如图 5.12 所示）

Example5_1.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY>

<% Connection con;
```

```

Statement sql;

ResultSet rs;

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

catch(ClassNotFoundException e){}

try { con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

    sql=con.createStatement();

    rs=sql.executeQuery("SELECT * FROM students");

    out.print("<Table Border>");

    out.print("<TR>");

        out.print("<TH width=100>"+ "        学号");

        out.print("<TH width=100>"+ "        姓名");

        out.print("<TH width=50>"+ "        数学成绩");

        out.print("<TH width=50>"+ "        英语成绩");

        out.print("<TH width=50>"+ "        物理成绩");

    out.print("</TR>");

    while(rs.next())

    { out.print("<TR>");

        out.print("<TD >"+rs.getString(1)+"</TD>");

        out.print("<TD >"+rs.getString(2)+"</TD>");

        out.print("<TD >"+rs.getInt("        数学成绩")+ "</TD>");

        out.print("<TD >"+rs.getInt("        英语成绩")+ "</TD>");

        out.print("<TD >"+rs.getInt("        物理成绩")+ "</TD>");

```

```

        out.print("</TR>");
    }

    out.print("</Table>");

    con.close();

}

catch(SQLException e1) {}

%>

</BODY>

</HTML>

```

在下面的例子 2 中查询“英语成绩”字段值大于 80 的记录，但只显示“姓名”字段（第 2 个字段）和“英语成绩”字段。

例子 2（效果如图 5.13 所示）

Example5_2.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY>

<% Connection con;

Statement sql;

ResultSet rs;

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

}

```



```

catch(ClassNotFoundException e){}

try { con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

    sql=con.createStatement();

    rs=sql.executeQuery("SELECT * FROM students WHERE      英语成绩 >=
80 ");

    out.print("<Table Border>");

    out.print("<TR>");

        out.print("<TH width=100>"+ "      姓名");

        out.print("<TH width=50>"+ "      英语成绩");

    out.print("</TR>");

    while(rs.next())

    { out.print("<TR>");

        out.print("<TD >"+rs.getString(2)+"</TD>");

        out.print("<TD >"+rs.getInt("      英语成绩")+"</TD>");

    out.print("</TR>") ;

    }

    out.print("</Table>");

    con.close();

}

catch(SQLException e1) {}

%>

</BODY>

</HTML>

```

地址 http://localhost:8080/Example5_2.jsp

姓名	英语成绩
张小盐	98
胡名明	97
柳名将	80
钟名伟	99

图 5.13 条件查询

5.3.2 游动查询

前面我们学习了使用 **Result** 的 **next()**方法顺序地查询数据，但有时候我们需要在结果集中前后移动、或显示结果集指定的一条记录等等。这时，我们必须返回一个可滚动的结果集。为了得到一个可滚动的结果集，和上一节不同的是，我们必须使用下述方法先获得一个 **Statement** 对象：

```
Statement stmt=con.createStatement(int type ,int concurrency);
```

然后，根据参数的 **type**、**concurrency** 的取值情况，**stmt** 返回相应类型的结果集：

```
ResultSet re=stmt.executeQuery(SQL 语句);
```

1 **type** 的取值决定滚动方式，取值可以是：

ResultSet.TYPE_FORWARD_ONLY ：结果集的游标只能向下滚动。

ResultSet.TYPE_SCROLL_INSENSITIVE ：结果集的游标可以上下移动，当数据库变化时，当前结果集不变。

ResultSet.TYPE_SCROLL_SENSITIVE ：返回可滚动的结果集，当数据库变化时，

当前结果集同步改变。

l **Concurrency** 取值决定是否可以用结果集更新数据库, **Concurrency** 取值:

ResultSet.CONCUR_READ_ONLY: 不能用结果集更新数据库中的表。

ResultSet.CONCUR_UPDATETABLE: 能用结果集更新数据库中的表。

滚动查询经常用到 **ResultSet** 的下述方法:

2 **public boolean previous()**: 将游标向上移动, 该方法返回 **boolean** 型数据, 当移到结果集第一行之前时返回 **false**.

2 **public void beforeFirst**: 将游标移动到结果集的初始位置, 即在第一行之前。

2 **public void afterLast()**: 将游标移到结果集最后一行之后。

2 **public void first()**: 将游标移到结果集的第一行。

2 **public void last()**: 将游标移到结果集的最后一行。

2 **public boolean isAfterLast()**: 判断游标是否在最后一行之后。

2 **public boolean isBeforeFirst()**: 判断游标是否在第一行之前

2 **public boolean isFirst()**: 判断游标是否指向结果集的第一行。

2 **public boolean isLast()**: 判断游标是否指向结果集的最后一行。

2 **public int getRow()**: 得到当前游标所指行的行号, 行号从 1 开始, 如果结果集没有行, 返回 0

2 **public boolean absolute(int row)**: 将游标移到参数 **row** 指定的行号。

注意, 如果 **row** 取负值, 就是倒数的行数, **absolute(-1)** 表示移到最后一行, **absolute(-2)** 表示移到倒数第 2 行。当移动到第一行前面或最后一行的后面时, 该方法返回 **false**。

在下面的例子中, 首先将游标移动到最后一行, 然后再获取行号, 这样就获得表中的记录数目。然后我们倒序输出结果集中的记录, 即首先输出最后一行。最后单独输出第 5 条记录。

地址: http://localhost:8080/Example5_3.jsp

该表有 108 条记录
现在逆序输出记录:

学号	姓名	数学成绩	英语成绩	物理成绩
199908	孙名伟	50	59	78
199917	范建东	51	77	90
199905	余大雷	68	73	48
199903	柳名辉	88	80	77
199904	胡名杰	68	67	88
199902	唐小江	77	66	56
199912	王六炎	61	73	85
199901	张永强	80	68	88

现在输出第 5 条记录
199905, 柳名辉, 88, 80, 77。

图 5.14 游动查询

例子 3 (效果如图 5.14 所示)

Example5_3.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY>

<% String name,number;

    int math,physics,english;

    Connection con;

    Statement sql;

    ResultSet rs;

    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        }
```

```

catch(ClassNotFoundException e){}

try{ con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

    sql=

con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_R
EAD_ONLY);

    //    返回可滚动的结果集:

    rs=sql.executeQuery("SELECT * FROM students");

    //    将游标移动到最后一行:

    rs.last();

    //    获取最后一行的行号:

    int lownumber=rs.getRow();

    out.print("    该表共有"+lownumber+"条记录");

    out.print("<BR>    现在逆序输出记录: ");

    out.print("<Table Border>");

    out.print("<TR>");

        out.print("<TH width=100>"+ "        学号");

        out.print("<TH width=100>"+ "        姓名");

        out.print("<TH width=50>"+ "        数学成绩");

        out.print("<TH width=50>"+ "        英语成绩");

        out.print("<TH width=50>"+ "        物理成绩");

    out.print("</TR>");

    //    为了逆序输出记录, 需将游标移动到最后一行之后:

    rs.afterLast();

```

```

while(rs.previous())

{ out.print("<TR>");

    number=rs.getString(1);

    out.print("<TD >" + number + "</TD>");

    name=rs.getString(2);

    out.print("<TD >" + name + "</TD>");

    math=rs.getInt("        数学成绩");

    out.print("<TD >" + math + "</TD>");

    english=rs.getInt("        英语成绩");

    out.print("<TD >" + english + "</TD>");

    physics=rs.getInt("        物理成绩");

    out.print("<TD >" + physics + "</TD>");

    out.print("</TR>") ;

}

out.print("</Table>");

out.print("        单独输出第 5 条记录<BR>");

rs.absolute(5);

    number=rs.getString(1);

    out.print(number+",");

    name=rs.getString(2);

    out.print(name+",");

    math=rs.getInt("        数学成绩");

    out.print(math+",");

```

```

        english=rs.getInt("        英语成绩");

        out.print(english+",");

        physics=rs.getInt("        物理成绩");

        out.print(physics+"        。 ");

    con.close();

}

catch(SQLException e1) {}

%>

</BODY>

</HTML>

```

5.3.3 随机查询

在下面的例子中，我们随机从结果集中取出 4 条记录，并计算 4 条记录的数学成绩的平均值。用 **Math** 类的静态方法 **random()** 可以产生一个大于 0 小于 1 的随机数，再用下述公式：

```
int i=(int)(Math.random()*number+1);
```

产生一个 1 到 **number** 之间的随机数，根据这个随机数将游标移动到相应的行，并输出该行，算法的进一步细节可见下述例子 4。

地址  http://localhost:8080/Example5_4.jsp		
学号	姓名	数学成绩
199904	胡名明	66
199907	语境竟	55
199903	历小记	77
199901	张小盐	89

平均成绩是： 71.75

图 5.15 随机查询

例子 4（效果如图 5.15 所示）

Example5_4.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY>

<% String xuehao,name;

    int math;

    Connection con;

    Statement sql;

    ResultSet rs;

    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        }

    catch(ClassNotFoundException e){}

    try { con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

        sql=

        con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_R
EAD_ONLY);
```



```

//    返回可滚动的结果集:

rs=sql.executeQuery("SELECT * FROM students");

out.print("<Table Border>");

out.print("<TR>");

    out.print("<TH width=100>"+        学号");

    out.print("<TH width=100>"+        姓名");

    out.print("<TH width=50>"+        数学成绩");

out.print("</TR>");

//    将游标移动到最后一行:

    rs.last();

//    获取最后一行的行号:

    int lownumber=rs.getRow();

//    获取记录数:

int number=lownumber;

double sum=0;

int    抽取数目=4;

int old_i[]={0,0,0,0};

int k=    抽取数目;

int j=0;

    while(    抽取数目>0)

        {int i=(int)(Math.random()*number+1);//        随机获取一个 1 到 number

之间的数。

        boolean boo=false;

```

```

for(int m=0;m<old_i.length;m++) //          查找该行是否已被取出。

    {if(i==old_i[m])

        boo=true;

    }

if(boo) continue; //          假如该行已被取出，结束本次循环，继续
产生随机数。

rs.absolute(i);          //          游标移到这一行。

out.print("<TR>");

xuehao=rs.getString(1);          //          获取该行学号字段的值。

out.print("<TD >" + xuehao + "</TD>");

name=rs.getString(2);          //          获取该行姓名字段的值。

out.print("<TD >" + name + "</TD>");

math=rs.getInt("          数学成绩"); //          获取该行数学成绩字段的
值。

out.print("<TD >" + math + "</TD>");

out.print("</TR>");

sum=sum+math;

        抽取数目--;

old_i[j]=i;          //          记录已取出的行号。

j++;

}

out.print("</Table>");

```

```

        out.print("        平均成绩是: "+sum/k);

        con.close();

    }

    catch(SQLException e1) {}

%>

</BODY>

</HTML>

```

5.3.4 参数查询

在下面的例子中，客户通过 **Example5_5.jsp** 页面输入查询条件，如按姓名查询成绩或按分数段查询学生成绩等等。输入姓名提交给 **byname.jsp** 页面，输入分数段提交给 **byscore.jsp** 页面。

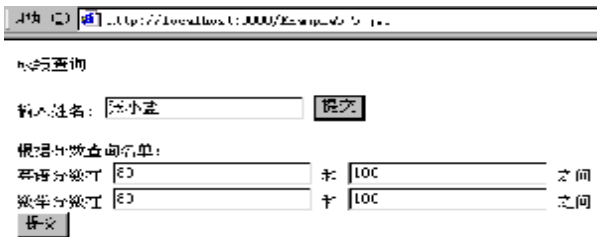


图 5.16 参数查询

学号	姓名	数学成绩	英语成绩	物理成绩
10000	张小北	89	88	100

图 5.17 根据名字查询

学号	姓名	数学成绩	英语成绩	物理成绩
10000	张小北	89	88	100
10001	陈纳德	88	80	77
10002	马占洋	80	89	71

图 5.18 根据分数段查询

例子 5（效果如图 5.16、5.17、5.18 所示）

Example5_5.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<Font size=1>

<FORM action="byname.jsp" Method="post">

<P> 成绩查询

<P> 输入姓名:

<Input type=text name="name">

<Input type=submit name="g" value=" 提交">

</Form>

<FORM action="byscore.jsp" Method="post" >

<P> 根据分数查询名单:<BR> 英语分数在

<Input type=text name="englishmin" value=0>

和

<Input type=text name="englishmax" value=100>

之间

<BR> 数学分数在
```

<Input type=text name="mathmin" value=0>

和

<Input type=text name="mathmax" value=100>

之间

**
 <Input type=submit value=" 提交">**

</Form>

</BODY>

</HTML>

bynename.jsp:

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY>

<% // 获取提交的姓名:

String name=request.getParameter("name");

if(name==null)

{name="";

}

byte b[]=name.getBytes("ISO-8859-1");

name=new String(b);

Connection con=null;

Statement sql=null;

```

ResultSet rs=null;

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

catch(ClassNotFoundException e){}

try { con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

    sql=con.createStatement();

    String condition="SELECT * FROM students WHERE          姓 名 =
"+'''+name+''';

    rs=sql.executeQuery(condition);

    out.print("<Table Border>");

    out.print("<TR>");

    out.print("<TH width=100>"+          学号");
    out.print("<TH width=100>"+          姓名");
    out.print("<TH width=50>"+          数学成绩");
    out.print("<TH width=50>"+          英语成绩");
    out.print("<TH width=50>"+          物理成绩");

    out.print("</TR>");

    while(rs.next())

        { out.print("<TR>");

            out.print("<TD >"+rs.getString(1)+"</TD>");

            out.print("<TD >"+rs.getString(2)+"</TD>");

            out.print("<TD >"+rs.getInt("          数学成绩")+"</TD>");

            out.print("<TD >"+rs.getInt("          英语成绩")+"</TD>");

```

```

        out.print("<TD >" + rs.getInt("          物理成绩") + "</TD>");

        out.print("</TR>");

    }

    out.print("</Table>");

    con.close();

}

catch(SQLException e)

{ }

%>

</BODY>

</HTML>

```

byscore.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY>

<% // 获取提交的分数的最大值和最小值:

    String englishmax=request.getParameter("englishmax");

    if(englishmax==null)

        {englishmax="100";

        }

    String englishmin=request.getParameter("englishmin");

```

```

        if(englishmin==null)

            {englishmin="0";

            }

String mathmax=request.getParameter("mathmax");

        if(mathmax==null)

            {mathmax="100";

            }

String mathmin=request.getParameter("mathmin");

        if(mathmin==null)

            {mathmin="0";

            }

Connection con=null;

Statement sql=null;

ResultSet rs=null;

        try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        }

        catch(ClassNotFoundException e){}

        try

        {   con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

            sql=con.createStatement();

            String eCondition="          英语成绩 <= "+englishmax+" AND "+"          英语成绩

            >= "+englishmin;

            String mCondition="          数学成绩 <= "+mathmax+" AND "+"          数学成绩 >=

```



```

"+mathmin;

String condition="SELECT * FROM students WHERE "+mCondition+"
and "+eCondition;

rs=sql.executeQuery(condition);

out.print("<Table Border>");

out.print("<TR>");

out.print("<TH width=100>"+ "      学号");
out.print("<TH width=100>"+ "      姓名");
out.print("<TH width=50>"+ "      数学成绩");
out.print("<TH width=50>"+ "      英语成绩");
out.print("<TH width=50>"+ "      物理成绩");

out.print("</TR>");

while(rs.next())

{ out.print("<TR>");

out.print("<TD >"+rs.getString(1)+"</TD>");

out.print("<TD >"+rs.getString(2)+"</TD>");

out.print("<TD >"+rs.getInt("      数学成绩")+"</TD>");
out.print("<TD >"+rs.getInt("      英语成绩")+"</TD>");
out.print("<TD >"+rs.getInt("      物理成绩")+"</TD>");

out.print("</TR>") ;

}

out.print("</Table>");

con.close();

```

```

    }

    catch(SQLException e)

    {}

%>

</BODY>

</HTML>

```

5.3.5 排序查询

可以在 SQL 语句中使用 **ORDER BY** 子语句，对记录排序。在下面的例子中，使用 SQL 语句的 **ORDER BY** 子语句查询所同学的成绩，可以选择按 3 科的总分从低到高排列记录、按姓氏拼音排序或英语成绩排序。

例如，按总成绩排序查询的 SQL 语句：

SELECT * FROM student ORDER BY 数学成绩+英语成绩+物理成绩。

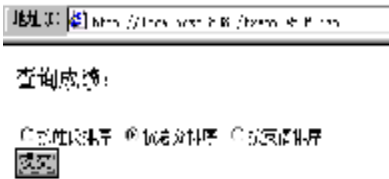


图 5.19 选择排序查询方式

学号	姓名	数学成绩	英语成绩	物理成绩	总成绩
199501	王小明	85	78	72	335
199502	李小红	75	68	62	305
199503	张小明	85	77	82	444
199504	王小红	88	78	82	448
199505	李小明	85	78	72	335
199506	王小红	85	77	82	444
199507	王小红	85	77	82	444
199508	王小红	85	77	82	444
199509	王小红	85	77	82	444
199510	王小红	85	77	82	444

图 5.20 按总分排序查询

例子 6（效果如图 5.19、5.20 所示）

Example5_6:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<P>查询成绩:

<Font size=1>

<FORM action="byname1.jsp" method=post name=form>

    <INPUT type="radio" name="R" value="    姓名">按姓氏排序

    <INPUT type="radio" name="R" value="    数学成绩+英语成绩+物理成绩">
```

按总分排序

```
    <INPUT type="radio" name="R" value="    英语成绩">按英语排序

<BR> <Input type=submit name="g" value=" 提交">

</Form>

</BODY>

</HTML>
```

byname1.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY>

    <% // 获取提交的排序方式:

        String name=request.getParameter("R");
```

```

    if(name==null)

        {name="";}

byte b[]=name.getBytes("ISO-8859-1");

name=new String(b);

String number,xingming;

Connection con=null;

Statement sql=null;

ResultSet rs=null;

int math,english,physics;

    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        }

    catch(ClassNotFoundException e){}

try { con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

    sql=con.createStatement();

    String condition="SELECT * FROM students ORDER BY "+name;

    rs=sql.executeQuery(condition);

    out.print("<Table Border>");

        out.print("<TR>");

        out.print("<TH width=100>"+ "        学号");

        out.print("<TH width=100>"+ "        姓名");

        out.print("<TH width=50>"+ "        数学成绩");

        out.print("<TH width=50>"+ "        英语成绩");

        out.print("<TH width=50>"+ "        物理成绩");

```

```

        out.print("<TH width=50>"+ "          总成绩");

        out.print("</TR>");

        while(rs.next())

        { out.print("<TR>");

            number=rs.getString(1);

            out.print("<TD >"+number+"</TD>");

            xingming=rs.getString(2);

            out.print("<TD >"+xingming+"</TD>");

            math=rs.getInt("          数学成绩");

            out.print("<TD >"+math+"</TD>");

            english=rs.getInt("          英语成绩");

            out.print("<TD >"+english+"</TD>");

            physics=rs.getInt("          物理成绩");

            out.print("<TD >"+physics+"</TD>");

            int total=math+english+physics;

            out.print("<TH >"+total+"</TH>");

            out.print("</TR>") ;

        }

        out.print("</Table>");

        con.close();

    }

    catch(SQLException e)

    { }

```

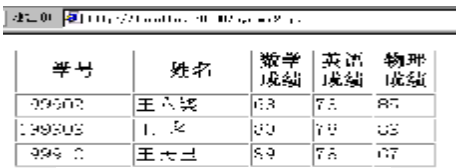
```
%>

</BODY>

</HTML>
```

5.3.6 分析结果集查询

通过分析结果集来输出记录。在下面的例子中查询所有姓王的同学的成绩，首先判断结果集中，姓氏字段的值是否是某个姓氏，然后输出全部该姓氏的同学的成绩。



The screenshot shows a web browser window with a table containing student records. The table has five columns: '学号' (Student ID), '姓名' (Name), '数学成绩' (Math Score), '英语成绩' (English Score), and '物理成绩' (Physics Score). There are three rows of data, all of which have the surname '王' (Wang).

学号	姓名	数学成绩	英语成绩	物理成绩
099002	王小明	63	72	85
099005	王小红	89	78	88
099001	王大山	89	76	87

图 5.21 查找指定姓氏记录

例子 7（效果如图 5.21 所示）

Example5_7.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY>

<P>查询成绩:

<Font size=1>

<P>输入学生的姓氏:

<BR> <FORM action="byname2.jsp" method=post name=form>

    <INPUT type="text" name="name" value="    王">
```

```
<BR> <Input type=submit name="g" value=" 提交">

</Form>

</BODY>

</HTML>
```

byname2.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY>

<% // 获取提交的姓氏:

String name=request.getParameter("name");

    if(name==null)

        {name="";

        }

byte b[]=name.getBytes("ISO-8859-1");

name=new String(b);

String number,xingming;

Connection con=null;

Statement sql=null;

ResultSet rs=null;

int math,english,physics;

    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```

    }

    catch(ClassNotFoundException e){}

    try{ con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

        sql=con.createStatement();

        String condition="SELECT * FROM students" ;

        rs=sql.executeQuery(condition);

    out.print("<Table Border>");

        out.print("<TR>");

        out.print("<TH width=100>"+ "        学号");
        out.print("<TH width=100>"+ "        姓名");
        out.print("<TH width=50>"+ "        数学成绩");
        out.print("<TH width=50>"+ "        英语成绩");
        out.print("<TH width=50>"+ "        物理成绩");

        out.print("</TR>");

    while(rs.next())

        { number=rs.getString(1);

            xingming=rs.getString(2);

            if(xingming.startsWith("        王"))

                { out.print("<TR>");

                    out.print("<TD >"+number+"</TD>");

                    out.print("<TD >"+xingming+"</TD>");

                    math=rs.getInt("        数学成绩");

                    out.print("<TD >"+math+"</TD>");

```



```

        english=rs.getInt("        英语成绩");

        out.print("<TD >" + english + "</TD>");

        physics=rs.getInt("        物理成绩");

        out.print("<TD >" + physics + "</TD>");

        out.print("</TR>");

    }

}

out.print("</Table>");

con.close();

}

catch(SQLException e)

{ }

%>

</BODY>

</HTML>

```

5.3.7 使用通配符查询

可以用 SQL 语句操作符 **LIKE** 进行模式般配，使用 “%” 代替一个或多个字符，用一个下划线 “_” 代替一个字符。比如，下述语句查询姓氏是 “王” 的记录：

```

rs=sql.executeQuery("SELECT * FROM students WHERE        姓名 LIKE '    王%'
");

```

请将上面的例子 7 改为通配符查询，查询所有姓氏是王的记录。

5.4 更新记录

我们可以使用 SQL 语句更新记录中字段的值

Statement 对象调用方法：

```
public int executeUpdate (String sqlStatement) ;
```

通过参数 `sqlStatement` 指定的方式实现对数据库表中记录的字段值的更新，例如，下述语句将表 `students` 中王名同学的数学字段的值更新 `88`：

```
executeUpdate("UPDATE students SET 数学成绩 = 88 WHERE 姓名='王名'");
```

注：你可以使用一个 `Statement` 对象进行更新和查询操作，但需要注意的是，当查询语句返回结果集后，没有立即输出结果集的记录，而接着执行了更新语句，那么结果集就不能输出记录了。要想输出记录就必须重新返回结果集。

在下面的例子 8 中，可以更新学生的各科的成绩。在 `Example5_8.jsp` 页面提交学生的姓名以及这个学生新的成绩到 `newResult.jsp` 页面，该页面负责更新记录的字段值。



Figure 5.22 shows two browser windows. The left window displays a form for updating a student's record and a table of existing records. The right window displays the updated table after a record has been modified.

学号	姓名	数学成绩	英语成绩	物理成绩
1989001	王小华	88	88	100
1989002	王大明	88	88	88
1989003	王小明	88	88	88
1989004	王小明	88	88	88

学号	姓名	数学成绩	英语成绩	物理成绩
1989001	王小华	100	100	95
1989002	王大明	88	88	88
1989003	王小明	88	88	88
1989004	王小明	88	88	88

图 5.22 提交字段的新值

图 5.23 更新后的字段值

例子 8（效果如图 5.22、5.23 所示）

Example5_8.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import="java.sql.*" %>
```

```
<HTML>
```

```
<BODY bgcolor=pink >
```

```
<Font size=1>
```

```
<FORM action="newResult.jsp" method=post>
```

输入要修改成绩的同学的姓名:


```
<Input type="text" name="name">
```


输入新的数学成绩:

```
<Input type="text" name="math">
```


输入新的英语成绩:

```
<Input type="text" name="english">
```


输入新的物理成绩:

```
<Input type="text" name="physics">
```


<Input type="submit" name="b" value="提交更新">

<P>数据库更新前的数据记录是:

```
<% String name,number;

int math,physics,english;

Connection con;

Statement sql;

ResultSet rs;

try{Class.forName('sun.jdbc.odbc.JdbcOdbcDriver');

    }

catch(ClassNotFoundException e){}

try{ con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

    sql=con.createStatement();

    rs=sql.executeQuery("SELECT * FROM students");

    out.print("<Table Border>");

    out.print("<TR>");

        out.print("<TH width=100>"+ "        学号");

        out.print("<TH width=100>"+ "        姓名");

        out.print("<TH width=50>"+ "        数学成绩");

        out.print("<TH width=50>"+ "        英语成绩");

        out.print("<TH width=50>"+ "        物理成绩");

    out.print("</TR>");

    while(rs.next())

    { out.print("<TR>");
```

```

        number=rs.getString(1);

        out.print("<TD >" + number + "</TD>");

        name=rs.getString(2);

        out.print("<TD >" + name + "</TD>");

        math=rs.getInt("        数学成绩");

        out.print("<TD >" + math + "</TD>");

        english=rs.getInt("        英语成绩");

        out.print("<TD >" + english + "</TD>");

        physics=rs.getInt("        物理成绩");

        out.print("<TD >" + physics + "</TD>");

        out.print("</TR>");
    }

    out.print("</Table>");

    con.close();

}

catch(SQLException e1) {}

%>

</Font>

</BODY>

</HTML>

```

newResult.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

```

```
<%@ page import="java.sql.*" %>

<HTML>

<BODY bgcolor=pink ><Font size=1>

<% // 获取提交的姓名:

String name=request.getParameter("name");

    if(name==null)

        {name="";

        }

byte b[]=name.getBytes("ISO-8859-1");

name=new String(b);

// 获取提交的新的数学成绩:

String newMath=request.getParameter("math");

    if(newMath==null)

        {newMath="-100";

        }

// 获取提交的新的英语成绩:

String newEnglish=request.getParameter("english");

    if(newEnglish==null)

        {newEnglish="-100";

        }

// 获取提交的新的物理成绩:

String newPhysics=request.getParameter("physics");

    if(newPhysics==null)
```

```

        {newPhysics="-100";

        }

Connection con=null;

Statement sql=null;

ResultSet rs=null;

String number,xingming;

int math,english,physics;

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

catch(ClassNotFoundException e){}

try {con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

sql=con.createStatement();

String condition1=

        "UPDATE students SET          数学成绩 = "+newMath+" WHERE   姓名

="+""+name+"" ,

        condition2=

        "UPDATE students SET          英语成绩 = "+newEnglish+" WHERE   姓

名="+""+name+"" ,

        condition3=

        "UPDATE students SET          物理成绩 = "+newPhysics+" WHERE   姓

名="+""+name+"" ;

//    执行更新操作：

sql.executeUpdate(condition1);

```

```

sql.executeUpdate(condition2);

sql.executeUpdate(condition3);

//    显示更新后的表中的记录:

%>

<P>    更新后的表的记录:

<%

rs=sql.executeQuery("SELECT * FROM students");

out.print("<Table Border>");

    out.print("<TR>");

    out.print("<TH width=100>"+ "        学号");

    out.print("<TH width=100>"+ "        姓名");

    out.print("<TH width=50>"+ "        数学成绩");

    out.print("<TH width=50>"+ "        英语成绩");

    out.print("<TH width=50>"+ "        物理成绩");

    out.print("</TR>");

while(rs.next())

{

    out.print("<TR>");

    number=rs.getString(1);

    out.print("<TD >"+number+"</TD>");

    xingming=rs.getString(2);

    out.print("<TD >"+xingming+"</TD>");

    math=rs.getInt("        数学成绩");

```



```

        out.print("<TD >" + math + "</TD>");

        english=rs.getInt("          英语成绩");

        out.print("<TD >" + english + "</TD>");

        physics=rs.getInt("          物理成绩");

        out.print("<TD >" + physics + "</TD>");

        out.print("</TR>");

    }

    out.print("</Table>");

    con.close();

}

catch(SQLException e)

{

}

%>

</FONT>

</BODY>

</HTML>

```

5.5 添加记录

我们可以使用 SQL 语句添加新的记录，Statement 对象调用方法：

```
public int executeUpdate (String sqlStatement) ;
```

通过参数 sqlStatement 指定的方式实现向数据库表中添加新记录，例如，下述语句将向表 students 中添加一条新的记录：（'199911'，'美丽家'，100,99,98）。

```
executeUpdate("INSERT INTO students VALUES ('199911','美丽家',100,99,98)");
```

在下面的例子 9 中，可以添加新的记录。在 `Example5_9.jsp` 页面提交新的记录到 `newDatabase.jsp` 页面，该页面负责添加新的记录。

注：你可以使用一个 `Statement` 对象进行添加和查询操作，但需要注意的是，当查询语句返回结果集后，没有立即输出结果集的记录，而接着执行了添加语句，那么结果集就不能输出记录了。要想输出记录就必须重新返回结果集。

例子 9（效果如图 5.24、5.25 所示）

`Example5_9.jsp`:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import="java.sql.*" %>
```

```
<HTML>
```

```
<BODY bgcolor=pink >
```

```
<Font size=1>
```

```
<P>添加新的记录到数据库：
```

```
<FORM action="newDatabase.jsp" method=post>
```

同学学号：

```
<Input type="text" name="number">
```


同学姓名：

```
<Input type="text" name="name">
```


数学成绩：

<Input type="text" name="math">

**
英语成绩:**

<Input type="text" name="english">

**
物理成绩:**

<Input type="text" name="physics">

**
<Input type="submit" name="b" value="提交添加">**

<P>数据库添加记录前的数据记录是:

<% String name,number;

int math,physics,english;

Connection con;

Statement sql;

ResultSet rs;

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

}

catch(ClassNotFoundException e){}

try { con=DriverManager.getConnection("jdbc:odbc:moon","sa","");

sql=con.createStatement();

rs=sql.executeQuery("SELECT * FROM students");

out.print("<Table Border>");

out.print("<TR>");

out.print("<TH width=100>"+ "学号");

out.print("<TH width=100>"+ "姓名");

out.print("<TH width=50>"+ "数学成绩");

```

        out.print("<TH width=50>"+ "        英语成绩");

        out.print("<TH width=50>"+ "        物理成绩");

        out.print("</TR>");

        while(rs.next())

        { out.print("<TR>");

            number=rs.getString(1);

            out.print("<TD >"+number+"</TD>");

            name=rs.getString(2);

            out.print("<TD >"+name+"</TD>");

            math=rs.getInt("        数学成绩");

            out.print("<TD >"+math+"</TD>");

            english=rs.getInt("        英语成绩");

            out.print("<TD >"+english+"</TD>");

            physics=rs.getInt("        物理成绩");

            out.print("<TD >"+physics+"</TD>");

            out.print("</TR>") ;

        }

        out.print("</Table>");

        con.close();

    }

    catch(SQLException e1) {}

    %>

</Font>

```

</BODY>

</HTML>

newDatabase.jsp :

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY bgcolor=pink >

<% // 获取提交的学号:

String number=request.getParameter("number");

if(number==null)

{number="";

}

byte b[]=number.getBytes("ISO-8859-1");

number=new String(b);

// 获取提交的姓名:

String name=request.getParameter("name");

if(name==null)

{name="";

}

byte c[]=name.getBytes("ISO-8859-1");

name=new String(c);

// 获取提交的新的数学成绩:

```

String m=request.getParameter("math");

    if(m==null)

        {m="-100"; }

// 获取提交的新的英语成绩:

String e=request.getParameter("english");

    if(e==null)

        {e="-100"; }

// 获取提交的新的物理成绩:

String p=request.getParameter("physics");

    if(p==null)

        {p="-100"; }

Connection con=null;

Statement sql=null;

ResultSet rs=null;

    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); }

    catch(ClassNotFoundException event){}

try {con=DriverManager.getConnection("jdbc:odbc:moon","sa","");

    sql=con.createStatement();

    String condition=

        "INSERT INTO students

VALUES"+"("+"'"+number+"','"+name+"','"+m+"','"+e+"','"+p+"')";

    sql.executeUpdate(condition); // 执行添加操作:

// 显示添加新记录后表中的记录:

```

%>

<P> 添加新记录后的表:

<%

```
rs=sql.executeQuery("SELECT * FROM students ORDER BY 学号 ");
```

```
out.print("<Table Border>");
```

```
out.print("<TR>");
```

```
out.print("<TH width=100>"+ "学号");
```

```
out.print("<TH width=100>"+ "姓名");
```

```
out.print("<TH width=50>"+ "数学成绩");
```

```
out.print("<TH width=50>"+ "英语成绩");
```

```
out.print("<TH width=50>"+ "物理成绩");
```

```
out.print("</TR>");
```

```
while(rs.next())
```

```
{ out.print("<TR>");
```

```
String n=rs.getString(1);
```

```
out.print("<TD >"+n+"</TD>");
```

```
String xingming=rs.getString(2);
```

```
out.print("<TD >"+xingming+"</TD>");
```

```
int math=rs.getInt("数学成绩");
```

```
out.print("<TD >"+math+"</TD>");
```

```
int english=rs.getInt("英语成绩");
```

```
out.print("<TD >"+english+"</TD>");
```

```
int physics=rs.getInt("物理成绩");
```

```

        out.print("<TD >" + physics + "</TD>");

        out.print("</TR>");
    }

    out.print("</Table>");

    con.close();

}

catch(SQLException event)

{

%>

</FONT>

</BODY>

</HTML>

```



图 5.24 提交添加的记录数据



图 5.25 添加记录

5.6 删除记录

我们可以使用 SQL 语句删除记录，Statement 对象调用方法：

```
public int executeUpdate (String sqlStatement) ;
```

通过参数 sqlStatement 指定的方式删除数据库表中的记录，例如，下述语句将删除学号是 199904 的记录：

```
executeUpdate("DELETE FROM students WHERE 学号 = '199904'");
```

在下面的例子 10 中，可以删除已有的记录。在 Example5_10.jsp 页面提交要被删除的记录的字段值到 delete.jsp 页面，delete.jsp 页面负责删除相应的记录。

注：你可以使用一个 Statement 对象进行删除和查询操作，但需要注意的是，当查询语句返回结果集后，没有立即输出结果集的记录，而接着执行了删除语句，那么结果集就不能输出记录了。要想输出记录就必须重新返回结果集。

学号	姓名	数学成绩	英语成绩	物理成绩
199901	张小华	80	78	87
199902	王大刚	86	80	82
199903	陈小红	77	68	58
199904	胡志明	90	87	83
199905	柳名松	80	80	77

图 5.26 提交预删除记录的学号

学号	姓名	数学成绩	英语成绩	物理成绩
199901	张小华	80	78	87
199902	王大刚	86	80	82
199903	陈小红	77	68	58
199905	柳名松	80	80	77
199906	王大刚	80	78	85
199907	高亮	85	77	90

图 5.27 删除记录

例子 10（效果如图 5.26、5.27 所示）

Example5_10.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY bgcolor=pink >

<Font size=1>

<P>删除记录:

<FORM action="delete.jsp" method=post>

输入被删除记录的学号:

<Input type="text" name="number">

<BR>

<BR><Input type="submit" name="b" value="提交删除">

</FORM>

<P>数据库删除前的数据记录是:

<% String name,number;

    int math,physics,english;

    Connection con;
```

```

Statement sql;

ResultSet rs;

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

catch(ClassNotFoundException e){}

try { con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

    sql=con.createStatement();

    rs=sql.executeQuery("SELECT * FROM students");

    out.print("<Table Border>");

    out.print("<TR>");

        out.print("<TH width=100>"+ "        学号");

        out.print("<TH width=100>"+ "        姓名");

        out.print("<TH width=50>"+ "        数学成绩");

        out.print("<TH width=50>"+ "        英语成绩");

        out.print("<TH width=50>"+ "        物理成绩");

    out.print("</TR>");

while(rs.next())

{ out.print("<TR>");

    number=rs.getString(1);

    out.print("<TD >"+number+"</TD>");

    name=rs.getString(2);

    out.print("<TD >"+name+"</TD>");

    math=rs.getInt("        数学成绩");

```

```

        out.print("<TD >" + math + "</TD>");

        english=rs.getInt("        英语成绩");

        out.print("<TD >" + english + "</TD>");

        physics=rs.getInt("        物理成绩");

        out.print("<TD >" + physics + "</TD>");

        out.print("</TR>");

    }

    out.print("</Table>");

    con.close();

}

catch(SQLException e1) {}

%>

</Font>

</BODY>

</HTML>

```

delete.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY bgcolor=pink ><Font size=1>

    <% // 获取提交的学号:

        String number=request.getParameter("number");

```

```

        if(number==null)

            {number="";

            }

byte b[]=number.getBytes("ISO-8859-1");

number=new String(b);

Connection con=null;

Statement sql=null;

ResultSet rs=null;

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

catch(ClassNotFoundException event){}

try {con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

sql=con.createStatement();

//    删除操作：

String deleteAll="DELETE FROM students WHERE

学号 "+"=

"+"+"'+number+'";

sql.executeUpdate(deleteAll);

%>

<P>    删除记录后的表：

<%

rs=sql.executeQuery("SELECT * FROM students ORDER BY

学号 ");

out.print("<Table Border>");

out.print("<TR>");

```

```

        out.print("<TH width=100>"+ "        学号");
        out.print("<TH width=100>"+ "        姓名");
        out.print("<TH width=50>"+ "        数学成绩");
        out.print("<TH width=50>"+ "        英语成绩");
        out.print("<TH width=50>"+ "        物理成绩");
        out.print("</TR>");

while(rs.next())
{
    out.print("<TR>");

    String n=rs.getString(1);

    out.print("<TD >"+n+"</TD>");

    String xingming=rs.getString(2);

    out.print("<TD >"+xingming+"</TD>");

    int math=rs.getInt("        数学成绩");

    out.print("<TD >"+math+"</TD>");

    int english=rs.getInt("        英语成绩");

    out.print("<TD >"+english+"</TD>");

    int physics=rs.getInt("        物理成绩");

    out.print("<TD >"+physics+"</TD>");

    out.print("</TR>");

}

out.print("</Table>");

con.close();

}

```

```

catch(SQLException event)

    {out.print("'" + event);

    }

%>

</FONT>

</BODY>

</HTML>

```

5.7 分页显示记录

我们简单地实现数据库表中记录的分页显示。

假设总记录数为 m ，每页显示数量是 n ，那么总页数的计算公式是：

- (1) 如果 m 除以 n 的余数大于 0，总页数等于 m 除以 n 的商加 1；
- (2) 如果 m 除以 n 的余数等于 0，总页数等于 m 除以 n 的商。

即

总页数 = $(m \% n) == 0 ? (m / n) : (m / n + 1)$;

如果准备显示第 p 页的内容，应当把游标移动到第 **$(p - 1) * n + 1$** 条记录处。

showByPage.jsp：（效果如图 5.28 所示）

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

```

<BODY>

<%! int pageSize=3; // 每页显示的记录数。

int pageCount=0; // 分页后的总页数。

%>

<%-- 客户通过表单提交欲要显示的页码数--%>

<FORM action="" method=get >

输入页码数<Input Type=text name=showPage size=4 >

<Input Type=submit name=g value= 提交>

</FORM>

<% Connection con;

Statement sql;

ResultSet rs;

try{Class.forName('sun.jdbc.odbc.JdbcOdbcDriver');

}

catch(ClassNotFoundException e){}

try { con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

sql=

con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);

// 返回可滚动的结果集:

rs=sql.executeQuery("SELECT * FROM students");

// 将游标移动到最后一行:

rs.last();


```
// 获取最后一行的行号:
```

```
int lastRow=rs.getRow();
```

```
// 计算分页后的总页数:
```

```
pageCount=(lastRow%pageSize==0)?(lastRow/pageSize):(lastRow/pageSize+1);
```

```
// 当前显示的初始页数:
```

```
int showPage=1;
```

```
// 告知客户总页数:
```

```
%>
```

```
<P> 共有<%=pageCount%>页
```

```
<BR> 每页显示<%=pageSize%>条记录.
```

```
<% // 获取客户想要显示的页数:
```

```
String integer=request.getParameter("showPage");
```

```
if(integer==null)
```

```
{ integer="1";
```

```
}
```

```
try {showPage=Integer.parseInt(integer);
```

```
}
```

```
catch(NumberFormatException e)
```

```
{showPage=1;
```

```
}
```

```
if(showPage<=1)
```

```
{showPage=1;
```

```

    }

    if(showPage>=pageCount)
    {showPage=pageCount;
    }

    %>

    <BR>    目前显示第<%=showPage%>页

    <% //    如果要显示第 showPage 页，那么游标应移到 posion 的值是：

    int posion=(showPage-1)*pageSize+1;

    rs.absolute(posion); //        设置游标的位置

    out.print("<Table Border>");

    out.print("<TR>");

    out.print("<TH width=100>"+"        学号");
    out.print("<TH width=100>"+"        姓名");
    out.print("<TH width=50>"+"        数学成绩");
    out.print("<TH width=50>"+"        英语成绩");
    out.print("<TH width=50>"+"        物理成绩");

    out.print("</TR>");

    for (int i=1;i<=pageSize;i++)
    { out.print("<TR>");

        out.print("<TD >"+rs.getString(1)+"</TD>");

        out.print("<TD >"+rs.getString(2)+"</TD>");

        out.print("<TD >"+rs.getInt("        数学成绩")+"</TD>");

        out.print("<TD >"+rs.getInt("        英语成绩")+"</TD>");

```

```

        out.print("<TD >" + rs.getInt("        物理成绩") + "</TD>");

        out.print("</TR>");

        rs.next();

    }

    out.print("</Table>");

    con.close();

}

catch(SQLException e1) {}

%>

</BODY>

</HTML>

```

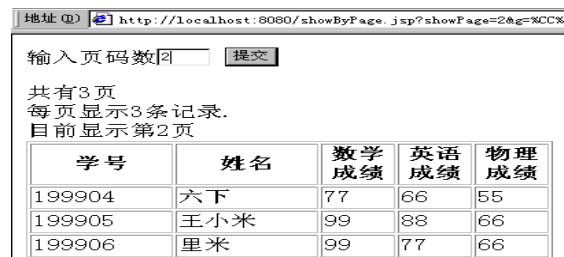


图 5.28 分页显示记录

5.8 连接数据库的其它方式

5.8.1 连接 Oracle 数据库

我们也可以通过 JDBC-ODBC 桥接器和 Oracle 数据库建立连接，但这种连接的质量

依赖于 ODBC。下面介绍通过直接加载 Oracle 数据库驱动程序来连接数据库的方法。

安装 Oracle 后，找到目录：Oracle/ora81/jdbc 中的文件：classes12.zip；将该文件拷贝到你的 JDK 的如下目录中：

```
f:/jdk1.3/jre/lib/ext
```

并将 classes.zip 重新命名为 classes.jre 或 classes.jar。

通过如下的两个步骤和一个 Oracle 数据库建立连接：

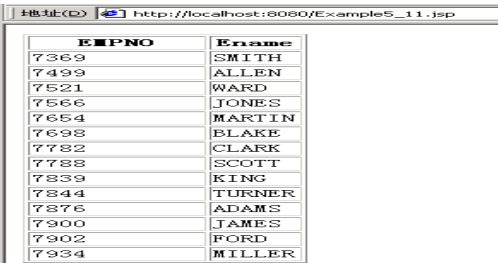
(1)加载驱动程序：

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

(2)建立连接：

```
Connection conn=
```

DriverManager.getConnection(“jdbc:oracle:thin:@ 主机 host:端口号:数据库名”，“用户名”，“密码”);



EMPNO	Ename
7369	SMITH
7499	ALLEN
7521	WARD
7566	JONES
7654	MARTIN
7698	BLAKE
7782	CLARK
7788	SCOTT
7839	KING
7844	TURNER
7876	ADAMS
7900	JAMES
7902	FORD
7934	MILLER

图 5.29 查询 Oracle 数据库

例子 11（效果如图 5.29 所示）

Example5_11.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY>

<% Connection con=null;

Statement sql=null;

ResultSet rs=null;

try{Class.forName("oracle.jdbc.driver.OracleDriver");

}

catch(ClassNotFoundException e){}

try

{ con=

DriverManager.getConnection("jdbc:oracle:thin:@192.168.0.35:1521:Lea","scott","ti

ger");

sql=con.createStatement();

rs=sql.executeQuery("select * from emp");

out.print("<Table Border>");

out.print("<TR>");

out.print("<TH width=100>"+ "EMPNO");

out.print("<TH width=50>"+ "Ename");
```

```

        out.print("</TR>");
    while(rs.next())
    { out.print("<TR>");
        int n=rs.getInt(1);
        out.print("<TD >"+n+"</TD>");
        String e=rs.getString(2);
        out.print("<TD >"+e+"</TD>");
        out.print("</TR>" );
    }
    out.print("</Table>");
    con.close();
}

catch(SQLException e1) {out.print(""+e1);}

%>

</BODY>

</HTML>

```

注:如果出现无法找到 OracleDriver 异常,请首先检查JDK 的目录:jdk/jre/lib/ext 中是否有 classes12.jre 文件,如果有该文件,仍出现 OracleDriver 异常,尝试将下列路径加入环境变量

```

jdk/jre/lib/ext/classes.jre

```

5.8.2 连接 MySql 数据库

可以到地址: <http://www.wordsercer.com/mm.mysql> 下载驱动程序,然后安装到某个

盘，比如 C:。

将下列路径加入环境变量

C:/mm.mysql.jdbc-2.0pre5。

与一个 Mysql 数据库建立连接有如下 2 步：

(1)加载驱动程序：

```
Class.forName("org.gjt.mm.mysql.Driver").newInstance();
```

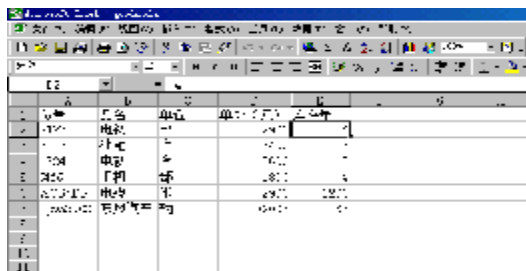
(2)建立连接：

```
Connection conn=DriverManager.getConnection( "jdbc:mysql://host:Port: 数据库名",  
用户名","密码");
```

5.9 查询 Excel 电子表格

有时需要查询 Excel 或更新删除 Excel 电子表格的内容，可以通过 JDBC-ODBC 桥接器访问 Excel 表格。访问 Excel 表格和我们访问其它的数据库有所不同，结合例子讲述如下：

假设有电子表格：goods.xls，见下图 5.30。



品名	规格	单位	单价	数量
大米	100%	袋	2.50	100
大米	100%	袋	2.50	100
大米	100%	袋	2.50	100
大米	100%	袋	2.50	100
大米	100%	袋	2.50	100
大米	100%	袋	2.50	100
大米	100%	袋	2.50	100
大米	100%	袋	2.50	100
大米	100%	袋	2.50	100
大米	100%	袋	2.50	100

图 5.30 Excel 表

(1) 设置数据源：

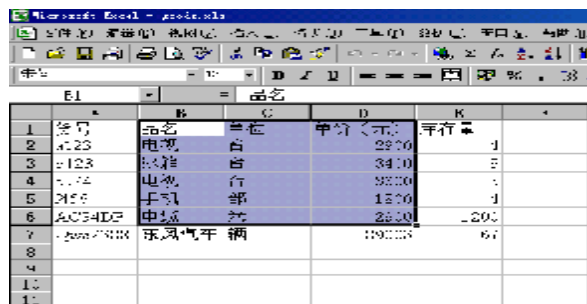
设置数据源的名字是 **star**，为数据源选择的驱动程序是：**Microsoft Excel Driver**

(2) 选择表：

与访问其它数据库不同的是，我们必须在电子表格中选出一工作区作为连接时使用的表。

在 **Excel** 电子表格中拖动鼠标选出范围，如下图 5.31 所示。然后在 **Excel** 菜单中选择 **插入→名称→定义**，给选中的工作区命名（这一工作区的名称将作为连接时使用的表名）。如图 5.32 所示。

这样我们就创建了一个名字是“品名”、有 3 个字段的表。现在就可以在 **JSP** 中查询、更新、删除这个表中的记录了。



	A	B	C	D	E	F
1	品名	品名	单位	单价(元)	库存量	
2	彩电	彩电	台	2200	1	
3	空调	空调	台	3400	5	
4	电视	电视	台	9500	1	
5	手机	手机	部	1500	1	
6	手机	手机	部	2500	1200	
7	东风汽车	东风汽车	辆	10000	50	
8						
9						
10						
11						

图 5.31 选择、创建表

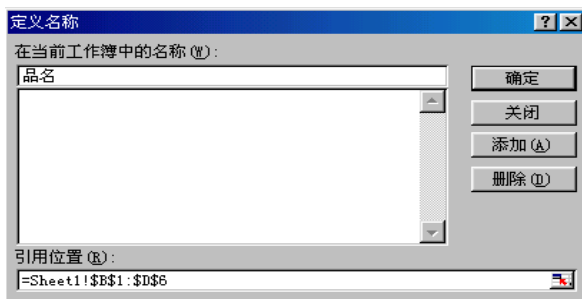


图 5.32 命名表

下面的例子 12 查询了“品名”表中的全部记录。

品名	单位	单价
电视	台	2500.00
冰箱	台	2500.00
电视	台	2500.00
手机	部	1500.00
电视	台	2500.00

图 5.33 查询 Excel 表的记录

例子 12（效果如图 5.33 所示）

Example5_12.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```

<%@ page import="java.sql.*" %>

<HTML>

<BODY>

<% Connection con;

    Statement sql;

    ResultSet rs;

    try{Class.forName('sun.jdbc.odbc.JdbcOdbcDriver');

        }

    catch(ClassNotFoundException e){}

    try { con=DriverManager.getConnection("jdbc:odbc:star","", "");

        sql=con.createStatement();

        rs=sql.executeQuery("SELECT * FROM      品名 ");

        out.print("<Table Border>");

        out.print("<TR>");

            out.print("<TH width=100>"+      品名");

            out.print("<TH width=50>"+      单位");

            out.print("<TH width=50>"+      单价");

        out.print("</TR>");

        while(rs.next())

        { out.print("<TR>");

            String name=rs.getString(1);

            out.print("<TD >"+name+"</TD>");

            String unit=rs.getString(2);

```

```

        out.print("<TD >" + unit + "</TD>");

        String unitprice = rs.getString(3);

        out.print("<TD >" + unitprice + "</TD>");

        out.print("</TR>");
    }

    out.print("</Table>");

    con.close();
}

catch(SQLException e1) {}

%>

</BODY>

</HTML>

```

5.10 使用同步连接

数据库操作中，建立连接是耗时最大的操作之一。如果客户访问的是同一数据库，那么，为每个客户都建立一个连接是不合理的。我们已经知道，在“<%!”和“%>”之间声明的变量在整个 JSP 页面内都有效，因为 JSP 引擎将 JSP 页面转译成 Java 文件时，将这些变量作为类的成员变量。这些变量的内存空间直到服务器关闭才释放。当多个客户请求一个 JSP 页面时，JSP 引擎为每个客户启动一个线程而不是启动一个进程，这些线程由 Web 服务器进程来管理，它们共享 JSP 页面的成员变量。在处理多线程问题时，可以将线程共享的变量放入一个 synchronized 块，或将修改该变量的方法用 synchronized 来修饰，这样，当一个客户用 synchronized 块或 synchronized 方法修改一个共享变量时，其它线程就必须等待，直到该线程执行完该方法或同步块。这样，我们可以把 Connection 对象作为一个成员变量被所有的客户共享，也就是说第一个访问数据库的客户负责建立连接，以后所有的客户共享

这个连接，每个客户都不要关闭这个共享的连接。

例子 13（效果如图 5.34、5.35 所示）

Example5_13.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY>

<%! // 声明一个共享的连接对象：

    Connection con=null;

%>

<% Statement sql=null;

    ResultSet rs=null;

    // 第一个客户负责建立连接对象：

    if(con==null)

    { try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        }

        catch(ClassNotFoundException e)

        {out.print(e);

        }

    try {con=DriverManager.getConnection("jdbc:odbc:moon","sa","");

        sql=con.createStatement();

        rs =sql.executeQuery("SELECT * FROM students");
```

```

        out.print("i am first");
    }
    catch(SQLException e)
    {out.print(e);
    }
}
// 其它客户通过同步块使用这个连接:
else
{ synchronized(con)
{ try { sql=con.createStatement();
        rs =sql.executeQuery("SELECT * FROM students");
        out.print("i am not first");
    }
    catch(SQLException e)
    {out.print(e);
    }
}
}
try
{ out.print("<Table Border>");
    out.print("<TR>");
    out.print("<TH width=100>"+ "学号");
    out.print("<TH width=100>"+ "姓名");

```

```

        out.print("<TH width=50>"+ "        数学成绩");
        out.print("<TH width=50>"+ "        英语成绩");
        out.print("<TH width=50>"+ "        物理成绩");
    out.print("</TR>");
    while(rs.next())
    { out.print("<TR>");

        String number=rs.getString(1);

        out.print("<TD >"+number+"</TD>");

        String name=rs.getString(2);

        out.print("<TD >"+name+"</TD>");

        int math=rs.getInt("        数学成绩");

        out.print("<TD >"+math+"</TD>");

        int english=rs.getInt("        英语成绩");

        out.print("<TD >"+english+"</TD>");

        int physics=rs.getInt("        物理成绩");

        out.print("<TD >"+physics+"</TD>");

    out.print("</TR>");
    }

    out.print("</Table>");

}

catch(SQLException e1) {}

%>

</BODY>

```

</HTML>



Figure 5.34 shows a web browser window displaying a table of student scores. The table has five columns: 学号 (Student ID), 姓名 (Name), 数学成绩 (Math Score), 英语成绩 (English Score), and 物理成绩 (Physics Score). The data is as follows:

学号	姓名	数学成绩	英语成绩	物理成绩
999001	王林	80	75	85
100002	李小明	80	80	85
100003	王下	75	65	75
100004	王小米	85	88	85
100005	王林	80	75	85

图 5.34 第一个用户负责建立连接



Figure 5.35 shows a web browser window displaying a table of student scores. The table has five columns: 学号 (Student ID), 姓名 (Name), 数学成绩 (Math Score), 英语成绩 (English Score), and 物理成绩 (Physics Score). The data is as follows:

学号	姓名	数学成绩	英语成绩	物理成绩
100001	王林	80	75	85
100002	李小明	80	80	85
100003	王下	75	65	75
100004	王小米	85	88	85
100005	王林	80	75	85

图 5.35 后继用户使用共享的连接

5.11 网上投票

我们创建一个 Access 数据库 vote.mdb，将该数据库设置为一个数据源，数据源的名字是 vote。该库含有 2 个表 “people” 和 “IP”，其结构如图 36、图 37 所示：

字段名称	数据类型	说明
name	文本	候选人姓名
count	数字	得票数

图 5.36 候选人表

字段名称	数据类型	说明
IP	文本	投票人的IP

图 5.37 候选人的 IP 地址表

people 表存放候选人的名字和得票数，IP 表存放投票人的 IP 地址。投票之前，我们要把候选人的名字和初始得票数存入 people 表中，如图 5.38 所示：



	name	count
▶	王下林	0
	林汉斌	0
	蒋焊接	0
	六小光	0
	蒋打围	0
*		0

图 5.38 录入候选人

投票系统由两个页面组成：vote.jsp 和 startvote.jsp。vote.jsp 按着 people 表中的候选人生成一个投票的表单。如图 5.39 所示：



姓名	投票 选择
王下林	<input type="radio"/>
林汉斌	<input type="radio"/>
蒋焊接	<input type="radio"/>
六小光	<input type="radio"/>
蒋打围	<input type="radio"/>

提交

图 5.39 投票选择

vote.jsp (效果如图 5.39 所示):

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY>

<% StringBuffer nameList=new StringBuffer();

    Connection con;

    Statement sql;

    ResultSet rs;

    try{ Class.forName('sun.jdbc.odbc.JdbcOdbcDriver');

        }

    catch(ClassNotFoundException e){}

    try{ con=DriverManager.getConnection("jdbc:odbc:vote","", "");

        sql=con.createStatement();

        rs=sql.executeQuery("SELECT * FROM people");

        nameList.append("<FORM action=startvote.jsp Method=post>");

        nameList.append("<Table Border>");

        nameList.append("<Table Border>");

        nameList.append("<TR>");

        nameList.append("<TH width=100>"+ "        姓名");

        nameList.append("<TH width=50>"+ "        投票选择");

        nameList.append("</TR>");
```

```

while(rs.next())

{ nameList.append("<TR>");

String name=rs.getString(1);

nameList.append("<TD >"+name+"</TD>");

String s="<Input type=radio name=name value="+name+" >";

nameList.append("<TD >"+s+"</TD>");

nameList.append("</TR>" );

}

nameList.append("</Table>");

nameList.append("<Input Type=submit value=      提交>");

nameList.append("</FORM ");

con.close();

out.print(nameList);

}

catch(SQLException e1) {}

%>

</BODY>

</HTML>

```

startvote.jsp 页面获取 **vote.jsp** 页面提交的候选人的名字。该页面在进行投票之前，首先查询 **IP** 表，判断该用户的 **IP** 地址是否已经投过票，如果该 **IP** 地址没有投过票，就可以参加投票了，投票之后，将投票用户的 **IP** 写入数据库的 **IP** 表中；如果该 **IP** 地址已经投过票，将不允许再投票。我们通过 **IP** 地址来防止一台计算机反复的投票，但不能有效地限制拨号上网的用户，因为拨号上网的用户的 **IP** 是动态分配的，用户可以重新拨号上

网获得一个新的 IP 地址。



图 5.40 投票成功

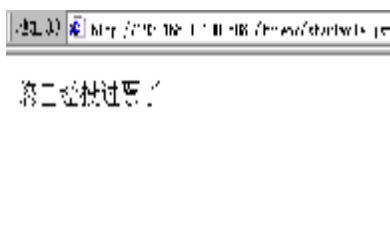


图 5.41 重复投票

startvote.jsp（效果如图 5.40、5.41 所示）：

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="java.io.*" %>

<html>

<body>

<%! //记录总票数的变量:

    int total=0;

    // 操作总票数的同步方法:

    synchronized void countTotal()

    { total++;

    }
```

%>

<% boolean vote=true; //决定用户是否有权投票的变量。

// 得到被选择的候选人名字:

String name="";

name=request.getParameter("name");

if(name==null)

{name="?";

}

byte a[]=name.getBytes("ISO-8859-1");

name =new String(a);

// 得到投票人的 IP 地址:

String IP=(String)request.getRemoteAddr();

// 加载桥接器:

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

}

catch(ClassNotFoundException e){}

Connection con=null;

Statement sql=null;

ResultSet rs=null;

// 首先查询 IP 表, 判断该用户的 IP 地址是否已经投过票:

try { con=DriverManager.getConnection("jdbc:odbc:vote","", "");

sql=con.createStatement();

rs=sql.executeQuery("SELECT * FROM IP WHERE IP = '"+IP+"'");

```

    int row=0;

    while(rs.next())

        { row++;

        }

    if(row>=1)

        { vote=false;    //          不允许投票。

        }

    }

    catch(SQLException e)

        {}

    if(name.equals("?"))

        { out.print("  您没有投票,没有权利看选举结果");

        }

    else

    {

        if(vote)

        { out.print("    您投了一票");

            //      将总票数加 1:

            countTotal();

            //      通过连接数据库, 给该候选人增加一票,

            //      同时将自己的 IP 地址写入数据库 。

            try

            { rs=sql.executeQuery("SELECT * FROM people WHERE name =

```

```

"+'''+name+''');

    rs.next();

    int count=rs.getInt("count");

    count++;

    String condition=

        "UPDATE people SET count = "+count+" WHERE

name="+'''+name+'''' ;

    //      执行更新操作(投票计数):

    sql.executeUpdate(condition);

    //      将 IP 地址写入 IP 表:

    String to=

        "INSERT INTO IP VALUES"+"("+'''+IP+'''+")";

    sql.executeUpdate(to);

}

catch(SQLException e)

{ out.print(""+e);

}

//      显示投票后的表中的记录:

try{ rs=sql.executeQuery("SELECT * FROM people");

    out.print("<Table Border>");

    out.print("<TR>");

    out.print("<TH width=100>"+"        姓名");

    out.print("<TH width=50>"+"        得票数");

```

```

        out.print("<TH width=50>"+ "        总票数:"+total);

        out.print("</TR>");

while(rs.next())

    { out.print("<TR>");

        out.print("<TD >"+rs.getString(1)+"</TD>");

        int count=rs.getInt("count");

        out.print("<TD >"+count+"</TD>");

        double b=(count*100)/total; //        得票的百分比。

        out.print("<TD >"+b+"%"+ "</TD>");

        out.print("</TR>");

    }

    out.print("</Table>");

    con.close();

}

catch(SQLException e)

    {}

}

else

    {out.print(" 您已经投过票了");

    }

}

%>

</BODY>

```

</HTML>

5.12 成绩录入查询系统

我们创建一个 Access 数据库 student.mdb，将该数据库设置为一个数据源，数据源的名字是 manage。该库含有 3 个表，“基本信息”、“成绩”和“verify”表，表结构如下：

“基本信息”表（学号作关键字）

字段名称	数据类型	主关键字
学号	文本	√
姓名	文本	—
专业	文本	—
年级	文本	—

“成绩”表（学号字段和课程名称字段组合作关键字）

字段名称	数据类型	主关键字
学号	文本	√
课程名称	文本	
成绩	文本	—

“基本信息”表和“成绩”表建立如图 5.42 所示关联关系。

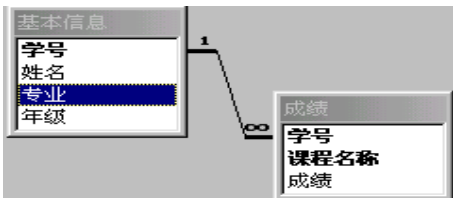


图 5.42 表之间的关联关系

□

verify 表存放了成绩录入人员的帐号和密码，用来验证成绩录入人员的身份，如图 5.43 所示。

verify: 表			
	帐号	姓名	密码
▶	apple	张林林	1999
	babana	李林林	2000
	orange	王林林	2001
*			

图 5.43 帐号密码表

该系统分如下几个模块：

I 主页面：

main.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<HTML>
```

```
<BODY BGcolor=yellow>
```

```
<CENTER>
```

```
<FONT SIZE=2>
```

<P>成绩录入查询系统

录入人员登录

录入学生的基本信息

录入学生成绩

修改密码

查询成绩

</CENTER>

</BODY>

</HTML>

- I 登录页面：具有特殊权限的人，有成绩录入的权利。帐号和密码是事先存放在一个数据库的 **verify** 表中，成绩录入人员必须通过这个页面进行身份验证。
- I 基本信息输入页面：该页面负责录入学生的基本信息。该页面有防止客户绕开登录的功能，即没有登录的客户不能直接进入该页面。如果试图通过 **URL** 直接进入该页面，会被引导至登录页面。
- I 成绩录入页面：该页面负责录入成绩。该页面有防止客户绕开登录的功能，即没有登录的客户不能直接进入该页面，如果试图通过 **URL** 直接进入该页面，会被引导至登录页面。在该页面如果输入了无基本信息的学生的成绩，系统会提示到基本信息页面输入学生的基本信息。
- I 成绩查询页面：通过学号可以查询该学号的全部考试成绩。
- I 修改密码页面：确认身份后，可修改密码。

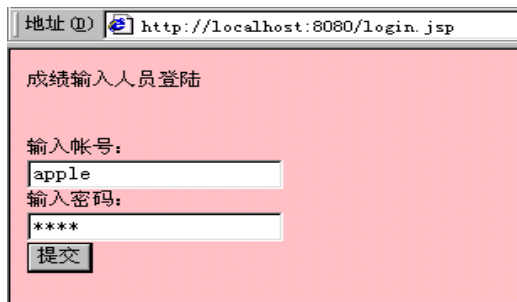


图 5.44 登录页面

登录页面（效果如图 5.44 所示）

login.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import="java.sql.*" %>
```

```
<HTML>
```

```
<BODY bgcolor=pink ><Font size=1>
```

```
<P> 成绩输入人员登录
```

```
<FORM action="login.jsp" Method="post">
```

```
<BR>输入帐号:
```

```
<BR><Input type=text name="account">
```

```
<BR>输入密码:
```

```
<BR><Input type=password name="secret">
```

```
<BR><Input type=submit name="g" value="提交">
```

</FORM>

<% // 获取提交的帐号:

String account=request.getParameter("account");

if(account==null)

{account="";

}

byte b[]=account.getBytes("ISO-8859-1");

account=new String(b);

// 获取提交的密码:

String secret=request.getParameter("secret");

if(secret==null)

{secret="";

}

byte c[]=secret.getBytes("ISO-8859-1");

secret=new String(c);

// 查询数据库信息，验证身份:

Connection con=null;

Statement sql=null;

ResultSet rs=null;

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

}

catch(ClassNotFoundException event){}

try

```

{con=DriverManager.getConnection("jdbc:odbc:manage","", "");
sql=con.createStatement();

String condition="SELECT * FROM verify WHERE          帐 号  =
"+"\""+account+"\"";

rs =sql.executeQuery(condition);

while(rs.next())

    { String n=rs.getString("          帐号");

      String s=rs.getString("          密码");

      if(account.equals(n)&&secret.equals(s))

          { //          将帐号和密码存入 session 对象，以 备在输入界面：
inputMessage.jsp 中使用：

              session.setAttribute("account",account);

              session.setAttribute("secret",secret);

              //          为了防止客户浏览器限制了 Cooker 的使用，需对连接进行重定
向处理：

String s1=response.encodeRedirectURL("inputMessage.jsp");

String s2=response.encodeRedirectURL("inputResult.jsp");

              //          连接到基本信息输入页面：

              response.sendRedirect(s1);

          }

      }

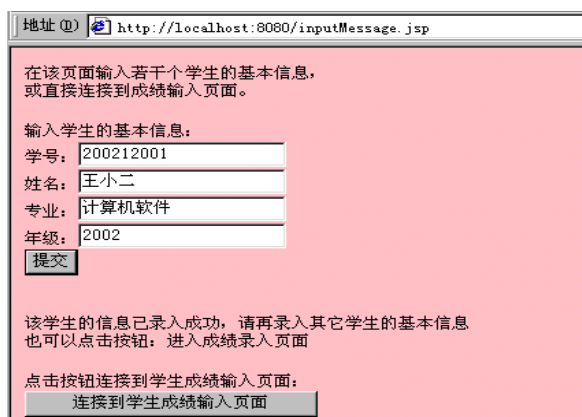
    }

if(!(session.isNew()))

    { out.print("<BR>          您输入的帐号或密码不正确"+account+": "+secret);

```

```
    }  
}  
  
catch(SQLException e1) {}  
  
%>  
  
<BR><A HREF="main.jsp">返回主页</A>  
  
</FONT>  
  
</BODY>  
  
</HTML>
```



The screenshot shows a web browser window with the address bar displaying "http://localhost:8080/inputMessage.jsp". The page content is as follows:

在该页面输入若干个学生的基本信息，
或直接连接到成绩输入页面。

输入学生的基本信息：

学号：	200212001
姓名：	王小二
专业：	计算机软件
年级：	2002

该学生的信息已录入成功，请再录入其它学生的基本信息
也可以点击按钮：进入成绩录入页面

点击按钮连接到学生成绩输入页面：

图 5.45 录入基本信息

基本信息输入页面（效果如图 5.45 所示）

inputMessage.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY bgcolor=pink ><Font size=1>

<P>在该页面输入若干个学生的基本信息，<BR>或直接连接到成绩输入页面。

<%!// 处理字符串的一个常用方法:

    public String getString(String s)

    { if(s==null) s="";

      try {byte a[]=s.getBytes("ISO-8859-1");

          s=new String(a);

        }

      catch(Exception e)

      {

      }

      return s;

    }

%>

<% //为了防止客户浏览器限制了 Cooker 的使用，需对连接进行重定向处理:

    String s1=response.encodeRedirectURL("inputMessage.jsp");

    String s2=response.encodeRedirectURL("inputResult.jsp");

%>
```

<% //为了防止客户直接进入该页面，首先从 session 对象获取密码和帐号信息：

```
String account="",secret="";
```

```
if(session.isNew())
```

```
{//    如果直接进入该页面就再连接到登录页面：
```

```
    response.sendRedirect("login.jsp");
```

```
}
```

```
else
```

```
{ account=(String)session.getAttribute("account");
```

```
    secret=(String)session.getAttribute("secret");
```

```
    secret=getString(secret);
```

```
    account=getString(account);
```

```
}
```

```
// 连接到数据库验证帐号和密码：
```

```
Connection con=null;
```

```
Statement sql=null;
```

```
ResultSet rs=null;
```

```
boolean boo=false;
```

```
try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
}
```

```
catch(ClassNotFoundException event){}
```

```
try
```

```
{con=DriverManager.getConnection("jdbc:odbc:manage","", "");
```

```
    sql=con.createStatement();
```


String condition="SELECT * FROM verify WHERE

帐号 =

```
"+"+"'"+account+"";
```

```
rs =sql.executeQuery(condition);
```

```
while(rs.next())
```

```
{ String n=rs.getString("    帐号");
```

```
String s=rs.getString("    密码");
```

```
if(account.equals(n)&&secret.equals(s))
```

```
{boo=true; break;
```

```
}
```

```
}
```

```
}
```

```
catch(SQLException e1) {}
```

```
// 如果帐号密码正确，就显示输入学生基本信息的表单界面：
```

```
if(boo)
```

```
{out.print("<FORM action= "+s1+" method=post>");
```

```
out.print("<P>    输入学生的基本信息： ");
```

```
out.print("<BR>    学号： ");
```

```
out.print("<Input type=text name=number1>");
```

```
out.print("<BR>    姓名： ");
```

```
out.print("<Input type=text name=name>");
```

```
out.print("<BR>    专业： ");
```

```
out.print("<Input type=text name=zhuanye>");
```

```
out.print("<BR>    年级： ");
```

```

    out.print("<Input type=text name=grade>");

    out.print("<BR> <Input type=submit value=    提交>");

    out.print("</FORM>");

}

else

    {response.sendRedirect('login.jsp');

    }

%>
<% //获取基本信息存入数据库中的“基本信息”表中:

String number1=request.getParameter("number1"),

    name  =request.getParameter("name"),

    zhuanYe=request.getParameter("zhuanYe"),

    grade =request.getParameter("grade");

if(number1==null)

    {number1="?????????";

    }

number1=getString(number1);

number1=number1.trim();

name  =getString(name);

zhuanYe=getString(zhuanYe);

grade =getString(grade);

String basicmessage=

"INSERT INTO    基本信息 VALUES"+"("+"'"+number1+"','"+name+"',' "+

```

```

        zhuanYe+" ',' "+grade+" ' '+")";

if(!(number1.startsWith("?")))

{ //    首先查找基本信息表中是否已存在该学生的信息：

    rs=sql.executeQuery("SELECT * FROM          基本信息 WHERE 学号 ="+"
"+number1+" ' ');

    //    如果该学号的学生已经存在，就转入成绩输入页面：

    boolean line=rs.next();

    //    通过 line 判断结果集是否有记录

    if(line==true)

    { out.print("          该考号已经存在，请再录入其它学生的基本信息");

      out.print("<BR>          也可以点击按钮：进入成绩录入页面");

      //          显示该生的基本信息：

      out.print("<BR>          学号"+rs.getString(1));

      out.print("<BR>          姓名"+rs.getString(2));

      out.print("<BR>          专业"+rs.getString(3));

      out.print("<BR>          年级"+rs.getString(4));

    }

    //    如果该学号的学生不存在，就将信息写入基本信息表，再转入成绩输入
页面：

    else

    {sql.executeUpdate(basicmessage);

      out.print("<BR>          该学生的信息已录入成功，请再录入其它学生的基本
信息");

```

```

        out.print("<BR>                也可以点击按钮：进入成绩录入页面");

    }

}

else

    {out.print("        必须输入学号，学号不可以用?开头");

    }

con.close();

%>

<FORM action=<%=s2%>>

<P>点击按钮连接到学生成绩输入页面：

<BR><Input type="submit" value="连接到学生成绩输入页面">

<BR><A HREF="main.jsp">返回主页</A>

</BODY>

</HTML>

```

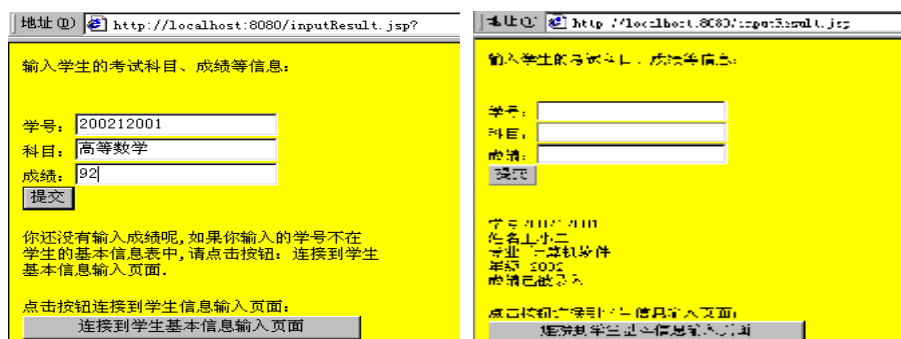


图 5.46 成绩录入与提交

成绩录入页面（效果如图 5.46 所示）

inputResult.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY bgcolor=yellow ><Font size=1>

<%!// 处理字符串的一个常用方法:

    public String getString(String s)

    { if(s==null) s="";

      try {byte a[]=s.getBytes("ISO-8859-1");

          s=new String(a);

      }

      catch(Exception e)

      { }

      return s;

    }

%>

<% String s1=response.encodeRedirectURL("inputMessage.jsp");

    String s2=response.encodeRedirectURL("inputResult.jsp");

%>
```

<% //为了防止客户直接进入该页面，首先从 **session** 对象获取密码和帐号信息：

```
String account="",secret="";
```

```
if(session.isNew())
```

```
    {//    如果直接进入该页面就连接到登录页面：
```

```
        response.sendRedirect("login.jsp");
```

```
    }
```

```
else
```

```
    { account=(String)session.getAttribute("account");
```

```
        secret=(String)session.getAttribute("secret");
```

```
        secret=getString(secret);
```

```
        account=getString(account);
```

```
    }
```

//连接到数据库验证帐号和密码：

```
Connection con=null;
```

```
Statement sql=null;
```

```
ResultSet rs=null;
```

```
boolean boo=false;
```

```
try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
    }
```

```
catch(ClassNotFoundException event){}
```

```
try
```

```
    {con=DriverManager.getConnection("jdbc:odbc:manage","", "");
```

```
        sql=con.createStatement();
```

String condition="SELECT * FROM verify WHERE

帐 号 =

+"'''+account+'''';

rs =sql.executeQuery(condition);

while(rs.next())

{ String n=rs.getString(" 帐号");

String s=rs.getString(" 密码");

if(account.equals(n)&&secret.equals(s))

{boo=true; break;

}

}

}

catch(SQLException e1) {}

// 如果帐号密码正确，就显示输入成绩的表单界面：

if(boo)

{out.print("<P> 输入学生的考试科目、成绩等信息： ");

out.print("<FORM action="+s2+" method=post>");

**out.print("
 学号： ");**

out.print("<Input type=text name=number2>");

**out.print("
 科目： ");**

out.print("<Input type=text name=subject>");

**out.print("
 成绩： ");**

out.print("<Input type=text name=result>");

**out.print("
 <Input type=submit value= 提交>");**

```

        out.print("</FORM>");
    }
else
    {response.sendRedirect("login.jsp");
    }
%>
<%    String number2=request.getParameter("number2"),
        subject=request.getParameter("subject"),
        result=request.getParameter("result");
        if(number2==null)
            {number2="#####";
            }
        number2=getString(number2);
        subject=getString(subject);
        result=getString(result);
        number2=number2.trim();
//    从“基本信息”表中查找学号是 number2 的记录:
        String basicmessage=
            "SELECT * FROM      基本信息 WHERE 学号 = '"+number2+"'";
try { sql=con.createStatement();
        rs=sql.executeQuery(basicmessage);
        boolean line=rs.next();
        //        通过 line 判断结果集是否有记录,

```



```

//      如果没有该学生的基本信息就返回提示:

if(line==false)

    {out.print("      你还没有输入成绩呢,");

      out.print("      如果你输入的学号不在<BR>学生的基本信息表中,");

      out.print("      请点击按钮: 连接到学生<BR>基本信息输入页面.");

    }

else

    { //      显示该生的基本信息:

      out.print("<BR>      学号"+rs.getString(1)) ;

      out.print("<BR>      姓名"+rs.getString(2)) ;

      out.print("<BR>      专业"+rs.getString(3)) ;

      out.print("<BR>      年级"+rs.getString(4)) ;

      String resultmessage=

        "INSERT INTO      成      绩

VALUES"+"("+"'"+number2+"','"+subject+" ','"+result+" ' "+')";

      sql.executeUpdate(resultmessage);

      out.print("<BR>      成绩已被录入");

    }

}

catch(SQLException e)

    {out.print("<BR>" +subject+"      该课程的成绩已经存在<BR>不允许重复录入

");

    }

```

```

con.close();

%>

<FORM action=<%=s1%>>

<P>点击按钮连接到学生信息输入页面:

<BR><Input type="submit" value="连接到学生基本信息输入页面">

</FORM>

</BODY>

</HTML>

```

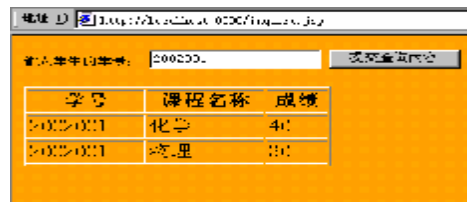


图 5.47 成绩查询

成绩查询页面（效果如图 5.47 所示）

inquire.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<HTML>

<BODY bgcolor=orange ><Font size=1>

<FORM action="" Method=post>

    输入学生的学号:

    <Input type=text name=number>

```

<Input type=submit name=g vale= 提交>

</FORM>

<%! Connection con=null; // 声明一个共享的连接对象

%>

<% // 获取学号:

String studentNumber=request.getParameter("number");

if(studentNumber==null)

{studentNumber="?";

}

byte b[]=studentNumber.getBytes("ISO-8859-1");

studentNumber=new String(b);

Statement sql=null;

ResultSet rs=null;

if(con==null)

{ try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

}

catch(ClassNotFoundException e)

{out.print(e);

}

try {con=DriverManager.getConnection("jdbc:odbc:manage","", "");

String condition="SELECT * FROM 成绩 WHERE 学号 =

"+"'+studentNumber+"';

rs =sql.executeQuery(condition);

```

    }

    catch(SQLException e)

    {out.print(e);

    }

}

else

{ synchronized(con)

{ try { sql=con.createStatement();

String condition="SELECT * FROM          成绩 WHERE 学号 =

"+""+studentNumber+"";

rs =sql.executeQuery(condition);

}

catch(SQLException e)

{out.print(e);

}

}

}

try { out.print("<Table Border>");

out.print("<TR>");

out.print("<TH width=100>"+          学号");

out.print("<TH width=100>"+          课程名称");

out.print("<TH width=50>"+          成绩");

out.print("</TR>");

```

```

while(rs.next())

{ out.print("<TR>");

    String number=rs.getString(1);

    out.print("<TD >" +number+"</TD>");

    String subjectName=rs.getString(2);

    out.print("<TD >" +subjectName+"</TD>");

    String chengji=rs.getString("        成绩");

    out.print("<TD >" +chengji+"</TD>");

    out.print("</TR>") ;

}

out.print("</Table>");

}

catch(SQLException e1) {}

%>

<BR><A HREF="main.jsp">返回主页</A>

</FONT>

</BODY>

</HTML>

```

修改密码页面（效果如图 5.48 所示）

modifySecret.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

```

<HTML>

<BODY bgcolor=pink >

<P> 修改密码,密码长度不能超过 30 个字符:

<FORM action="" Method="post"

**
**输入您的帐号:

**
<Input type=text name="acco**

**
**输入您的密码:

**
<Input type=password name="secret">**

**
**输入您的新的密码:

**
<Input type=text name="newSecret1">**

**
**请再输入一次新密码:

**
<Input type=text name="newSecret2">**

**
<Input type=submit name="g" value="提交">**

</FORM>

<%!//处理字符串的一个常用方法:

```
public String getString(String s)
{
    if(s==null) s="";
    try {byte a[]=s.getBytes("ISO-8859-1");
        s=new String(a);
    }
    catch(Exception e)
    {
    }
    return s;
}
```

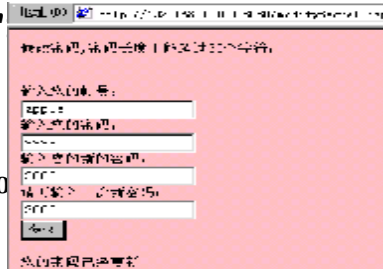


图 5.48 修改密码

```

    }
%>
<% // 获取提交的帐号:

    String account=request.getParameter("account");

    account=getString(account);

    // 获取提交的密码:

    String secret=request.getParameter("secret");

    secret=getString(secret);

    // 获取提交的新密码:

    String newSecret1=request.getParameter("newSecret1");

    newSecret1=getString(newSecret1);

    // 获取提交的新密码:

    String newSecret2=request.getParameter("newSecret2");

    newSecret2=getString(newSecret2);

    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

    catch(ClassNotFoundException event){}

    // 查询数据库信息, 验证身份:

    Connection con=null;

    Statement sql=null;

    ResultSet rs=null;

    boolean modify=false;

    boolean ifEquals=false;

```

```

ifEquals=(newSecret1.equals(newSecret2))&&(newSecret1.length()<=30);

if(ifEquals==true)

    {try

        { con=DriverManager.getConnection("jdbc:odbc:manage","", "");

            sql=con.createStatement();

            String condition="SELECT * FROM verify WHERE          帐号 =
            "+"""+account+""";

            rs =sql.executeQuery(condition);

            while(rs.next())

                { String n=rs.getString("          帐号");

                    String s=rs.getString("          密码");

                    if(account.equals(n)&&secret.equals(s))

                        { //          修改密码:

                            modify=true;

                            out.print("          您的密码已经更新");

                            String c=

                                "UPDATE verify SET          密码 = "+"""+newSecret1+"""+ " WHERE          帐号 =
                                "+""""+account+""";

                                rs =sql.executeQuery(c);

                                }

                                }

                            }

```



```

        catch(SQLException e1) {}

    }

    else

        { out.print("    你两次输入的密码不一致或长度过大");

        }

    if(modify==false&&ifEquals==true)

        { out.print("<BR>    您没有输入密码帐号或<BR>您输入的帐号或密码不正确

"+account+": "+secret);

        }

    %>

</FONT>

</BODY>

</HTML>

```

第6章 JSP 与 JavaBeans

在谈论组件之前让我们看一个通俗的事情：组装电视机。组装一台电视机时，人们可以选择多个组件，例如电阻、电容、显象管等，一个组装电视机的人不必关心显象管是怎么研制的，只要根据说明书了解其中的属性和功能就可以了。不同的电视机可以安装相同的显象管，显象管的功能完全相同，但他们是在不同的电视机里面，一台电视机的显象管发生了故障并不影响其它的电视机；也可能两台电视安装了一个共享的组件：天线，如果天线发生了故障，两台电视机都受到同样的影响。

“可视化组件编程”非常成功的一个例子就是微软公司的 VB。人们在使用 VB 编写程序时，经常把一个按钮组件或文本框组件拖放到你的应用程序窗体中，并了解这个按钮的名字、它有哪些功能、方法等，而且你还可以重新更改它的名字，当你创建生成应用程序时，这个按钮的名字被保存了下来。但是，微软的组件只适用于微软的操作平台上，不能为其它的平台所使用。

按着 Sun 公司的定义，JavaBeans 是一个可重复使用的软件组件。实际上 JavaBeans 是一种 Java 类，通过封装属性和方法成为具有某种功能或者处理某个业务的对象，简称 beans。由于 javabeans 是基于 java 语言的，因此 javabeans 不依赖平台，具有以下特点：

1. 可以实现代码的重复利用
2. 易编写、易维护、易使用
3. 可以在任何安装了 Java 运行环境的平台上的使用，而不需要重新编译。

我们已经知道，一个基本的 JSP 页面就是由普通的 HTML 标签和 java 程序片组成，如果程序片和 HTML 大量交互在一起，就显得页面混杂，不易维护。JSP 页面应当将数据的处理过程指派给一个或几个 beans 来完成，我们只需在 JSP 页面中调用这个 beans

即可。不提倡大量的数据处理都用 **java** 程序片来完成。在 **JSP** 页面中调用 **beans**，可有效的分离的静态工作部分和动态工作部分。

6.1 编写 **javabeans** 和使用 **javabeans**

6.1.1 编写 **beans**

Javabeans 分为可视组件和非可视组件。在 **JSP** 中主要使用非可视组件。对于非可视组件，我们不必去设计它的外观，主要关心它的属性和方法。

编写 **javabeans** 就是编写一个 **java** 的类，所以你只要会写类就能编写一个 **beans**，这个类创建的一个对象称做一个 **beans**。为了能让使用这个 **beans** 的应用程序构建工具（比如 **JSP** 引擎）知道这个 **beans** 的属性和方法，只需在类的方法命名上遵守以下规则：

1. 如果类的成员变量的名字是 **xxx**，那么为了更改或获取成员变量的值，即更改或获取属性，在类中就需要有两个方法：
getXxx()：用来获取属性 **xxx**。
setXxx()：用来修改属性 **xxx**。。
2. 对于 **boolean** 类型的成员变量，即布尔逻辑类型的属性，允许使用“**is**”代替上面的“**get**”和“**set**”。
3. 类中的普通方法不适合上面的命名规则，但这个方法必须是 **public** 的。
4. 类中如果有构造方法，那么这个构造方法也是 **public** 的并且是无参数的。

下面我们编写一个简单的 **beans**，并说明在 **JSP** 中怎样使用这个 **beans**。

Circle.java:

```
import java.io.*;

public class Circle

{ int radius;
```

```

public Circle()
{
    radius=1;
}

public int getRadius()
{
    return radius;
}

public void setRadius(int newRadius)
{
    radius=newRadius;
}

public double circleArea()
{
    return Math.PI*radius*radius;
}

public double circlLength()
{
    return 2.0*Math.PI*radius;
}
}

```

将上述 java 文件保存为 **Circle.java**，并编译通过，得到字节码文件 **Circle.class**。

6.1.2 使用 beans

为了在 JSP 页面中使用 beans，我们必须使用 JSP 动作标签：**useBean**

useBean 格式：

```

<jsp:useBean id= “给 bean 起的名字” class= “创建 beans 的类” scope= “bean 有效范围”>

```

`</jsp:useBean>`

或

`<jsp:useBean id= “给 bean 起的名字” class= “创建 beans 的类” scope= “bean 有效范围”/>`

当服务器上某个含有 **useBean** 动作标签的 JSP 页面被加载执行时，JSP 引擎将首先根据 **id** 的名字，在一个同步块中，查找 JSP 引擎内置 **pageContent** 对象中是否含有名字 **id** 和作用域 **scope** 的对象，如果这个对象存在，JSP 引擎就分配一个这样的对象给客户，这样，客户就获得了一个作用域是 **scope**、名字是 **id** 的 **beans**（就像我们组装电视机时获得了一个有一定功能和使用范围的电子元件）。如果在 **pageContent** 中没有查找到指定作用域、名字是 **id** 的对象，就根据 **class** 指定的类创建一个名字是 **id** 对象，即创建了一个名字是 **id** 的 **beans**，并添加到 **pageContent** 内置对象中，并指定该 **beans** 的作用域是 **scope**，同时 JSP 引擎分配给客户一个作用域是 **scope**、名字是 **id** 的 **beans**。

下面就 **useBean** 标签中 **scope** 取值的不同情况阐述如下：

2 scope 取值 page

JSP 引擎分配给每个客户的 **beans** 是互不相同的，也就是说，尽管每个客户的 **beans** 的功能相同，但它们占有不同的内存空间。该 **beans** 的有效范围是当前页面，当客户离开这个页面时，JSP 引擎取消分配给该客户的 **beans**。

2 scope 取值 session

JSP 引擎分配给每个客户的 **beans** 是互不相同的，该 **beans** 的有效范围是客户的会话期间，也就是说，如果客户在多个页面中相互连接，每个页面都含有一个 **useBeans** 标签，

这些 `useBean` 标签中 `id` 的值相同，并且 `scope` 的值都是 `session`，那么，该客户在这些页面得到的 `beans` 是相同的一个。如果客户在某个页面更改了这个 `beans` 的属性，其它页面的这个 `beans` 的属性也将发生同样的变化。当客户关闭浏览器时，JSP 引擎取消分配给客户的 `beans`。

2 scope 取值 request

JSP 引擎分配给每个客户的 `beans` 是互不相同的，该 `beans` 的有效范围是 `request` 期间。客户在网站的访问期间可能请求过多个页面，如果这些页面含有 `scope` 取值是 `request` 的 `useBeans` 标签，那么 `pageContent` 对象在每个页面分配给客户的 `beans` 也是互不相同的。JSP 引擎对请求作出响应之后，取消分配给客户的这个 `beans`。

2 scope 取值 application

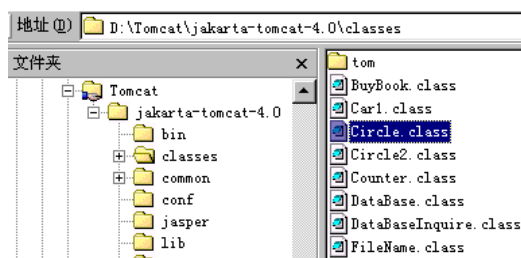
JSP 引擎为每个客户分配一个共享的 `beans`，也就是说，所有客户共享这个 `beans`，如果一个客户改变这个 `beans` 的某个属性的值，那么所有客户的这个 `beans` 的属性值都发生了变化。这个 `beans` 直到服务器关闭才被取消。

注：当使用作用域是 `session` 的 `beans` 时，要保证客户端支持 `Cookie`。

为了使服务器的所有 `web` 服务目录下的 JSP 页面文件都能使用我们的 `beans`，我们必须将上面编译通过生成的字节码类文件：`Circle.class` 拷贝到 JSP 引擎的 `classes` 文件夹下，即 `D:\tomcat\Jakarta-tomcat-4.0\classes` 下，如图 6.1 所示。

另外，在使用 `beans` 的 JSP 页面中，必须有如下的 `import` 指令：

```
<@page import= "Circle">
```



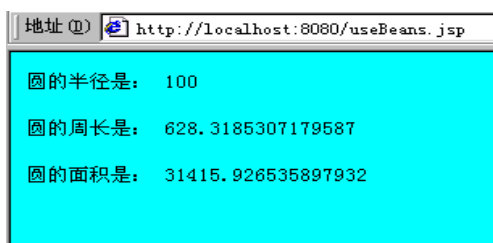


图 6.2 使用 beans 的 JSP 页面

在下面的例子 1 中，负责创建 **beans** 的类是上述的 **Circle** 类，创建的 **beans** 的名字是 **girl**，**girl** 的 **scope** 取值 **page**。

例子 1(效果如图 6.2 所示)

useBeans.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import="Circle"%>

<HTML>

<BODY bgcolor=cyan><Font size=1>

    <jsp:useBean id="girl" class="Circle" scope="page" >

    </jsp:useBean>

    <%-- 通过上述 JSP 标签, 客户获得了一个作用域是 page, 名字是 girl 的 beans --%>

    <% // 设置圆的半径:

        girl.setRadius(100);

    %>

    <P>圆的半径是:

        <%=girl.getRadius()%>

    <P>圆的周长是:

        <%=girl.circlLength()%>

    <P>圆的面积是:

        <%=girl.circleArea()%>

</BODY>

</HTML>
```

在下面的例子 2 中我们将 beans 的 scope 的值设为 session

创建的 beans 的名字是 girl, 创建该 beans 的类文件仍然是上述的 Circle.class。在 beans1.jsp 页面中, girl 的半径 radius 的值是 1 (图 6.3), 然后连接到 beans2.jsp 页面, 显示半径 radius 的值, 然后将 girl 的半径 radius 的值更改为 400 (图 6.4), 当再刷新 beans1.jsp 时会发现 radius 的值已经变成了 400 (图 6.5)。

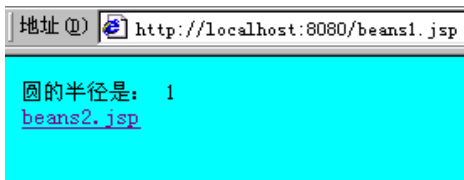


图 6.3 beans1.jsp 效果



图 6.4 beans 2.jsp 效果

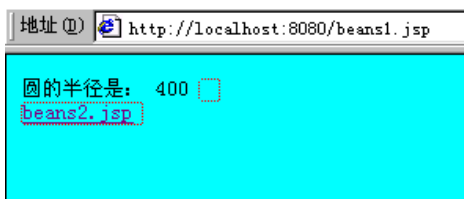


图 6.5 刷新 beans1.jsp 后效果

例子 2(效果如图 6.3、6.4、6.5 所示)

beans1.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```

<%@ page import="Circle"%>

<HTML>

<BODY bgcolor=cyan><Font size=1>

    <jsp:useBean id="girl" class="Circle" scope="session" >

    </jsp:useBean>

<P>圆的半径是:

    <%=girl.getRadius()%>

    <A href="beans2.jsp"><BR>beans2.jsp </A>

</BODY>

</HTML>

```

beans2.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="Circle"%>

<HTML>

<BODY bgcolor=cyan><Font size=1>

    <jsp:useBean id="girl" class="Circle" scope="session" >

    </jsp:useBean>

<P>圆的半径是:

    <%=girl.getRadius()%>

    <%=girl.setRadius(400);%>

<P>修改后的圆的半径是:

    <%=girl.getRadius()%>

```

</BODY>

</HTML>

在下面的例子 3 中，将 beans 的 scope 的值设为 application。当第一个客户访问这个页面时，显示 beans 的属性 radius 的值，然后把这个属性的值修改为 1000（图 6.6）。当其它客户访问这个网页时，看到的这个属性的值都是 1000（图 6.7）。

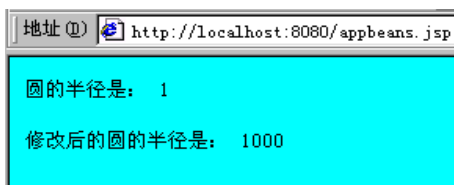


图 6.6 第一个客户访问 appbeans.jsp 效果

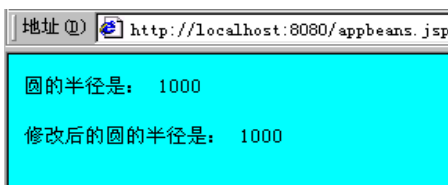


图 6.7 后续客户访问 appbeans.jsp 的效果

例子 3(效果如图 6.6、6.7 所示)

appbeans.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="Circle"%>

<HTML> <BODY>

<jsp:useBean id="girl" class="Circle" scope="application" >

    </jsp:useBean>

<P>圆的初始半径是:

    <%=girl.getRadius()%>
```

```
<%girl.setRadius(1000);%>

<P>修改后的圆的半径是:

<%=girl.getRadius()%>

</BODY>

</HTML>
```

6.2 beans 的存放目录

(1) 任何 web 服务目录都可使用的 beans 的存放位置

如果让任何 web 服务目录中的 JSP 页面都可以使用某个 beans, 那么创建这个 beans 的字节码文件需存放在 Tomcat 安装目录的 classes 目录中, 例如, 本书所用机器的 Tomcat 安装目录就是: **D:\tomcat\Jakarta-tomcat-4.0\classes**, 如图 6.1 所示。

我们已经知道, 当服务器上某个含有 useBean 动作标签的 JSP 页面被加载执行时, JSP 引擎将首先根据 id 的名字, 在一个同步块中, 查找 JSP 引擎内置 pageContent 对象中是否含有名字 id 和作用域 scope 的对象, 如果这个对象存在, JSP 引擎就分配一个这样的对象给客户, 这样, 客户就获得了一个作用域是 scope、名字是 id 的 beans。如果在 pageContent 中没有查找到指定作用域、名字是 id 的对象, 就根据 class 指定的类创建一个名字是 id 的对象, 并添加到 pageContent 内置对象中, 并指定该对象的作用域是 scope, 同时 JSP 引擎分配给客户一个作用域是 scope、名字是 id 的 beans。

JSP 引擎的内置 PageContentet 对象用来存储供服务器使用的数据信息, 通过该对象向客户提供不同类型的各种数据对象。当含有 useBeans 标签的 JSP 页面被执行后, beans 就被存放在 pageContentet 对象中, 如果你更改了创建 beans 的 java 类文件后, pageContent 对象中的 beans 并不能被更新, 这是因为任何 JSP 页面再次被访问执行时, 总是先到 pageContentet 中查找 beans。而 pageContent 对象直到服务器关闭才释放它存储的数据对象。

(2) 只对 **examples** 服务目录可用的 **beans** 的存放目录

examples 是默认 web 服务目录之一，如图 6.8

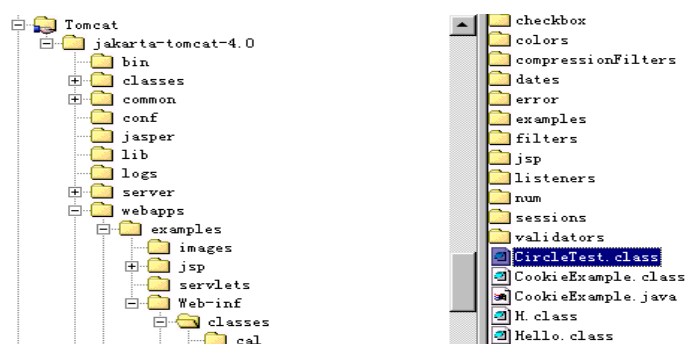


图 6.8 存放 beans 的目录

如果想让某个 **beans** 只对 **examples** 目录下的 **JSP** 页面可用，那么创建该 **beans** 的字节码文件需存放在 **webapps/example/Web-inf/classes** 目录中，如图 6.8 所示。

存放在该目录中的 **beans** 和存放在上面(1)中所述的目录中的 **beans** 有所不同的是：**JSP** 引擎首先检查 **webapps/example/Web-inf/classes** 目录中的创建该 **beans** 的字节码文件是否被修改过，如果重新修改过，就会用新的字节码文件创建一个 **beans**，**beans** 的名字是 **id**，并添加到 **pageContent** 内置对象中，并指定该对象的作用域是 **scope**，同时 **JSP** 引擎分配给客户一个作用域是 **scope**、名字是 **id** 的 **beans**。

如果经常调试 **beans**，可以把 **beans** 放在 **webapps/example/Web-inf/classes**。需要注意的是，当用户请求服务时，由于服务器引擎每次都要检查字节码文件是否被修改过，

将降低服务器的运行效率。

下面是创建 beans 的 java 文件，将该文件编译通过，把字节码文件 CircleTest.class 存放到/webapps/example/Web-inf/classes 目录中。

```
import java.io.*;

public class CircleTest

{ int radius;

    public CircleTest()

        { radius=6500;

        }

    public int getRadius()

        { return radius;

        }

    public void setRadius(int newRadius)

        {radius=newRadius;

        }

    public double circleArea()

        {return Math.PI*radius*radius;

        }

    public double circlLength()

        {return 2.0*Math.PI*radius;

        }

}
```

下面的 JSP 页面 `useBeansTest.jsp` 存放在 web 服务目录 `examples` 中，该页面效果如图 6.9。

UseBeansTest.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="CircleTest"%>

<HTML>

<BODY bgcolor=cyan><Font size=1>

    <jsp:useBean id="girl" class="CircleTest" scope="page" >
    </jsp:useBean>

    <%-- 通过上述 JSP 标签, 客户获得了一个作用域是 page, 名字是 girl 的 beans --%>

    <P> 圆的半径是:

    <%=girl.getRadius()%>

    <p> 重新设置圆的半径是 100:

    <%

        girl.setRadius(100);

    %>

    <P>现在圆的半径是:

    <%=girl.getRadius()%>

    <P>圆的周长是:

    <%=girl.circlLength()%>

    <P>圆的面积是:

    <%=girl.circleArea()%>
```

</BODY>

</HTML>

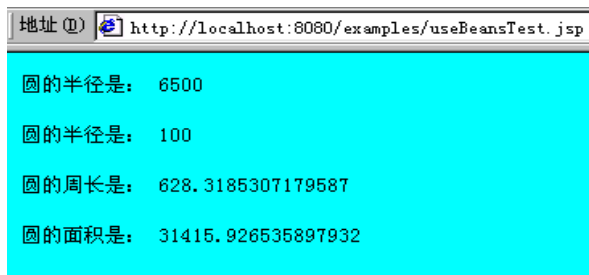


图 6.9 只对 examples 服务目录可用的 beans

6.3 获取和修改 beans 的属性

当我们使用 **useBean** 动作标签创建一个 **beans** 后，在 **java** 程序片中这个 **beans** 就可以调用方法产生行为，比如修改属性，使用类的中的方法等，如前面的例子 1、2、3 所示。获取或修改 **beans** 的属性还可以使用动作标签 **getProperty**、**setProperty**，下面讲述怎样使用 **JSP** 的动作标签去获取和修改 **beans** 的属性。

6.3.1 getProperty 动作标签

使用该标签可以获得 **beans** 的属性值，并将这个值用串的形式显示给客户。使用这个标签之前，必须使用 **useBean** 标签获取得到一个 **beans**

getProperty 动作标签：


```
<jsp:getProperty name= “beans 的名字” property= “beans 的属性” />
```

或

```
<jsp:getProperty name= “beans 的名字” property= “beans 的属性” />
```

```
</jsp:getProperty>
```

其中，**name** 取值是 **beans** 的名字，用来指定要获取哪个 **beans** 的属性的值；**property** 取值是该 **beans** 的一个属性的名字。该指令的作用相当于在程序片中使用 **beans** 调用 **getXxx()** 方法。

现在，我们将 **Circle** 类文件给予改进，增加 **circleArea** 和 **circleLength** 两个属性。

Circle2.java:

```
import java.io.*;

public class Circle2
{ double radius=1;

  double circleArea=0;

  double circleLength=0;

  public double getRadius()

  { return radius;

  }

  public void setRadius(double newRadius)

  {radius=newRadius;
```

```

    }

    public double getCircleArea()

    {circleArea=Math.PI*radius*radius;

    return circleArea;

    }

    public double getCircleLength()

    { circleLength=2.0*Math.PI*radius;

    return circleLength;

    }

}

```

在下面的 JSP 页面中使用 `useBean` 标签，用 `Circle2` 类创建一个名字是 `apple` 的 beans。并使用 `getProperty` 获取 `apple` 的各个属性的值。把 `Circle2.java` 编译生成的字节码 `Circle2.class` 文件拷贝到 JSP 引擎的 `classes` 文件下。本书所使用的机器的该目录是：

D:\tomcat\Jakarta-tomcat-4.0\classes

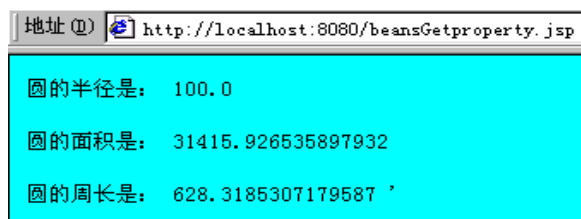


图 6.10 获取 beans 的属性值

例子 4(效果如图 6.10 所示)

beansGetProperty.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="Circle2"%>

<HTML>

<BODY bgcolor=cyan><Font size=1>

    <jsp:useBean id="apple" class="Circle2" scope="page" >

    </jsp:useBean>

    <%apple.setRadius(100);%>

    <P>圆的半径是:

    <jsp:getProperty name="apple" property="radius" />

    <P>圆的面积是:

    <jsp:getProperty name="apple" property="circleArea" />

    <P>圆的周长是:

    <jsp:getProperty name="apple" property="circleLength" />

</BODY>

</HTML>
```

6.3.2 setProperty 动作标签

使用该标签可以设置 beans 的属性值。使用这个标签之前,必须使用 useBean 标签得到一个可操作的 beans。

setProperty 动作标签可以通过 3 种方式设置 beans 属性的值。

(1) 将 benas 属性的值设置为一个表达式的值或字符串。

这种方式不如后面的两种方式方便，但当涉及属性值是汉字时，使用这种方式更好一些。

benas 属性的值设置为一个表达式的值：

```
<jsp:setProperty name= “beans 的名字 ” property= “beans 的属性” value=
“<%=expression%>” />
```

benas 属性的值设置为一个字符串：

```
<jsp:setProperty name= “beans 的名字” property= “beans 的属性” value= 字符串
/>
```

如果将表达式的值设置为 **beans** 属性的值，表达式值的类型必须和 **beans** 的属性的类型一致。如果将字符串设置为 **beans** 的属性的值，这个字符串会自动被转化为 **beans** 的属性的类型。**Java** 语言将字符串转化为其它数值类型的方法如下：

转化到 **int** ： **Integer.parseInt(Sting s)**,

转化到 **long** ： **Long.parseInt(Sting s)**,

转化到 **float** ： **Float.parseInt(Sting s)**,

转化到 **double** ： **Double.parseInt(Sting s)**,

这些方法都可能发生 **NumberFormatException** 异常，例如，当试图将字符串：“**23**”转化为 **int** 型数据时就发生了 **NumberFormatException**。

在下面的例子中，我们写了一个描述学生的 **beans**，在一个 **JSP** 页面中获得一个这样的 **beans**，其有效范围是 **page**。在 **JSP** 页面中使用动作标签设置、获取该 **beans** 的属性。

创建 beans 的源文件

Student.java:

```
public class Student  
  
{ String name=null;  
  
long number;  
  
double height,weight;  
  
public String getName()  
  
    { return name;  
  
    }  
  
public void setName(String newName)  
  
    {name=newName;  
  
    }  
  
public long getNumber()  
  
    {return number;  
  
    }  
  
public void setNumber(long newNumber)  
  
    { number=newNumber;  
  
    }  
  
public double getHeight()  
  
    {return height;  
  
    }  
  
public void setHeight(double newHeight)
```

```

    { height=newHeight;

    }

    public double getWeight()

    {return weight;

    }

    public void setWeight(double newWeight)

    { weight=newWeight;

    }

}

```

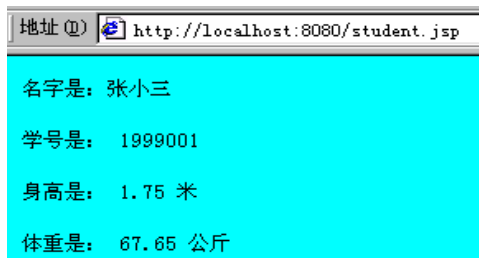


图 6.11 使用表达式的值设置 beans 的属性

例子 5(效果如图 6.11 所示)

student.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="Student"%>

```

<HTML>

<BODY bgcolor=cyan>

<jsp:useBean id="zhang" class="Student" scope="page" >

</jsp:useBean>

<jsp:setProperty name="zhang" property="name" value=" 张小三" />

<P>名字是:

<jsp:getProperty name="zhang" property="name" />

<jsp:setProperty name="zhang" property="number" value="1999001" />

<P>学号是:

<jsp:getProperty name="zhang" property="number" />

<% double height=1.70;

%>

<jsp:setProperty name="zhang" property="height" value="<%=height+0.05%>"

/>

<P>身高是:

<jsp:getProperty name="zhang" property="height" />

米

<jsp:setProperty name="zhang" property="weight" value="67.65" />

<P>体重是:

<jsp:getProperty name="zhang" property="weight" />

公斤

</BODY>

</HTML>

(2) 使用 `setProperty` 设置 `beans` 属性值的第 2 种方式是：通过 HTTP 表单的参数值来设置 `beans` 的相应属性的值，要求表单参数名字必须与 `beans` 属性的名字相同，JSP 引擎会自动将字符串转换为 `beans` 属性的类型。

```
<jsp:setProperty name="beans 的名字" property="*" />
```

该标签不用再具体指定 `beans` 属性的值将对应表单中哪个参数指定的值，系统会自动根据名字进行般配对应。

下面的例子 6 通过表单来指定 `beans` 的属性值。由于客户可能提交汉语的姓名，所以我们将 `Student.java` 文件中的 `getName` 方法做以下改进：

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

地址 http://localhost:8080/student2.jsp

输入学生的姓名: 张民民

输入学生的学号: 2002008

输入学生的身高: 1.78

输入学生的体重: 76 提交

名字是: 张民民

学号是: 2002008

身高是: 1.78 米

体重是: 76.0 公斤

例子 6(效果如图 6.12 所示)

student2.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import="Student"%>
```

```
<HTML>
```

```
<BODY ><Font size=1>
```

```
<FORM action="" Method="post" >
```

```
<P>输入学生的姓名:
```

```
<Input type=text name="name">
```

```
<P>输入学生的学号:
```

```
<Input type=text name="number">
```

```
<P>输入学生的身高:
```

```
<Input type=text name="height">
```

```
<P>输入学生的体重:
```

```
<Input type=text name="weight">
```

```
<Input type=submit value="提交">

</FORM>

<jsp:useBean id="zhang" class="Student" scope="page" >

</jsp:useBean>

<jsp:setProperty name= "zhang" property="*" />

<P>名字是:

<jsp:getProperty name= "zhang" property="name" />

<P>学号是:

<jsp:getProperty name= "zhang" property="number" />

<P>身高是:

<jsp:getProperty name= "zhang" property="height" />

米

<P>体重是:

<jsp:getProperty name= "zhang" property="weight" />

公斤

</FONT>

</BODY>

</HTML>
```

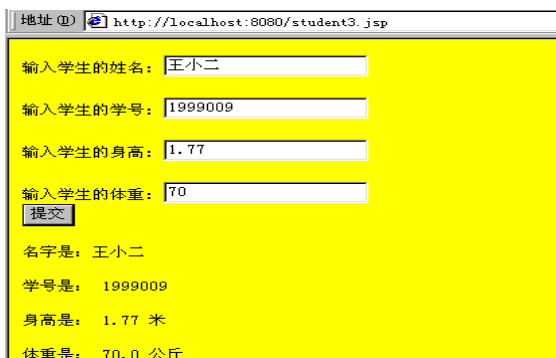
注：需要注意的是，只有提交了和该 beans 相对应的表单后，该指令标签才被执行。

注：使用这种方式设置和获取 beans 的值很方便，我们把汉字的处理放在了 beans 中，但需要注意的是，如果使用第 1 种方式，不要在 beans 中处理汉字，而应当在 JSP 页面中处理。

(3) 使用 setProperty 设置 beans 属性值的第 3 种方式是：通过 request 的参数值

来设置 beans 的相应属性的值, 要求 request 参数名字必须与 beans 属性的名字相同, JSP 引擎会自动将 request 获取的字符串数据类型转换为 beans 相应的属性的类型。

```
<jsp:setProperty name="beans 的名字" property=" 属性名" param= “参数名”  
/>
```



地址 http://localhost:8080/student3.jsp

输入学生的姓名: 王小二

输入学生的学号: 1999009

输入学生的身高: 1.77

输入学生的体重: 70

提交

名字是: 王小二

学号是: 1999009

身高是: 1.77 米

体重是: 70.0 公斤

图 6.13 使用 request 参数设置属性值

下面的例子 7 说明使用 request 参数设置 beans 的属性的值。

例子 7(效果如图 6.13 所示)

student3.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```

<%@ page import="Student"%>

<HTML>

<BODY bgcolor=yellow><Font size=1>

<FORM action="" Method="post" >

<P>输入学生的姓名:

<Input type=text name="name">

<P>输入学生的学号:

<Input type=text name="number">

<P>输入学生的身高:

<Input type=text name="height">

<P>输入学生的体重:

<Input type=text name="weight">

<BR> <Input type=submit value="提交">

</FORM>

<jsp:useBean id="zhang" class="Student" scope="page" >

</jsp:useBean>

<jsp:setProperty name="zhang" property="name" param="name" />

<P>名字是:

<jsp:getProperty name="zhang" property="name" />

<jsp:setProperty name="zhang" property="number" param="number" />

<P>学号是:

<jsp:getProperty name="zhang" property="number" />

<% double height=1.70;

```

%>

```
<jsp:setProperty name= "zhang" property="height" param="height" />
```

<P>身高是:

```
<jsp:getProperty name= "zhang" property="height" />
```

米

```
<jsp:setProperty name= "zhang" property="weight" param="weight" />
```

<P>体重是:

```
<jsp:getProperty name= "zhang" property="weight" />
```

公斤

</BODY>

</HTML>

注: 需要注意的是, 只有提交了和该 beans 相对应的表单后, 该指令标签才被执行。

注: 不能在<jsp:setProperty> 中同时使用 value 和 param。

6.4 beans 的辅助类

通过上面的学习, 我们已经知道怎样使用一个简单的 beans。有时我们在写一个 beans 的时候, 除了需要用 **import** 语句引入 Java 的内置包中的类, 可能还需要其它自己写的一些类。那么你只要将这些类的字节码文件放在默认的 **classes** 目录中即可, 本书配置的服务器的 **classes** 目录在:

D:\Tomcat\jakarta-tomcat-4.0\classes

在下面的例子 8 中, 我们使用一个 beans: **ListFile**, 列出 JSP 页面所在目录中特定扩展名的文件。在写 beans 的类文件 **ListFile** 时, 我们需要一个实现 **FilenameFilter** 接口的辅助类: **FileName**, 该类可以帮助我们的 beans 列出指定扩展名的文件。

创建 beans 的源文件

ListFile.java:

```
import java.io.*;

class FileName implements FilenameFilter

    { String str=null;

        FileName (String s)

            {str="."+s;

                }

        public boolean accept(File dir,String name)

            { return name.endsWith(str);

                }

    }

public class ListFile

{

    String extendsName=null;

    public void setExtendsName(String s)

    { extendsName=s;

        }

    public String getExtendsName()

    {return extendsName;

        }

    public String[] listFile()
```

```

{ File dir=new File("d:/Tomcat/Jakarta-tomcat-4.0/webapps/root/");

  FileName file_jsp=new FileName(extendsName);

  String file_name[]=dir.list(file_jsp);

  return file_name;

}

}

```

上述 java 源文件编译通过后，会生成两个字节码文件：**ListFile.class** 和 **FileName.class**。我们需要将这两个字节码文件放在 **classes** 目录中。

注：也可以将上述的 **ListFile.java** 文件分为两个 java 文件：**ListFile.java** 和 **FileNmae.java**，然后，首先编译 **FileNmae.java**，再编译 **ListFile.java**，生成各自的字节码文件，然后将这两个字节码文件存放在 **classes** 目录中。

现在，就可以在 JSP 页面中使用 **FileList** 创建 beans 了。在下面的例子 8 中，客户通过表单设置 beans 的 **extendsName** 属性的值，beans 列出相应扩展名的文件给客户。



图 6.14 列出特定扩展名的文件

例子 8(效果如图 6.14 所示)

listfile.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="ListFile" %>

<HTML>

<BODY ><Font size=1>

<FORM action="" Method="post" >

<P>输入文件的扩展名:

<Input type=text name="extendsName">

<Input type=submit value="提交">

<jsp:useBean id="file" class="ListFile" scope="page" >

</jsp:useBean>

<jsp:setProperty name="file" property="*" />

<P>当前 JSP 页面所在目录中, 扩展名是:

<jsp:getProperty name="file" property="extendsName" />

文件有:

<% String name[]=file.listFiles();

    for(int i=0;i<name.length;i++)

        {out.print("<BR>" +name[i]);

        }

%>

</BODY>
```


</HTML>

6.5 带包名的 beans

在写一个 beans 的 java 文件时，可以使用 **package** 语句给 beans 一个包名。包名可以是一个合法的标识符，也可以是若干个标识符加 “.” 分割而成，如：

```
package gping;
```

```
package tom.jiafei;
```

程序如果使用了包语句，例如

```
package tom.jiafei;
```

那么必须在 **classes** 目录下有如下的子目录，例如，在 **D:\Tomcat\jakarta-tomcat-4.0\classes** 下再建立如下的目录结构。

\tom\jiafei

并将 beans 的字节码文件存在该目录中，如图 6.15 所示。

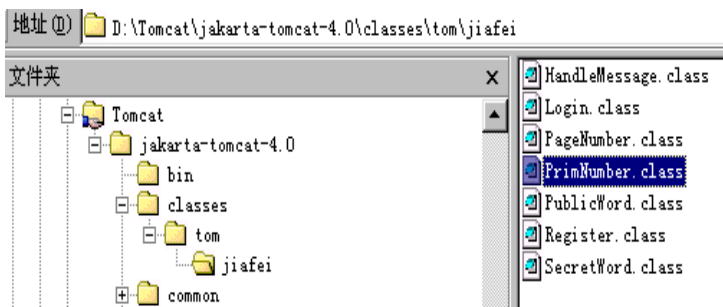


图 6.15 带包名的 beans 的存放目录

假如有一个包名是 **tom.jiafei** 的类 **Primnumber**，如果在 **JSP** 页面中使用这个 **beans**，必须有如下的语句：

```
<%@ import = "tom.jiafei.* ">
```

或

```
<%@ import = "tom.jiafei.PrimNumber ">
```

求素数的 beans

PrimNumber.java:

```
package tom.jiafei;

public class PrimNumber

{ int number;

  StringBuffer result;

  public PrimNumber()

  { result=new StringBuffer();

  }
```

```

public void setNumber(int n)
{
    number=n;
    int i,j;
    for( i=2;i<=number;i++) //    找出 number 以内的素数。
    {
        for(j=2;j<i;j++)
        {
            if(i%j==0)
            {
                break;
            }
        }
        if(j>=i)
        {
            result.append(""+i+"<BR>");
        }
    }
}

public int getNumber()
{
    return number;
}

public StringBuffer getResult()
{
    return result;
}
}

```

例子 9 使用求素数 beans 的 JSP 页面(效果如图 6.16 所示)

primnumber.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```

<%@ page import="tom.jiafei.*" %>

<HTML>

<BODY bgcolor=cyan><Font size=1>

<FORM action="" Method="post" >

<P>输入一个正整数:

<Input type=text name="number">

<Input type=submit value="提交">

    <jsp:useBean id="prim" class="tom.jiafei.PrimNumber" scope="page" >

    </jsp:useBean>

    <jsp:setProperty name= "prim" property="number" param="number" />

    <P>小于

    <jsp:getProperty name= "prim" property="number" />

    这个数的全部素数是:

    <BR> <jsp:getProperty name= "prim" property="result" />

    </Font>

</BODY>

</HTML>

```

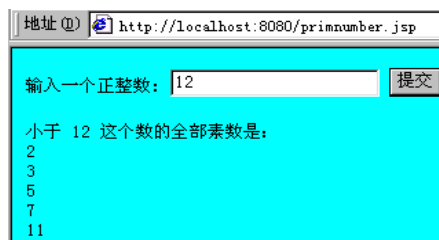


图 6.16 使用求素数的 beans

6.6 JSP 与 beans 结合的简单例子

6.6.1 三角形 beans

三角形 beans

Triangle.java:

```
public class Triangle
```

```
{ double sideA,sideB,sideC;
```

```
    double area;
```

```
    boolean triangle;
```

```
    public void setSideA(double a
```

```
    {sideA=a;
```

```
    }
```

```
    public double getSideA()
```

```
    {return sideA;
```

```
    }
```

```
    public void setSideB(double b)
```

```
    {sideB=b;
```

```
    }
```

```
    public double getSideB()
```

```
    {return sideB;
```

```
    }
```

```
    public void setSideC(double c)
```

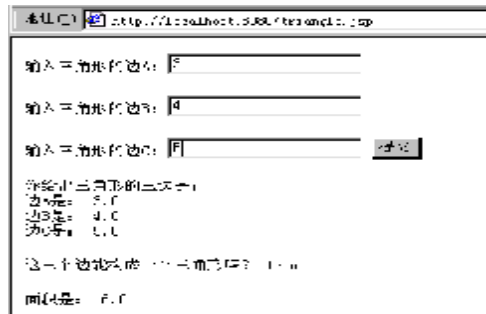


图 6.17 使用 beans 计算三角形面积

```

    {sideC=c;
    }

    public double getSideC()

    {return sideC;
    }

    public double getArea()

    { double p=(sideA+sideB+sideC)/2.0;

      area=Math.sqrt(p*(p-sideA)*(p-sideB)*(p-sideC));

      return area;
    }

    public boolean isTriangle()

    { if(sideA<sideB+sideC&&sideB<sideA+sideC&&sideC<sideA+sideB)

      triangle=true;

      else

      triangle=false;

      return triangle;
    }

}

```

使用三角形 beans 的 JSP 页面(效果如图 6.17 所示)

triangle.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

```

```

<%@ page import="Triangle"%>

<HTML>

<BODY ><Font size=1>

<FORM action="" Method="post" >

<P>输入三角形的边 A:

<Input type=text name="sideA" value=0>

<P>输入三角形的边 B:

<Input type=text name="sideB" value=0>

<P>输入三角形的边 C:

<Input type=text name="sideC" value=0>

<Input type=submit value="提交">

<P>你给出三角形的三边是:

<jsp:useBean id="triangle" class="Triangle" scope="page" >

</jsp:useBean>

<jsp:setProperty name="triangle" property="*" />

<BR>边 A 是:

<jsp:getProperty name="triangle" property="sideA" />

<BR>边 B 是:

<jsp:getProperty name="triangle" property="sideB" />

<BR>边 C 是:

<jsp:getProperty name="triangle" property="sideC" />

<P>这三个边能构成一个三角形吗?

<jsp:getProperty name="triangle" property="triangle" />

```

<P>面积是:

<jsp:getProperty name= "triangle" property="area" />

</BODY>

</HTML>

6.6.2 计数器 beans

计数器 beans 是一个 application 范围的 beans，所有的客户共享这个 beans，任何客户改变这个 beans 的属性值，都会对其它客户产生影响。计数器 beans 有一个记录访问次数的属性 count。

Counter.java:

```
public class Counter  
{ long count=0;  
  
    public synchronized long getCount()  
  
        {count++;  
  
        return count;  
  
    }  
}
```

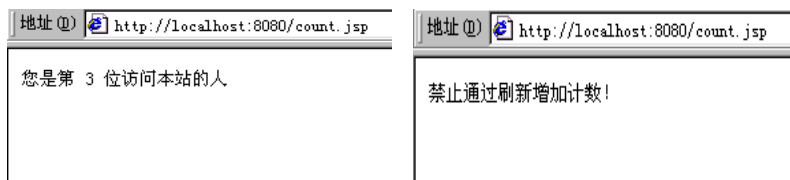


图 6.18 使用 beans 的计数器

count.jsp: (效果如图 6.18 所示)

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="Counter"%>

<HTML>

<BODY ><Font size=1>

<jsp:useBean id="people" class="Counter" scope="application" >

</jsp:useBean>

<% if(session.isNew())

{ %>

<P> 您是第

<jsp:getProperty name="people" property="count" />

位访问本站的人

<%

}

else

{ out.print(" 禁止通过刷新增加计数! ");

}

%>

</FONT>

</BODY>
```

</HTML>

6.6.3 购物车 beans

Car1.java:

```
import java.util.*;

import java.io.*;

public class Car1 implements Serializable
{
    Hashtable list=new Hashtable();

    String item="Welcome!";

    int mount=0;

    String unit=null;

    public void Car1()
    {
    }

    public void setItem(String newItem)
    {
        item=newItem;
    }

    public void setUnit(String newUnit)
    {
        unit=newUnit;
    }

    public void setMount(int m)
    {
        mount=m;
    }

    public void 添加商品到购物车()
```

```

{ String str="Name: "+item+" Mount:"+mount+" Unit:"+unit;

  list.put(item,str);

}

public Hashtable 列出购物车中的商品()

{ return list;

}

public void 删除货物(String s)

{ list.remove(s);

}

}

```

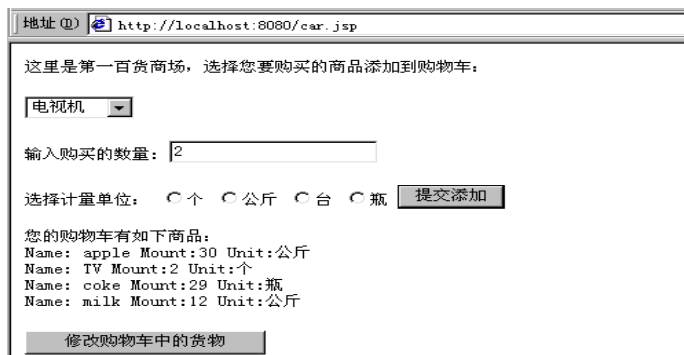


图 6.19 选择商品

选择物品页面(效果如图 6.19 所示)

car.jsp:

```
<%@ page contentType="text/html; charset=GB2312" %>

<%@ page import="java.util.*" %>

<%@ page import="Car1" %>

<HTML>

<BODY ><Font size=1>

<jsp:useBean id="car1" class="Car1" scope="session" >

</jsp:useBean>

<P>这里是第一百货商场，选择您要购买的商品添加到购物车：

<% String str=response.encodeRedirectURL("add.jsp");

%>

<FORM action="<%=str%>" method=post name=form>

    <Select name="item" value="    没选择">

        <Option value="TV">        电视机

        <Option value="apple">        苹果

        <Option value="coke">        可口可乐

        <Option value="milk">        牛奶

        <Option value="tea">        茶叶

    </Select>

<P>输入购买的数量：

    <Input type=text name="mount">

<P>选择计量单位：

    <INPUT type="radio" name="unit" value="    个">个
```

```

<INPUT type="radio" name="unit" value=" 公斤">公斤
<INPUT type="radio" name="unit" value=" 台">台
<INPUT type="radio" name="unit" value=" 瓶">瓶
<Input type=submit value="提交添加">
</FORM>
<P>您的购物车有如下商品：
<% Hashtable list=car1.列出购物车中的商品();

Enumeration enum=list.elements();

while(enum.hasMoreElements())

    { String goods=(String)enum.nextElement();

      byte b[]=goods.getBytes("ISO-8859-1");

      goods=new String(b);

      out.print("<BR>" + goods);

    }

%>

<% String str1=response.encodeRedirectURL("selectRemovedGoods.jsp");

%>

<FORM action="<%=str1%>" method=post name=form>

<Input type=submit value="修改购物车中的货物">

</FORM>

</FONT>

</BODY>

</HTML>

```

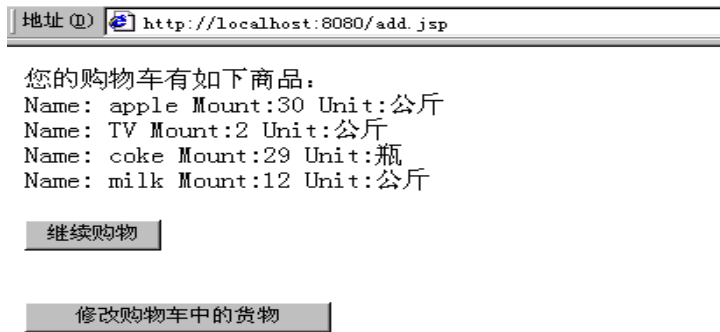


图 6.20 添加商品

添加货物到购物车页面(效果如图 6.20 所示)

add.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.util.*" %>

<%@ page import="Car1" %>

<HTML>

<BODY ><Font size=2>

<jsp:useBean id="car1" class="Car1" scope="session" >

</jsp:useBean>

<jsp:setProperty name="car1" property="*" />

<% car1.添加商品到购物车();
```

%>

<P>您的购物车有如下商品：

<% Hashtable list=car1.列出购物车中的商品();

Enumeration enum=list.elements();

while(enum.hasMoreElements())

{ String goods=(String)enum.nextElement();

byte b[]=goods.getBytes("ISO-8859-1");

goods=new String(b);

**out.print("
" + goods);**

}

%>

<% String str=response.encodeRedirectURL("car.jsp");

%>

**
<FORM action="<%=str%>" method=post name=form>**

<Input type=submit value="继续购物">

</FORM>

<% String str1=response.encodeRedirectURL("selectRemovedGoods.jsp");

%>

**
<FORM action="<%=str1%>" method=post name=form>**

<Input type=submit value="修改购物车中的货物">

</FORM>

</BODY>

</HTML>



图 6.21 选择删除的商品

选择删除货物的页面(效果如图 6.21 所示)

selectRemovedGoods.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import="java.util.*" %>
```

```
<%@ page import="Car1" %>
```

```
<HTML>
```

```
<BODY ><Font size=1>
```

```
<jsp:useBean id="car1" class="Car1" scope="session" >
```

```
</jsp:useBean>
```

```
<P>选择从购物车删除的商品:
```

```
<% String str=response.encodeRedirectURL("removeWork.jsp");
```


%>

<FORM action="<%=str%>" method=post name=form2>

<Select name="deleteitem" size=1>

<Option value="TV"> 电视机

<Option value="apple"> 苹果

<Option value="coke"> 可口可乐

<Option value="milk"> 牛奶

<Option value="tea"> 茶叶

</Select>

<Input type=submit value=" 提交删除">

</FORM>

<P> 您的购物车有如下商品:

<% Hashtable list=car1. 列出购物车中的商品();

Enumeration enum=list.elements();

while(enum.hasMoreElements())

{ String goods=(String)enum.nextElement();

byte b[]=goods.getBytes("ISO-8859-1");

goods=new String(b);

**out.print("
" + goods);**

}

%>

<% String str1=response.encodeRedirectURL("car.jsp");

%>

```

<FORM action="<%=str1%>" method=post name=form>

<Input type=submit value="继续购物">

</FORM>

</FONT>

</BODY>

</HTML>

```



图 6.22 删除货物

删除货物页面(效果如图 6.22 所示)

removeWork.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.util.*" %>

<%@ page import="Car1" %>

<HTML>

```

```

<BODY ><Font size=1>

<jsp:useBean id="car1" class="Car1" scope="session" >

</jsp:useBean>

<% String name=request.getParameter("deleteitem");

    if(name==null)

        {name="";

        }

    byte c[]=name.getBytes("ISO-8859-1");

    name=new String(c);

    car1. 删除货物(name);

    out.print(" 您删除了货物: "+name);

%>

<P>购物车中现在的货物:

<% Hashtable list=car1.列出购物车中的商品();

    Enumeration enum=list.elements();

    while(enum.hasMoreElements())

        { String goods=(String)enum.nextElement();

            byte b[]=goods.getBytes("ISO-8859-1");

            goods=new String(b);

            out.print("<BR>" +goods);

        }

%>

<% String str1=response.encodeRedirectURL("car.jsp");

```

```
%>
```

```
<FORM action="<%=str1%>" method=post name=form>
```

```
<Input type=submit value="继续购物">
```

```
</FORM>
```

```
<% String str=response.encodeRedirectURL("selectRemovedGoods.jsp");
```

```
%>
```

```
<FORM action="<%=str%>" method=post name=form1>
```

```
<Input type=submit value="修改购物车中的货物">
```

```
</FORM>
```

6.6.4 读文件 beans

下面的 **beans** 可以列出特定目录中的文件、并可读取目录中文件的内容

ReadFile.java:

```
import java.io.*;

public class ReadFile

{ String filePath="c:/",fileName="";

  // 设置目录属性的值:

  public void setFilePath(String s)

  {filePath=s;

    try{byte b[]=filePath.getBytes("ISO-8859-1");

      filePath= new String(b);

    }

    catch(Exception ee)
```

```

        { }

    }

// 设置文件名字属性的值:

public String getFilePath()

{return filePath;

}

public void setFileName(String s)

{ fileName=s;

    try{byte b[]=fileName.getBytes("ISO-8859-1");

        fileName=new String(b);

    }

    catch(Exception ee)

    { }

}

public String getFileName()

{return fileName;

}

// 列出目录中的文件:

public String[] listFile()

{ File dir=new File(filePath);

    String file_name[]=dir.list();

    return file_name;

}

```

// 读取文件的原始信息:

```
public StringBuffer readFile()
```

```
{ try{ File file=new File(filePath,fileName);
```

```
    FileReader in=new FileReader(file) ;
```

```
    PushbackReader push=new PushbackReader(in);
```

```
    StringBuffer stringbuffer=new StringBuffer();
```

```
    int c;
```

```
    char b[]=new char[1];
```

```
    while ( (c=push.read(b,0,1))!=-1)//      读取 1 个字符放入字符数组 b。
```

```
    { String s=new String(b);
```

```
        if(s.equals("<"))      //      回压的条件
```

```
        { push.unread('&');
```

```
            push.read(b,0,1); //push      读出被回压的字符字节,放入数组 b.
```

```
            stringbuffer.append(new String(b));
```

```
            push.unread('L');
```

```
            push.read(b,0,1); //push      读出被回压的字符字节,放入数组 b.
```

```
            stringbuffer.append(new String(b));
```

```
            push.unread('T');
```

```
            push.read(b,0,1); //push      读出被回压的字符字节,放入数组 b.
```

```
            stringbuffer.append(new String(b));
```

```
        }
```

```
    else if(s.equals(">"))      //      回压的条件
```

```
    { push.unread('&');
```

```

        push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
        stringBuffer.append(new String(b));
        push.unread('G');
        push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
        stringBuffer.append(new String(b));
        push.unread('T');
        push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
        stringBuffer.append(new String(b));
    }
    else if(s.equals("\n"))
    { stringBuffer.append("<BR>");
    }
    else
    { stringBuffer.append(s);
    }
}

push.close();

in.close();

return stringBuffer;
}

catch(IOException e)

{return new StringBuffer("    不能读取文件");

}

```

```
}  
  
}
```



图 6.23 选择目录

选择目录的页面(效果如图 6.23 所示)

filepathselect.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>  
  
<%@ page import="java.util.*" %>  
  
<%@ page import="ReadFile" %>  
  
<HTML>  
  
<BODY ><Font size=1>  
  
<P>请选择一个目录:  
  
<FORM action="listfilename.jsp" method=post>  
  
<Select name="filePath" >  
  
<Option value="f:/2000"> f:/2000  
  
<Option value="d:/tomcat">D:/tomcat  
  
<Option value="d:/tomcat/jakarta-tomcat-4.0/webapps/root">Root
```



```

<Option value="F:/javabook">f:/javabook

<Option value="f:/Example">f:/Example

</Select>

<Input type=submit value="提交">

</FORM>

</FONT>

</BODY>

</HTML>

```



图 6.24 选择文件

选择文件页面(效果如图 6.24 所示)

listfilename.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="ReadFile" %>

<HTML>

<BODY ><Font size=2>

```

```

<jsp:useBean id="file" class="ReadFile" scope="session" >

</jsp:useBean>

<jsp:setProperty name= "file" property="filePath" param="filePath" />

<P>该目录

<jsp:getProperty name= "file" property="filePath" />

有如下文件:<BR>

<% String name[]=file.listFiles();

    for(int i=0;i<name.length;i++)

        {out.print("<BR>" +name[i]);

        }

    %>

<Form action=readfile.jsp method="post">

    <P> 输入文件的名字:

    <Input type=text name="fileName" name= "f" >

    <Input type=submit value="提交">

</Form>

<BR>

<FORM action="filepathselect.jsp" method=post name=form>

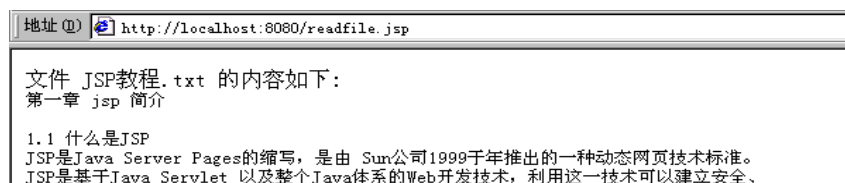
<Input type=submit value="重新选择目录">

</FORM>

</Body>

</HTML>

```



读取文件内容页面(效果如图 6.25 所示)

readfile.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="ReadFile" %>

<HTML>

<BODY ><Font size=2>

<jsp:useBean id="file" class="ReadFile" scope="session" >

</jsp:useBean>

<jsp:setProperty name= "file" property="fileName" param="fileName" />

<P>文件
```

```
<jsp:getProperty name="file" property="fileName" />
```

的内容如下:


```
</Font>
```

```
<Font size=1>
```

```
<% StringBuffer s=file.readFile();
```

```
    out.print(s);
```

```
%>
```

```
<FORM action="filepathselect.jsp" method=post name=form>
```

```
<Input type=submit value="重新选择目录">
```

```
</FORM>
```

```
<BR>
```

```
<FORM action="listfilename.jsp" method=post name=form>
```

```
<Input type=submit value="重新选择文件">
```

```
</FORM>
```

```
</Body>
```

```
</HTML>
```

6.6.5 写文件 beans

这个写文件的 **beans** 可以将客户提交的文本内容写入一个指定目录的文件中。

WriterFile.java:

```
import java.io.*;
```

```
public class WriterFile
```

```
{ String filePath=null,
```

```

        fileName=null,

        fileContent=null;

public WriterFile()

{ filePath="C:/";

    fileName="    无标题";

    fileContent="    无内容";

}

public void setFilePath(String s)

{filePath=s;

    try{byte b[]=filePath.getBytes("ISO-8859-1");

        filePath= new String(b);

    }

    catch(Exception ee)

    { }

}

public String getFilePath()

{return filePath;

}

public void setFileName(String s)

{ fileName=s;

    try{byte b[]=fileName.getBytes("ISO-8859-1");

        fileName=new String(b);

    }

```

```

        catch(Exception ee)

        {}

    }

    public String getFileName()

    {return fileName;

    }

```

// 获取属性 fileContent 的值，为了能显示 HTML 或 JSP 源文件，需进行流的处理技术：

```

public String getFileContent()

{ try{ StringReader in=new StringReader(fileContent) ;// 指向字符串的字符流。

    PushbackReader push=new PushbackReader(in);

    StringBuffer stringbuffer=new StringBuffer();

    int c;

    char b[]=new char[1];

    while ( (c=push.read(b,0,1))!=-1)//      读取 1 个字符放入字符数组 b。

    { String s=new String(b);

        if(s.equals("<"))      //      回压的条件

        { push.unread('&');

            push.read(b,0,1); //push      读出被回压的字符字节,放入数组 b.

            stringbuffer.append(new String(b));

            push.unread('L');

            push.read(b,0,1); //push      读出被回压的字符字节,放入数组 b.

            stringbuffer.append(new String(b));

```

```

    push.unread('T');

    push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
    stringBuffer.append(new String(b));
}

else if(s.equals(">"))    //          回压的条件
{
    push.unread('&');

    push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
    stringBuffer.append(new String(b));

    push.unread('G');

    push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
    stringBuffer.append(new String(b));

    push.unread('T');

    push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
    stringBuffer.append(new String(b));

}

else if(s.equals("\n"))
{
    stringBuffer.append("<BR>");
}

else
{
    stringBuffer.append(s);
}

}

push.close();

```

```

        in.close();

        return fileContent=new String(stringbuffer);
    }
catch(IOException e)

    {return fileContent=new String("    不能读取内容");

    }

}

// 写文件：

public void setFileContent(String s)

{ fileContent=s;

try{

    byte b[]=fileContent.getBytes("ISO-8859-1");

    fileContent=new String(b);

    File file=new File(filePath,fileName);

    FileWriter in=new FileWriter(file) ;

    BufferedWriter buffer=new BufferedWriter(in);

    buffer.write(fileContent);

    buffer.flush();

    buffer.close();

    in.close();

}

catch(Exception e)

{}

```



```
}  
  
}
```

提交文件内容的页面（包括文件所在目录、文件名及内容。效果如图 6.26 所示）

writeContent.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>  
  
<%@ page import="ReadFile" %>  
  
<HTML>  
  
<BODY ><Font size=1>  
  
<P>请选择一个目录:  
  
<FORM action="writeFile.jsp" method=post>  
  
  <Select name="filePath" >  
  
    <Option value="f:/2000"> f:/2000  
  
    <Option value="d:/tomcat">D:/tomcat  
  
    <Option value="d:/root">Root  
  
    <Option value="F:/javabook">f:/javabook  
  
    <Option value="f:/Example">f:/Example  
  
  </Select>  
  
<P>输入保存文件的 名字:  
  
<Input type=text name="fileName" >  
  
<P>输入文件的内 容:  
  
<BR>  
  
<TextArea name= "fileContent" Rows= "10" Cols= "40" >
```

</TextArea>

**
<Input type=submit value="提交">**

</FORM>

</BODY>

</HTML>



地址  http://localhost:8080/writeContent.jsp

请选择一个目录:

输入保存文件的名字:

输入文件的内容:

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="ReadFile" %>
<HTML>
<BODY ><Font size=1>
<P>请选择一个目录:
<FORM action="writeFile.jsp" method=post>
  <Select name="FilePath" >
    <Option value="f:/2000"> f:/2000
    <Option value="d:/tomcat">D:/tomcat
    <Option value="d:/tomcat/jakarta-tomcat->
```

图 6.26 提交内容和保存文件的名字

将内容写入文件的页面(效果如图 6.27 所示)

writeFile.jsp:

<%@ page contentType="text/html; charset=GB2312" %>

```

<%@ page import="WriterFile" %>

<HTML>

<BODY ><Font size=1>

<jsp:useBean id="file" class="WriterFile" scope="page" >

</jsp:useBean>

<jsp:setProperty name="file" property="filePath" param="filePath" />

<jsp:setProperty name="file" property="fileName" param="fileName" />

<jsp:setProperty name="file" property="fileContent" param="fileContent" />

<BR>你写文件到目录:

<jsp:getProperty name="file" property="filePath" />

<BR>文件的 名字是:

<jsp:getProperty name="file" property="fileName" />

<BR>文件的 内容是:

<jsp:getProperty name="file" property="fileContent" />

</Font>

</Body>

</HTML>

```



图 6.27 将内容写入文件并显示写入的信息

6.6.6 查询数据库 beans

我们给出一个通过数据库的主关键字进行记录查询的 **beans**，其中数据源是 **sun**（一个 **SQL server** 数据库，见第 4 章）。

DataBaseInquire.java:

```
import java.sql.*;

public class DataBaseInquire
{ String keyword;

  public DataBaseInquire()
  {keyword="";

    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

      }

    catch(ClassNotFoundException e){}

  }

  public void setKeyword(String s)

  {keyword=s;

    try{byte b[]=keyword.getBytes("ISO-8859-1");
```

```

        keyword=new String(b);

    }

    catch(Exception e)

    { }

}

public StringBuffer byKeywordInquire()

{ String number,xingming;

    Connection con=null;

    Statement sql=null;

    ResultSet rs=null;

    StringBuffer buffer=new StringBuffer();

    int math,english,physics;

    try{ con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

        sql=con.createStatement();

        String condition="SELECT * FROM students WHERE

"+keyword+""";

        rs=sql.executeQuery(condition);

        buffer.append("<Table Border>");

        buffer.append("<TR>");

        buffer.append("<TH width=100>"+

        buffer.append("<TH width=100>"+

        buffer.append("<TH width=50>"+

        buffer.append("<TH width=50>"+

```

学 号 =

学号");

姓名");

数学成绩");

英语成绩");

```

        buffer.append("<TH width=50>"+"          物理成绩");

        buffer.append("</TR>");

while(rs.next())

    { buffer.append("<TR>");

        number=rs.getString(1);

        buffer.append("<TD >" +number+"</TD>");

        xingming=rs.getString(2);

        buffer.append("<TD >" +xingming+"</TD>");

        math=rs.getInt("          数学成绩");

        buffer.append("<TD >" +math+"</TD>");

        english=rs.getInt("          英语成绩");

        buffer.append("<TD >" +english+"</TD>");

        physics=rs.getInt("          物理成绩");

        buffer.append("<TD >" +physics+"</TD>");

        buffer.append("</TR>" );

    }

    buffer.append("</Table>");

    con.close();

    return buffer;

}

catch(SQLException e)

    {return new StringBuffer("          无法建立查询");

    }

```

}

}

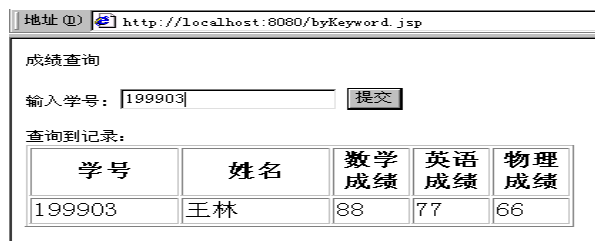


图 6.28 使用 beans 查询数据库

通过主关键字查询记录的页面(效果如图 6.28 所示)

byKeyword.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import="DataBaseInquire" %>
```

```
<HTML>
```

```
<BODY ><Font size=1>
```

```
<Font size=1>
```

```
<FORM action="byKeyword.jsp" Method="post" 1>
```

```
<P> 成绩查询
```

```
<P> 输入学号:
```

```
<Input type=text name="keyword">
```

```
<Input type=submit name="g" value=" 提交">
```

```

</Form>

<jsp:useBean id="database" class="DataBaseInquire" scope="request" >
</jsp:useBean>

<jsp:setProperty name="database" property="keyword" param="keyword" />

<P>查询到记录: <BR>

<% StringBuffer b=database.byKeywordInquire();

%>

<%=b%>

</Body>

</HTML>

```

6.6.7 猜数字 beans

当客户访问 `getNumber.jsp` 页面时，随机获取一个 1 到 100 之间的整数，由客户去猜测这个数是多少。

`GuessNumber.java`:

```

public class GuessNumber
{
    int answer=0, // 实际答案。

    guessNumber=0, // 客户猜测的数。

    guessCount=0; // 客户猜到正确答案之前所用的次数。

    String result=null;

    public void setAnswer(int n)
    {
        answer=n;

        guessCount=0;
    }
}

```



```

    }

    public int  getAnswer()

    {return answer;

    }

    public void  setGuessNumber(int n)

    { guessNumber=n;

      guessCount++;

      if(guessNumber==answer)

          result="    恭喜，猜对了";

      else if(guessNumber>answer)

          result="    猜大了";

      else if(guessNumber<answer)

          result="    猜小了";

    }

    public int getGuessNumber()

    {return guessNumber;

    }

    public int getGuessCount()

    {return guessCount;

    }

    public String getResult()

    {return result;

    }

```

}

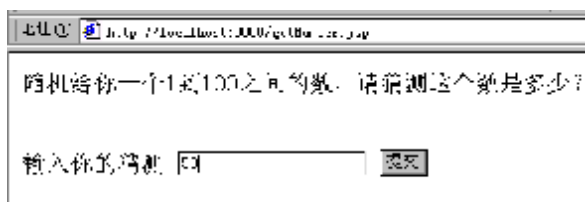


图 6.29 获得一个随机数

获取一个随机数页面(效果如图 6.29 所示)

getNumber.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="GuessNumber" %>

<HTML>

<BODY>

<% int n=(int)(Math.random()*100)+1;%>

<jsp:useBean id="guess" class="GuessNumber" scope="session" >

</jsp:useBean>

<jsp:setProperty name="guess" property="answer" value="<%=n%>" />

<p>随机给你一个 1 到 100 之间的数，请猜测这个数是多少？

<% String str=response.encodeRedirectURL("guess.jsp");

%>

<Form action="<%=str%>" method=post >
```

```

<BR>输入你的猜测 <Input type=text name="guessNumber">

<Input type=submit value="提交">

</FORM>

</BODY>

```

猜数页面(效果如图 6.30 所示)

guess.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="GuessNumber" %>

<HTML>

<BODY>

<jsp:useBean id="guess" class="GuessNumber" scope="session" >

</jsp:useBean>

<jsp:setProperty name="guess" property="guessNumber"

param="guessNumber" />

<BR>

<jsp:getProperty name="guess" property="result" />

<br>这是第

<jsp:getProperty name="guess" property="guessCount" />

猜

<BR>你给出的数是

<jsp:getProperty name="guess" property="guessNumber" />

<% String str=response.encodeRedirectURL("guess.jsp");

```

```

%>

<Form action="<%=str%>" method=post >

<BR>再输入你的猜测 <Input type=text name="guessNumber">

<Input type=submit value="提交">

</FORM>

<% String str1=response.encodeRedirectURL("getNumber.jsp");

%>

<BR><FORM action="<%=str1%>" method="post" name="f">

    <Input type="submit" value="    重新玩">

</FORM>

</BODY>

</HTML>

```

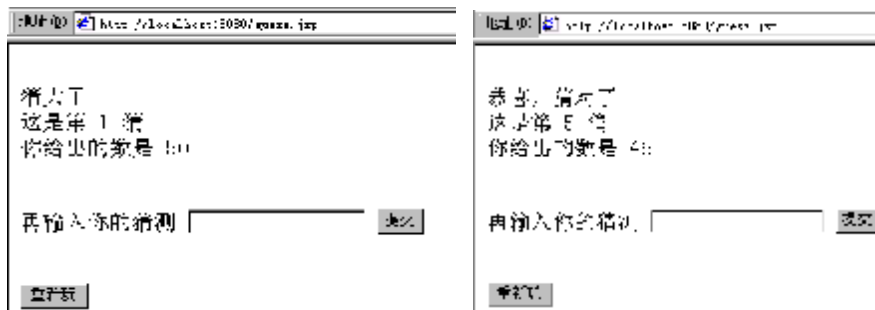


图 6.30 猜数

6.6.8 标准化考试 beans

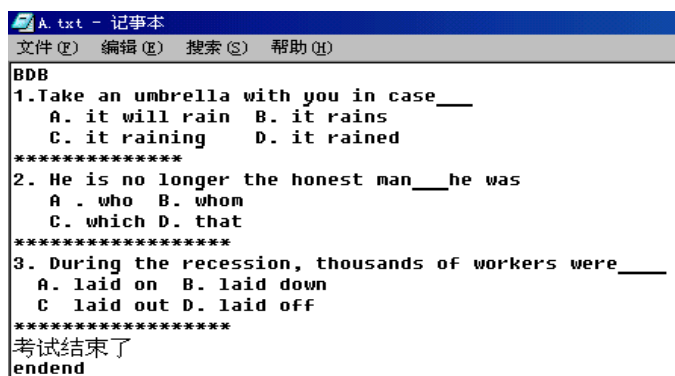
我们用一个 **beans** 每次读取一道试题。

试题文件的书写格式是：

- l 第一行必须是全部试题的答案（用来判定考试者的分数）。
- l 每道题目之间用一行：*****分割(至少含有两个**）。
- l 整个试题用 **endend** 结尾（编写 **beans** 需要这个标记）

试题可以是任何一套标准的测试题，例如英语的标准化考试，包括单词测试，阅读理解等，试题中每道题目提供 A、B、C、D 四个选择。客户可以在几套试题中任选一套试题。

Test.java 负责每次读入试题的一道题目，客户阅读题目，然后给出自己的选择。试题存放在 **F:/2000** 目录中。例如，试题 **A.Txt** 的书写格式如图 6.31。



```
BDB
1.Take an umbrella with you in case___
  A. it will rain  B. it rains
  C. it raining   D. it rained
*****
2. He is no longer the honest man___he was
  A . who  B. whom
  C. which D. that
*****
3. During the recession, thousands of workers were___
  A. laid on  B. laid down
  C laid out D. laid off
*****
考试结束了
endend
```

图 6.31 试题书写格式

Test.java:

```
import java.io.*;

public class Test

{ String filename="", // 存放考题文件名字的字符串。

  correctAnswer="", // 存放正确答案的字符串。

  // 存放试题和客户提交的答案的字符串:

  testContent="",

  selection="";

  int score=0;      //      考试者的得分。

  File f=null;

  FileReader in=null;

  BufferedReader buffer=null;

  public void setFilename(String name)

  { filename=name;

    //      当选择了新的考题文件后，将用户的答案字符串清空，

    //      将分数设为 0:

    score=0;

    selection="";

    //      读取试题文件的第一行：标准答案

    try { f=new File("F:/2000",filename);
```

```
in=new FileReader(f);
```

```
buffer=new BufferedReader(in);
```

```
correctAnswer=(buffer.readLine()).trim();// 读取一行,去掉前后空
```

格。

```
}
```

```
catch(Exception e)
```

```
{testContent=" 没有选择试题";
```

```
}
```

```
}
```

```
public String getFilename()
```

```
{ return filename;
```

```
}
```

```
public String getTestContent() // 获取试题的内容
```

```
{ try { String s=null;
```

```
StringBuffer temp=new StringBuffer();
```

```
if(buffer!=null) // 如果客户选择了试题文件, buffer 就不是空对象。
```

```
{ while((s=buffer.readLine())!=null) // 继续取某个试题。
```

```
{ if(s.startsWith("**")) // 试题结束标志。
```

```
break;
```

```
// 为了能显示原始的 HTML 或 JSP 文件考题内容需使用回压
```

流技术:

```
s=getString(s+"\n");
```

```
temp.append(s);
```

```

        if(s.startsWith("endend")) //          试题文件结束标志。

        { in.close();          //          关闭和文件的连接。

          buffer.close();

        }

        testContent=new String(temp);

    }

}

else

    { testContent=new String("          没有选择试题");

    }

}

catch(Exception e)

    { testContent="          试题无内容,考试结束了!!";

    }

return testContent;

}

public void setSelection(String s)

{

    selection=selection+s; //          将用户提交的答案依次尾加到 selection。

}

public int getScore()

{ int i=selection.length()-1; //          客户提交的第 i 题答案在 selection 中的位置。

    int m=correctAnswer.length();

```



```

if(i<=m-1)

    { try{ //          判定分数:

        if(selection.charAt(i)==correctAnswer.charAt(i))

            score++;

        }

        catch(StringIndexOutOfBoundsException e)

            { i=0;

              }

    }

    return score;

}

```

//对字符串进行处理的方法:

```

public String getString(String content)

{try{ StringReader in=new StringReader(content) ;//    指向字符串的字符流。

    PushbackReader push=new PushbackReader(in); //        回压流

    StringBuffer stringbuffer=new StringBuffer(); //        缓冲字符串对象。

    int c;

    char b[]=new char[1];

    while ( ( c=push.read(b,0,1))!=-1)//        读取 1 个字符放入字符数组 b。

        { String s=new String(b);

          if(s.equals("<")) //        回压的条件

            { push.unread('&');

              push.read(b,0,1); //push        读出被回压的字符字节,放入数组 b.

```

```

        stringBuffer.append(new String(b));

        push.unread('L');

        push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.

        stringBuffer.append(new String(b));

        push.unread('T');

        push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.

        stringBuffer.append(new String(b));

    }

    else if(s.equals(">"))    //          回压的条件

    { push.unread('&');

        push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.

        stringBuffer.append(new String(b));

        push.unread('G');

        push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.

        stringBuffer.append(new String(b));

        push.unread('T');

        push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.

        stringBuffer.append(new String(b));

    }

    else if(s.equals("\n"))

    { stringBuffer.append("<BR>");

    }

    else

```

```

        { stringBuffer.append(s);

        }

    }

    push.close();

    in.close();

    return new String(stringbuffer); //      返回处理后的字符串。

}

catch(IOException e)

    {return content=new String("      不能读取内容");

    }

}

}

```

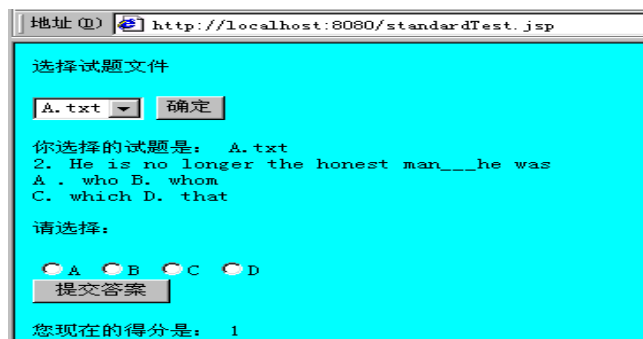


图 6.32 考试页面

考试页面(效果如图 6.32 所示)

standardTest.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="Test" %>

<HTML>

<BODY bgcolor=cyan> <Font size=1 >

<jsp:useBean id="test" class="Test" scope="session" >

</jsp:useBean>

<P>选择试题文件

<% String str=response.encodeRedirectURL("standardTest.jsp");

%>

<Form action="<%=str%>" method="post">

<Select name="filename" value="A.txt">

    <Option value="A.txt" > A.txt

    <Option value="B.txt"> B.txt

    <Option value="C.txt"> C.txt

    <Option value="D.txt"> D.txt

    <Option value="E.txt"> E.txt

</Select>

<Input type="submit" name="sub" value="确定">

</FORM>

<%-- 过 beans 设置文件的 名字 , 下面的标签只有提交了相应的表单才被执行: --%>

<jsp:setProperty name="test" property="filename" param="filename"/>

<P>你选择的试题是:
```

```

<jsp:getProperty name="test" property="filename" />

<%-- 通过 beans 获取试题的内容--%>

<BR>

    <jsp:getProperty name="test" property="testContent" />

<BR>请选择:

<BR><Form action="<%=str%>" method="post">

    <Input type="radio" name="selection" value="A">A

    <Input type="radio" name="selection" value="B">B

    <Input type="radio" name="selection" value="C">C

    <Input type="radio" name="selection" value="D">D

<BR><Input type="submit" name="tijaio" value="提交答案">

</FORM>

<% //判断客户是否选择了答案:

    String s=request.getParameter("selection");

    if(s==null)

        {s="";

        }

%>

<%if(!(s.equals("")))

    { // 将答案送给 beans, 下面的标签只有提交了相应的表单才被执行:

    %>

        <jsp:setProperty name="test" property="selection" />

    <%}

```

```

%>

<%--通过 beans 得到分数--%>

<P> 您现在的得分是:

<% if(!(s.equals(''))))

{
%>

    <jsp:getProperty name= "test" property="score" />

<%}

%>

</BODY>

</HTML>

```

6.6.9 日期 beans

使用 Java 的 Calendar 类制作 beans，用来显示年、月、日、星期、小时、分钟、秒等时间。

JSPCalendar.java:

```

import java.util.*;

public class JSPCalendar

{
    Calendar calendar = null;

    int year,dayOfMonth,dayOfYear,weekOfYear,

    weekOfMonth,dayOfWeek,hour,minute,second;

    String day,date,time;

    public JSPCalendar()

    {calendar = Calendar.getInstance();

```

```

    Date time = new Date();

    calendar.setTime(time);

}

// 获取年份:

public int getYear()

{return calendar.get(Calendar.YEAR);

}

// 获取月, 进行格式处理:

public String getMonth() {

int m=1+calendar.get(Calendar.MONTH);

String months[]={ "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12" };

if (m>12)

    return "0";

return months[m - 1];

}

// 获取星期几,进行格式处理:

public String getDay()

{ int n =getDayOfWeek();

String days[]={ "    日","一", "二", "三","四", "五", "六"};

if(n>7)

    return "    星期? ";

return days[n];

}

```

// 获取：年、月、日

public String getDate()

{ return getYear()+ "/" + getMonth()+"/"+getDayOfMonth();

}

public String getTime() // 获取：时：分：秒

{return getHour() + ":" + getMinute() + ":" + getSecond();

}

public int getDayOfMonth() // 获取当前时间是一月中的哪一天

{return calendar.get(Calendar.DAY_OF_MONTH);

}

public int getDayOfYear() // 获取当前时间是一年中的哪一天

{return calendar.get(Calendar.DAY_OF_YEAR);

}

public int getWeekOfYear() // 获取当前时间是一年中的哪个星期

{return calendar.get(Calendar.WEEK_OF_YEAR);

}

public int getWeekOfMonth() // 获取当前时间是一年中的哪个星期

{ return calendar.get(Calendar.WEEK_OF_MONTH);

}

public int getDayOfWeek() // 获取当前时间是一周中的哪一天

{return calendar.get(Calendar.DAY_OF_WEEK)-1;

}

public int getHour() // 获取小时


```

    {return calendar.get(Calendar.HOUR_OF_DAY);

}

public int getMinute() //    获取分钟

{return calendar.get(Calendar.MINUTE);

}

public int getSecond() //    获取秒

{return calendar.get(Calendar.SECOND);

}

}

```

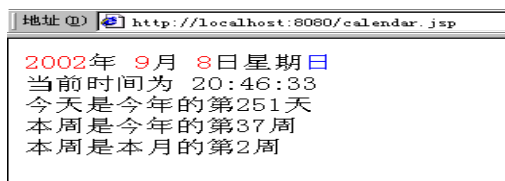


图 6.33 显示时间

显示日历时间的页面(效果如图 6.33 所示)

calendar.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="JSPCalendar" %>

<HTML>

<BODY>

<jsp:useBean id="clock" class="JSPCalendar" scope="page" >

```

```

</jsp:useBean>

<td align="center">

<font color="red"><jsp:getProperty name="clock" property="year"/></font>年
<font color="red"><jsp:getProperty name="clock" property="month"/></font>月
<font color="red"><jsp:getProperty name="clock"
property="dayOfMonth"/></font>日
    星期          <font color="blue"><jsp:getProperty name="clock"
property="day"/></b></font>

</td>

<BR>当前时间为<jsp:getProperty name="clock" property="time"/>
<BR>今天是今年的第<jsp:getProperty name="clock" property="dayOfYear"/>天
<BR>本周是今年的第<jsp:getProperty name="clock" property="weekOfYear"/>周
<BR>本周是本月的第<jsp:getProperty name="clock" property="weekOfMonth"/>
周

</BODY>

</HTML>

```

6.6.10 分页显示记录 beans

我们使用一个 **beans** 来实现数据库的记录的分页显示。

假设总记录数为 **m**，每页显示数量是 **n**，那么总页数的计算公式是：

- l 如果 **m** 除以 **n** 的余数大于 0，总页数等于 **m** 除以 **n** 的商加 1；
- l 如果 **m** 除以 **n** 的余数等于 0，总页数等于 **m** 除以 **n** 的商。

即

总页数=(m%n)==0?(m/n):(m/n+1);

如果准备显示第 p 页的内容，应当把游标移动到第(p-1)*n+1 条记录处。

分页 beans

PageNumber.java:

```
public class PageNumber
```

```
{ int rowCount=1, // 总的记录数。
```

```
    pageSize=1, // 每页显示的记录数。
```

```
    showPage=1, // 设置欲显示的页码数。
```

```
    pageCount=1; // 分页之后的总页数。
```

```
public void setRowCount(int n)
```

```
    { rowCount=n;
```

```
}
```

```
public int getRowCount()
```

```
    { return rowCount;
```

```
}
```

```
public void setPageCount(int r,int p)
```

```
    { rowCount=r;
```

```
        pageSize=p;
```

```
        int
```

```
n=(rowCount%pageSize)==0?(rowCount/pageSize):(rowCount/pageSize+1) ;
```

```
        pageCount=n;
```

```

    }

    public int getPageCount()

    {return pageCount;

    }

    public void setShowPage(int n)

    {showPage=n;

    }

    public int getShowPage()

    { return showPage;

    }

    public void setPageSize(int n)

    { pageSize=n;

    }

    public int getPageSize()

    { return pageSize;

    }

}

```

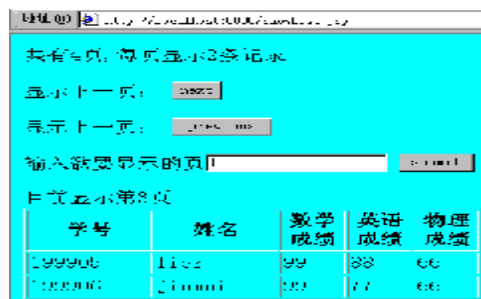


图 6.34 分页显示数据

分页显示记录的页面(效果如图 6.34 所示)

showList.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="PageNumber" %>

<%@ page import="java.io.*" %>

<jsp:useBean id="handlePage" class="PageNumber" scope="session" >

</jsp:useBean>

<HTML>

<BODY bgcolor=cyan><Size=1>

<%! // 声明一个共享的连接对象:

    Connection con=null;

    // 显示数据库记录的方法:

    public void showList(ResultSet rs,javax.servlet.jsp.JspWriter out, int n)

    {try

    {

        out.print("<Table Border>");

        out.print("<TR>");

        out.print("<TH width=100>"+ "        学号");
```

```

        out.print("<TH width=100>"+ "        姓名");
        out.print("<TH width=50>"+ "        数学成绩");
        out.print("<TH width=50>"+ "        英语成绩");
        out.print("<TH width=50>"+ "        物理成绩");

        out.print("</TR>");

        for(int i=1;i<=n;i++)

        {   out.print("<TR>");

            String number=rs.getString(1);

            out.print("<TD >"+number+"</TD>");

            String name=rs.getString(2);

            out.print("<TD >"+name+"</TD>");

            int math=rs.getInt("        数学成绩");

            out.print("<TD >"+math+"</TD>");

            int english=rs.getInt("        英语成绩");

            out.print("<TD >"+english+"</TD>");

            int physics=rs.getInt("        物理成绩");

            out.print("<TD >"+physics+"</TD>");

            out.print("</TR>");

            rs.next();

        }

        out.print("</Table>");

    }

    catch(Exception e1) {}

```

```

    }

    %>

<% Statement sql=null;

    ResultSet rs=null;

    int rowCount=0; // 总的记录数。

    // 第一个客户负责建立连接对象：

    if(con==null)

        { try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            }

            catch(ClassNotFoundException e)

                {out.print(e);

                }

            try

                {con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

                sql=

con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);

                // 返回可滚动的结果集：

                rs=sql.executeQuery("SELECT * FROM students");

                // 将游标移动到最后一行：

                rs.last();

                // 获取最后一行的行号：

```

```

    int number=rs.getRow();

    //      获取记录数:

    rowCount=number;

    //      设置每页显示的记录数:

    handlePage.setPageSize(2);

    //      计算总页数:

    handlePage.setPageCount(rowCount,handlePage.getPageSize());

    out.print("      共有"+handlePage.getPageCount()+"页,");

    out.print("      每页显示"+handlePage.getPageSize()+"条记录");

}

catch(SQLException e)

{out.print(e);

}

}

//  其它客户通过同步块使用这个连接:

else

{ synchronized(con)

{ try { sql=

con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);

//      返回可滚动的结果集:

rs=sql.executeQuery("SELECT * FROM students");

```



```

//      将游标移动到最后一行:

    rs.last();

//      获取最后一行的行号:

    int number=rs.getRow();

//      获取记录数:

    rowCount=number;

//      设置每页显示的记录数:

    handlePage.setPageSize(2);

//      计算总页数:

    handlePage.setPageCount(rowCount,handlePage.getPageSize());

    out.print("      共有"+handlePage.getPageCount()+"页,");

    out.print("      每页显示"+handlePage.getPageSize()+"条记录");

}

catch(SQLException e)

    {out.print(e);

    }

}

}

%>

<%-- 选择显示某页的表单 --%>

<% String str=response.encodeRedirectURL("showList.jsp");

%>

<Form action="<%=str%>" method="post" >

```

显示下一页: <Input Type="hidden" name="a" value="next">

<Input type=submit value="next">

</FORM>

<Form action="<%=str%>" method="post" >

显示上一页: <Input Type="hidden" name="a" value="previous">

<Input type="submit" value="previous">

</FORM>

<Form action="<%=str%>" method="post" >

输入欲要显示的页<Input type="text" name="a" value="1">

<Input type="submit" value="submit">

</FORM>

<% // 获取表单提交的信息:

```
String s=request.getParameter("a");
```

```
if(s==null)
```

```
{s="1";
```

```
}
```

```
if(s.equals("next"))
```

```
{ int n=handlePage.getShowPage(); // 获取目前的页数。
```

```
n=(n+1); // 将页数增 1。
```

```
if(n>handlePage.getPageCount())
```

```
{ n=1;
```

```
}
```

```
handlePage.setShowPage(n); // 显示该页
```

```

        out.print("        目前显示第"+handlePage.getShowPage()+"页");
        //        将游标移到:
        rs.absolute((n-1)*handlePage.getPageSize()+1);
        //        显示第该页的内容:
        showList(rs,out,handlePage.getPageSize());
    }

else if(s.equals("previous"))

    { int n=handlePage.getShowPage(); //        获取目前的页数。
        n=(n-1); //        将页数减 1。
        if(n<=0)
            { n=handlePage.getPageCount();
            }

        handlePage.setShowPage(n); //        显示该页
        out.print("        目前显示第"+handlePage.getShowPage()+"页");
        //        将游标移到:
        rs.absolute((n-1)*handlePage.getPageSize()+1);
        //        显示第该页的内容:
        showList(rs,out,handlePage.getPageSize());
    }

else

    { int m=Integer.parseInt(s);
        handlePage.setShowPage(m);
        out.print("        目前显示第"+handlePage.getShowPage()+"页");
    }

```

```
int n=handlePage.getShowPage();

//      将游标移到 (n-1)*handlePage.getPageSize()+1;

rs.absolute((n-1)*handlePage.getPageSize()+1);

//      显示该页的内容:

showList(rs,out,handlePage.getPageSize());

}

%>

</BODY>

</HTML>
```

第7章 基于会员制的网络交友

这一章讲述如何用 **JSP** 技术建立一个简单的网络交友系统，我们将采用 **Sun** 公司倡导的 **JSP+Javabeans** 模式。

7.1 系统设计

会员注册：新会员填写表单，包括会员名、**Email** 地址等信息。如果输入的会员名已经被其他用户注册使用，系统提示新用户更改自己的会员名。

会员登录：输入会员名、密码。如果用户输入的会员名或密码有错误，系统将显示错误信息；如果登录成功，就将一个成功登录的信息赋值给用户，同时用户被链接到“浏览其他会员”页面。

浏览会员：成功登录的会员可以分页浏览其他会员，如果用户直接进入该页面或没有成功登录就进入该页面，将被链接到“会员登录”页面。

查看会员信息：成功登录的会员可以在该页面输入要查阅的会员名，然后显示该会员的详细信息。如果用户直接进入该页面或没有成功登录就进入该页面，将被链接到“会员登录”页面。

留言板：成功登录的会员可以在该页面留言，如果直接进入该页面或没有成功登录就进入该页面，将被链接到“会员登录”页面。

修改密码：成功登录的会员可以在该页面修改自己的登录密码，如果用户直接进入该页面或没有成功登录就进入该页面，将被链接到“会员登录”页面。

修改注册信息：成功登录的会员可以在该页面修改自己的注册信息，比如联系电话、通信地址等，如果用户直接进入该页面或没有成功登录就进入该页面，将被链接到“会员登录”页面。

7.2 数据库设计及连接

我们采用 JDBC-ODBC 桥接器方式访问数据库。我们用 Access 建立一个数据库 **friend.mdb**，并将该数据库设置成一个名字是 **friend** 的数据源。该库有如下的表：

(1)新会员注册信息表：**member**。

会员的注册信息存入数据库 **friend.mdb** 的 **member** 表中，**member** 表的结构如图 7.1。

(2)会员的公共留言表：**wordpad**。

会员的公共留言存入数据库 **friend.mdb** 的 **wordpad** 表中，每个会员都可以浏览该表中的所有留言。**wordpad** 表的结构如图 7.2。

(3)会员的私人留言表：**secretwordpad**。

会员的私人留言存入数据库 **friend.dbm** 的 **secretwordpad** 表中，会员在该表中查找自己的私人留言。**secretwordpad** 表的结构如图 7.3。

(4) 三个表之间的关系如图 7.4。

字段名称	数据类型	说明
logname	文本	会员的用户名，不允许空值。
password	文本	密码，不允许空值。
sex	文本	性别，不允许空值。
age	文本	年龄，不允许空值。
phone	文本	电话，允许空值。
email	文本	电子邮件，允许空值。
address	文本	地址，允许空值。
message	备注	个人介绍，允许空值

字段属性

图 7.1 会员注册表

wordpad: 表			
	字段名称	数据类型	说明
	logname	文本	会员名
	public	备注	该会员的公共留言
字段属性			

图 7.2 公共留言表表

secretwordpad: 表			
	字段名称	数据类型	说明
	logname	文本	会员名字
	message	备注	其他会员发来的私人留言
	time	文本	留言时间
字段属性			

图 7.3 私人留言表

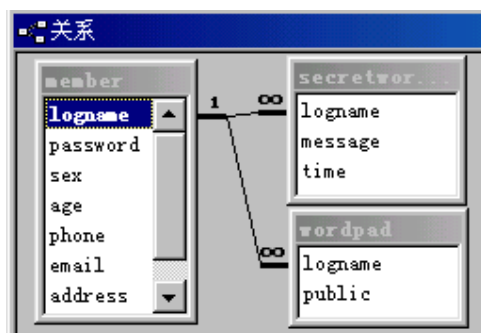


图 7.4 表之间的相互关系

7.3 页面管理

所有的页面将包括一个导航条，该导航条由注册、登录、浏览会员、查看会员、留言板、修改密码、修改个人信息组成。为了减少代码的编写，其它页面通过使用 JSP 的`<%@include>`标签将导航条文件：`head.txt` 嵌入自己的页面

`head.txt`:

```
<table align="center" border="0" width="790" height="12" bgcolor=cyan
cellspacing="0">

  <tr>

    <td width="100%">

      <a href="<%=response.encodeURL("showMember.jsp")%>">浏览会员</a> |
      <a href="<%=response.encodeURL("register.jsp")%>">会员注册</a> |
      <a href="<%=response.encodeURL("login.jsp")%>">会员登录  </a> |
      <a href="<%=response.encodeURL("find.jsp")%>">查找会员  </a> |
      <a href="<%=response.encodeURL("message.jsp")%>">留言板</a>|
      <a href="<%=response.encodeURL("publicMessage.jsp")%>">查看公共留言</a>|
      <a href="<%=response.encodeURL("secretMessage.jsp")%>">查看私人留言</a>|
      <a href="<%=response.encodeURL("modifyPassword.jsp")%>">修改密码 </a> |
      <a href="<%=response.encodeURL("modifyMessage.jsp")%>">修改个人信息 </a>|

    </td>
```



```

</tr>

</table></table>

```

我们使用的 beans 的包名都是 tom.jiafei，因此，我们将 beans 的类文件存放在 JSP 引擎的 classes\tom\jiafei 目录中。也就是说，我们需要在 classes 下再建立一个目录结构：tom\jiafei，将创建 beans 的类文件存放在 jiafei 文件夹中。

另外，我们自己配置一个 web 服务目录，将 D:\test 作为服务目录，并让用户使用 /friend 虚拟目录访问。首先用记事本打开主配置文件 server.xml（该文件在 Tomcat\Jakarta-tomcat-4.0\conf 文件下），找到出现

```

<!-- Tomcat Root Context -->

<!--
  <Context path="" docBase="ROOT" debug="0"/>
-->

<!-- Tomcat Examples Context -->

<Context path="/examples" docBase="examples" debug="0"
  reloadable="true">
  <Logger className="org.apache.catalina.logger.FileLogger"
    prefix="localhost_examples_log." suffix=".txt"
    timestamp="true"/>
    ...      ...      ...
    ...      ...      ...

```

</Context>

(在这里加入您的服务目录)

</Host>

的部分。然后在</Context>和 </Host> 之间加入:

```
<Context path="/friend" docBase="d:/test" debug="0" reloadable="true">
```

```
</Context>
```

所有的 JSP 页面以及导航条文件存放在 D:/test 目录中。

7.4 各个页面的设计

对使用 beans 的页面, 我们首先给出 beans 的代码, 然后阐述页面的设计过程。

我们尽量减少不必要的 HTML 标签, 重点体现 JSP 的功能。主页由导航条和一个欢迎语组成。



图 7.5 主页

主页面(效果如图 7.5 所示)

welcomeFriend.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor =yellow>

<%@ include file="head.txt" %>

<H1>

<CENTER> 欢迎网上结交朋友</CENTER>

</BODY>

</HTML>
```

7.4.1 会员注册

当新会员注册时,该模块要求用户必须输入会员名、密码、年龄等信息,否则不允许注册。用户的注册信息被存入数据库 **friend.dbm** 的 **member** 表中。

数据库的插入记录操作由 **Register** 创建的 **beans** 负责。为了防止由于插入空字段而发生 SQL 语句异常,该 **beans** 需含有判断注册者是否提交了空字段的语句。将下述的 **Register.java** 文件编译通过,并将生成的字节码文件 **Register.class** 存放到 **classes\tom\jiafei** 目录中。

注册页面使用的 beans

Register.java:

```
package tom.jiafei;

import java.sql.*;
```

```

public class Register
{
    String logname="",
        password="",
        sex="",age="",
        email="",
        phone="",
        address="",
        message="";

    String backNews;// 用来返回注册是否成功。

    Connection con;

    Statement sql;

    ResultSet rs;

    public Register()
    {
        try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");// 加载桥接器。
        }

        catch(ClassNotFoundException e){}
    }

    // 设置属性值、获取属性值的方法：

    public void setLogname(String name)
    {
        logname=name;
    }

    public String getLogname()
    {
        return logname;
    }
}

```

```
}  
  
public void setAge(String n)  
{ age=n;  
}  
  
public String getAge()  
{return age;  
}  
  
public void setSex(String s)  
{ sex=s;  
}  
  
public String getSex()  
{return sex;  
}  
  
public void setPassword(String pw)  
{ password=pw;  
}  
  
public String getPassword()  
{return password;  
}  
  
public void setEmail(String em)  
{ email=em;  
}  
  
public String getEmail()
```

```
{return email;

}

public void setPhone(String ph)

{ phone=ph;

}

public String getPhone()

{return phone;

}

public void setAddress(String ad)

{ address=ad;

}

public String getAddress()

{return address;

}

public String getMessage()

{return message;

}

public void setMessage(String m)

{ message=m;

}

public String getBackNews()

{return backNews;

}
```

```

public void setBackNews(String s)

{backNews=s;

}

// 添加记录到数据库的 member 表:

public void addItem()

{try{

    con=DriverManager.getConnection("jdbc:odbc:friend","","");

    sql=con.createStatement();

    if(phone.length()==0) //    如果用户没有提供电话

        {phone="        无";

        }

    if(email.length()==0) //    如果用户没有提供 Email

        {email="        无";

        }

    if(address.length()==0) //    如果用户没有提供地址

        {address="        无";

        }

    if(message.length()==0) //    如果用户没有提供信息

        {message="        无";

        }

    String s=

        ""+logname+""+", "+""+password+""+", "+""+sex+""+", "+

        ""+age+""+", "+ ""+phone+""+", "+""+email+""+", "+

```

```
""+address+""+", "+""+message+"";
```

```
String condition="INSERT INTO member VALUES"+"("+s+")";
```

```
sql.executeUpdate(condition);
```

```
backNews="    注册成功了";
```

```
con.close();
```

```
}
```

```
catch(SQLException e)
```

{// 如果用户使用 member 表中已经存在的名字, 或使用了空字段值, 就会发生 SQL 异常

```
backNews="    你还没有注册, 或该用户已经存在, 请你更换一个名字";
```

```
}
```

```
}
```

```
}
```

地址 http://localhost:8080/friend/register.jsp

[浏览会员](#) | [会员注册](#) | [会员登录](#) | [查找会员](#) | [修改密码](#) | [修改个人信息](#)

输入您的信息, 会员名字不允许含有空格, 带*号项必须填写, :

会员名称 * 设置密码 *

性别 (*) ☒ 男 ☐ 女

会员年龄 * 电子邮件

联系电话 . 通信地址

输入您的简介和交友要求

王小闯 注册成功了

图 7.6 会员注册

会员注册页面（效果如图 7.6 所示）

register.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import="tom.jiafei.Register" %>
```

```
<%! //处理字符串的方法:
```

```
public String codeString(String s)
```

```
{ String str=s;
```

```
try{byte b[]=str.getBytes("ISO-8859-1");
```

```
str=new String(b);
```

```
return str;
```

```
}
```

```
catch(Exception e)
```

```
{ return str;
```

```
}
```

```
}
```

```
%>
```

```
<HTML>
```

```
<BODY bgcolor=yellow><Font size=1>
```

```

<%@ include file="head.txt"%>

<Font size=1>

<BR>输入您的信息，会员名字不允许含有空格，带*号项必须填写，:

<%String str=response.encodeURL("register.jsp");

%>

<FORM action="<%=str%>" Method="post">

<BR>会员名称<Input type=text name="logname">*

    设置密码<Input type=password name="password">*

<BR>性别（*）<Input type=radio name="sex" checked="o" value=" 男">男

    <Input type=radio name="sex" value="          女">女

<BR>会员年龄<Input type=text name="age" value="0">*

    电子邮件<Input type=text name="email">

<BR>联系电话<Input type=text name="phone">.

    通信地址<Input type=text name="address">

<BR>输入您的简介和交友要求

<BR><TextArea name="message" Rows="4" Cols="57">

    </TextArea>

<BR><Input type=submit name="g" value="提交">

</Form>

<jsp:useBean id="memberlogin" class="tom.jiafei.Register" scope="request" >

</jsp:useBean>

<%

    String logname="",sex="",age="", password="",

```

```

        email="",phone="",address="",message="";

int n=0; // 用来验证年龄的变量。

// 提交信息后，进行注册操作：

if(!(session.isNew()))

{ logname=request.getParameter("logname");

    if(logname==null)

        {logname="";

        }

    logname=codeString(logname);

    // 判断名字是否含有空格：

    int space=logname.indexOf(" ");

    if(space!=-1)

        {response.sendRedirect("register.jsp");

        }

    password=request.getParameter("password");

    if(password==null)

        {password="";

        }

    password=codeString(password);

    sex=request.getParameter("sex");

    if(sex==null)

        {sex="";

        }

```

```
sex=codeString(sex);

age=request.getParameter("age");

if(age==null)

    {age="0";

    }

age=codeString(age);

try{ n=Integer.parseInt(age);

    }

catch(NumberFormatException e)

    {n=0;

    }

email=request.getParameter("email");

    if(email==null)

        {email="";

        }

email=codeString(email);

phone=request.getParameter("phone");

    if(phone==null)

        {phone="";

        }

phone=codeString(phone);

address=request.getParameter("address");

    if(address==null)
```

```

        {address="";
    }

    address=codeString(address);

    message=request.getParameter("message");

    if(message==null)

        {message="";

        }

    message=codeString(message);
}

%>

<% // 检查用户是否按要求填写了必要的信息:用户名、年龄、密码,

    // 为了以后处理汉字方便, 我们采用了第 1 种方式初始化 beans

    boolean

b=!(logname.equals(""))&&!(password.equals(""))&&(n<=150)&&(n>=0);

    if(b)

        { out.print(logname);%>

            <jsp:setProperty name="memberlogin" property="logname"

value="<%=logname%>" />

            <jsp:setProperty name="memberlogin" property="password"

value="<%=password%>" />

            <jsp:setProperty name="memberlogin" property="sex"

value="<%=sex%>" />

            <jsp:setProperty name="memberlogin" property="age"

```

```
value="<%=age%>" />
```

```
    <jsp:setProperty name="memberlogin" property="email"
```

```
value="<%=email%>" />
```

```
    <jsp:setProperty name="memberlogin" property="phone"
```

```
value="<%=phone%>" />
```

```
    <jsp:setProperty name="memberlogin" property="address"
```

```
value="<%=address%>" />
```

```
    <jsp:setProperty name="memberlogin" property="message"
```

```
value="<%=message%>" />
```

```
    <%
```

```
        memberlogin.addItem();
```

```
    }
```

```
else
```

```
    {out.print("  您还没有填写信息，或信息填写不完整、年龄或名字不正确");
```

```
    }
```

```
%>
```

```
<% // 返回注册是否成功的信息
```

```
    if(!(session.isNew()))
```

```
    {
```

```
%>
```

```
    <jsp:getProperty name="memberlogin" property="backNews" />
```

```
<%
```

```
    }
```

%>

</Body>

</HTML>

7.4.2 会员登录

用户可在该页面输入自己的会员名和密码，系统将对会员名和密码进行验证，如果名字和密码都正确将被链接到“浏览会员”页面，否则提示用户输入的密码或用户名不正确。该页面使用一个 beans 负责查询 member 表来验证登录者的身份

登录页面使用的 beans

Login.java:

```
package tom.jiafei;  
  
import java.sql.*;  
  
public class Login  
{ String logname,  
  
    password,  
  
    success="false",  
  
    message=""; //      用来返回登录是否成功的消息。  
  
Connection con;  
  
Statement sql;  
  
ResultSet rs;  
  
public Login()  
  
{ //      加载桥接器:  
  
    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```

    }

    catch(ClassNotFoundException e){}

}

// 设置属性值、获取属性值的方法：
public void setLogname(String name)

{ logname=name;

}

public String getLogname()

{return logname;

}

public void setPassword(String pw)

{ password=pw;

}

public String getPassword()

{return password;

}

public String getSuccess()

{return success;

}

// 查询数据库的 member 表：
public String getMessage()

{try{

    con=DriverManager.getConnection("jdbc:odbc:friend","","");

```



```

sql=con.createStatement();

String condition=

"SELECT * FROM member WHERE logname = '"+logname+"'";

rs=sql.executeQuery(condition);

int rowcount=0;

String ps=null;

while(rs.next())

{
    rowcount++;

    logname=rs.getString("logname");

    ps=rs.getString("password");

}

if((rowcount==1)&&(password.equals(ps)))

{
    message="ok";

    success="ok";

}

else

{
    message="    输入的用户名或密码不正确";

    success="false";

}

con.close();

return message;

}

catch(SQLException e)

```

```
{ message="    输入的用户名或密码不正确";  
  
  success="false";  
  
  return message;  
  
}  
  
}  
  
}
```

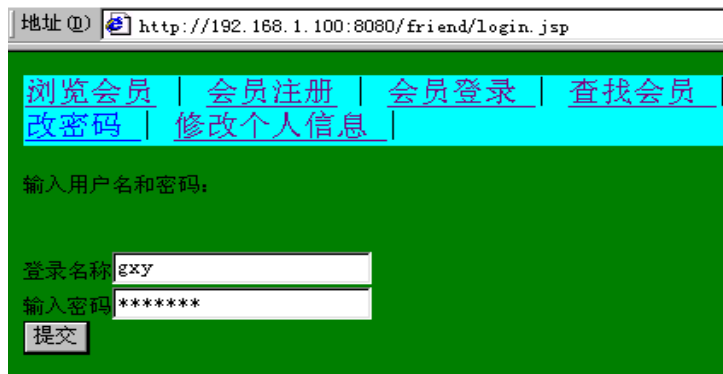


图 7.7 会员登录

会员登录页面（效果如图 7.7 所示）

login.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import="tom.jiafei.Login" %>
```

```
<%! //处理字符串的方法:
```

```
public String codeString(String s)
```

```
{ String str=s;
```

```
try{byte b[]=str.getBytes("ISO-8859-1");
```

```
str=new String(b);
```

```
return str;
```

```
}
```

```
catch(Exception e)
```

```
{ return str;
```

```
}
```

```
}
```

```
%>
```

```
<HTML>
```

```
<BODY bgcolor=cyan ><Font size=1>
```

```
<%@ include file="head.txt" %>
```

```
<Font size=1>
```

```
<P>输入用户名和密码:
```

```
<%String str=response.encodeURL("login.jsp");
```

```
%>
```

```
<FORM action="<%=str%>" Method="post">
```

```
<BR>登录名称<Input type=text name="logname">
```

```

<BR>输入密码<Input type=password name="password">

<BR><Input type=submit name="g" value="提交">

</Form>

<jsp:useBean id="login" class="tom.jiafei.Login" scope="session" >
</jsp:useBean>

<% // 提交信息后，验证信息是否正确：

String message="",
    logname="",
    password="";

if(!(session.isNew()))
{logname=request.getParameter("logname");

    if(logname==null)
        {logname="";

        }

    logname=codeString(logname);

    password=request.getParameter("password");

    if(password==null)
        {password="";

        }

    password=codeString(password);

}

%>

<%

```

```

    if(!(logname.equals("")))
    {
        %>

        <jsp:setProperty name= "login" property="logname"
value="<%=logname%>" />

        <jsp:setProperty name= "login" property="password"
value="<%=password%>" />

        <%

            message=login.getMessage(); //    获取返回的验证信息。

            if(message==null)

                {message="";

                }

            }

        %>

        <% if(!(session.isNew()))

            { if(message.equals("ok"))

                {String meb=response.encodeURL("showMember.jsp");

                response.sendRedirect(meb);

                }

            else

                {out.print(message);

                }

            }

        }

```

```
%>

</Body>

</HTML>
```

7.4.3 浏览会员

该模块负责分页显示注册会员的基本信息，包括会员名和性别。当浏览会员基本信息时，该模块在每个会员的后面显示一个表单，用户点击该表单可将所要查看的会员名字提交到会员详细信息页面：**find.jsp**，然后在该页面查看该会员的详细信息。该页面使用一个 **beans** 负责完成信息的分页显示。

浏览会员页面使用的 beans

PageNumber.java:

```
package tom.jiafei;

public class PageNumber
{ int rowCount=1, //    总的记录数。

    pageSize=1, //    每页显示的记录数。

    showPage=1, //    设置欲显示的页码数。

    pageCount=1; //    分页之后的总页数。

    public void setRowCount(int n)
    { rowCount=n;
    }

    public int getRowCount()
    { return rowCount;
    }
}
```

```

public void setPageCount(int r,int p)

    { rowCount=r;

      pageSize=p;

      int

n=(rowCount%pageSize)==0?(rowCount/pageSize):(rowCount/pageSize+1) ;

      pageCount=n;

    }

public int getPageCount()

    {return pageCount;

    }

public void setShowPage(int n)

    {showPage=n;

    }

public int getShowPage()

    { return showPage;

    }

public void setPageSize(int n)

    { pageSize=n;

    }

public int getPageSize()

    { return pageSize;

    }

}

```

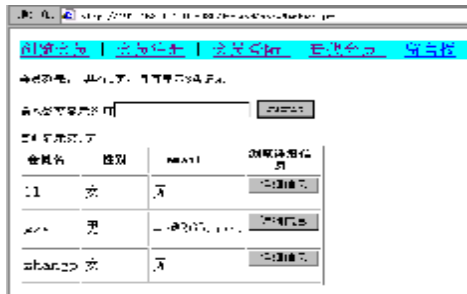


图 7.8 浏览会员基本信息

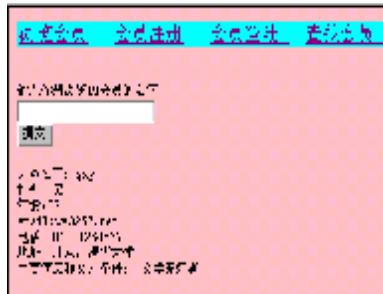


图 7.9 查看会员详细信息

浏览会员页面（效果如图 7.8、7.9 所示）

showMemeber.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="tom.jiafei.Login" %>

<%@ page import="tom.jiafei.PageNumber" %>

<%@ page import="java.io.*" %>

<jsp:useBean id="handlePage" class="tom.jiafei.PageNumber" scope="session" >

</jsp:useBean>

<jsp:useBean id="login" class="tom.jiafei.Login" scope="session" >

</jsp:useBean>

<% //如果客户直接进入该页面将被转向登录页面。
```



```

if(session.isNew())

{response.sendRedirect("login.jsp");

}

// 如果没有成功登录将被转向登录页面

String success=login.getSuccess();

    if(success==null)

        {success="";

        }

    if(!(success.equals("ok")))

        {response.sendRedirect("login.jsp");

        }

%>

<HTML>

<BODY ><Font size=1>

<%@ include file="head.txt" %>

<P> 会员列表:

<%! // 声明一个共享的连接对象:

    Connection con=null;

    // 显示数据库记录的方法:

    public void showList(ResultSet rs,javax.servlet.jsp.JspWriter out,int n,String

find)

    {try

    {

```

```

out.print("<Table Border>");

out.print("<TR>");

out.print("<TH width=50>"+ "<Font size=1>"+ "      会员名"+ "</FONT>");

out.print("<TH width=70>"+ "<Font size=1>"+ "      性别"+ "</FONT>");

out.print("<TH width=70>"+ "<Font size=1>"+ "email"+ "</FONT>");

out.print("<TH width=70>"+ "<Font size=1>"+ "      浏览    详细    信  息
"+"</FONT>");

out.print("</TR>");

for(int i=1;i<=n;i++)
{
String logname=rs.getString("logname");

String email=rs.getString("email");

out.print("<TR>");

out.print("<TD >"+logname+"</TD>");

out.print("<TD >"+rs.getString("sex")+"</TD>");

out.print("<TD >"+email+"</TD>");

//      在每个会员的后面显示一个表单，该表单将内容提交到 find.jsp,
//      以便查看该会员的详细信息：

String s1="<Form action="+find+" method=get>";

String s2="<input type=hidden name=logname value="+logname+">";

String s3="<input type=submit value=      详细信息></FORM> ";

String s=s1+s2+s3;

out.print("<TD >"+s+"</TD>");

```

```

        out.print("</TR>");

        rs.next();

    }

    out.print("</Table>");

}

catch(Exception e1) {}

}

%>

<%

Statement sql=null;

ResultSet rs=null;

int rowCount=0; //    总的记录数。

String logname="";

//    第一个客户负责 建立连接对象：

if(con==null)

{ try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

    catch(ClassNotFoundException e)

    {out.print(e);

    }

try

{con=DriverManager.getConnection("jdbc:odbc:friend","", "");

sql=

```

```
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
```

```
rs=sql.executeQuery("SELECT * FROM member"); // 返回可滚动的结果集。
```

```
rs.last(); // 将游标移动到最后一行。
```

```
int number=rs.getRow();// 获取最后一行的行号。
```

```
rowCount=number; // 获取记录数。
```

```
handlePage.setPageSize(3); // 设置每页显示的记录数。
```

```
handlePage.setShowPage(1); // 设置欲显示的页码数。
```

```
handlePage.setPageCount(rowCount,handlePage.getPageSize());// 计算总页数。
```

```
out.print(" 共有"+handlePage.getPageCount()+"页， ");
```

```
out.print(" 每页显示"+ handlePage.getPageSize()+"条记录");
```

```
}
```

```
catch(SQLException e)
```

```
{out.print(e);
```

```
}
```

```
}
```

```
// 其它客户通过同步块使用这个连接：
```

```
else
```

```
{ synchronized(con)
```

```
{ try { sql=
```

```
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
```

```
rs=sql.executeQuery("SELECT * FROM member"); // 返回可滚动的结果集。
```

```
rs.last(); // 将游标移动到最后一行。
```

```
int number=rs.getRow(); // 获取最后一行的行号。
```

```
rowCount=number; // 获取记录数。
```

```
handlePage.setPageSize(3); // 设置每页显示的记录数。
```

```
handlePage.setShowPage(1); // 设置欲显示的页码数。
```

```
handlePage.setPageCount(rowCount,handlePage.getPageSize()); // 计算总页数。
```

```
out.print(" 共有"+handlePage.getPageCount()+"页， ");
```

```
out.print(" 每页显示"+ handlePage.getPageSize()+"条记录");
```

```
}
```

```
catch(SQLException e)
```

```
{out.print(e);
```

```
}
```

```
}
```

```
}
```

```
%>
```

```
<%-- 选择显示某页的表单 --%>
```

```
<%String str=response.encodeURL("showMember.jsp");
```

```

String find=response.encodeURL("find.jsp");

%>

<Form action="<%=str%>" method="post" >

    输入欲要显示的页<Input type="text" name="ok" value="1">

    <Input type="submit" value="submit">

</FORM>

<% // 获取表单提交的信息:

    String s=request.getParameter("ok");

    if(s==null)

        {s="1";

        }

    int m=Integer.parseInt(s);

    handlePage.setShowPage(m);

    out.print("        目前显示第"+handlePage.getShowPage()+"页");

    int n=handlePage.getShowPage();

    //        将游标移到:

    rs.absolute((n-1)*handlePage.getPageSize()+1);

    showList(rs,out,handlePage.getPageSize(),find); //        显示该页的内容。

%>

</FONT>

</BODY>

</HTML>

```

7.4.4 查找会员

登录的会员可以在该页面查找其他会员，并浏览这个会员的详细信息。会员的信息中可能会含有 **HTML** 标记，甚至，一个会员的信息可能是一个 **HTML** 或 **JSP** 文档，为了能显示原始的 **HTML** 和 **JSP** 标记，该页面使用一个 **beans** 对信息进行处理。

查找会员页面使用的 beans

HandleMessage.java:

```
package tom.jiafei;

import java.io.*;

public class HandleMessage
{ String content=null;

    public void setContent(String s)

        {content=s;

        }

    // 获取属性 content 的值，为了能显示 HTML 或 JSP 源文件，需进行流的处理技术：

    public String getContent()

        { try{ StringReader in=new StringReader(content) ;// 指向字符串的字符流。

            PushbackReader push=new PushbackReader(in);

            StringBuffer stringbuffer=new StringBuffer();

            int c;

            char b[]=new char[1];

            while ( (c=push.read(b,0,1))!=-1)//      读取 1 个字符放入字符数组 b。

                { String s=new String(b);

                    if(s.equals("<"))      //      回压的条件
```

```

{ push.unread('&');

  push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
  stringBuffer.append(new String(b));

  push.unread('L');

  push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
  stringBuffer.append(new String(b));

  push.unread('T');

  push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
  stringBuffer.append(new String(b));

}

else if(s.equals(">"))    //          回压的条件

{ push.unread('&');

  push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
  stringBuffer.append(new String(b));

  push.unread('G');

  push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
  stringBuffer.append(new String(b));

  push.unread('T');

  push.read(b,0,1); //push          读出被回压的字符字节,放入数组 b.
  stringBuffer.append(new String(b));

}

else if(s.equals("\n"))

{ stringBuffer.append("<BR>");

```



```

    }

    else

        { stringBuffer.append(s);

        }

    }

    push.close();

    in.close();

    return content=new String(stringbuffer);

}

catch(IOException e)

    {return content=new String("    不能读取内容");

    }

}

}

```



图 7.10 查找会员

查找会员页面（效果如图 7.10 所示）

find.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="java.io.*" %>

<%@ page import="tom.jiafei.Login" %>

<%@ page import="tom.jiafei.HandleMessage" %>

<jsp:useBean id="login" class="tom.jiafei.Login" scope="session" >

</jsp:useBean>

<jsp:useBean id="handle" class="tom.jiafei.HandleMessage" scope="page" >

</jsp:useBean>

<%!//处理字符串的一个常用方法:

    public String getString(String s)

    { if(s==null) s="";

        try {byte a[]=s.getBytes("ISO-8859-1");

            s=new String(a);
```

```

    }

    catch(Exception e)

    { }

    return s;

}

%>

<% //如果客户直接进入该页面将被转向登录页面。

    if(session.isNew())

    {response.sendRedirect("login.jsp");

    }

    // 如果没有成功登录将被转向登录页面

    String success=login.getSuccess();

    if(success==null)

    {success="";

    }

    if(!(success.equals("ok")))

    {response.sendRedirect("login.jsp");

    }

%>

<% //获取本页面或 showMember 页面表单提交的会员 名字:

    String logname=request.getParameter("logname");

    if(logname==null)

    {logname="";

```

```

    }

    logname=getString(logname);

%>

<HTML>

<BODY bgcolor=pink ><Font size=1>

<%@ include file="head.txt" %>

<%String find=response.encodeURL("find.jsp");

%>

<FORM action="<%=find%>" Method="post">

<BR>输入你想浏览的会员的名字:

<BR><Input type=text name="logname" >

<BR><Input type=submit name="g" value="提交">

</FORM>

<%    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        }

        catch(ClassNotFoundException event){}

//    验证身份:

    Connection con=null;

    Statement sql=null;

    ResultSet rs=null;

    boolean modify=false;

    try{ con=DriverManager.getConnection("jdbc:odbc:friend","", "");

        sql=con.createStatement();

```

```

String condition="SELECT * FROM member WHERE logname =
"+"'''+logname+'''';

rs=sql.executeQuery(condition);

while(rs.next())

{ StringBuffer str=new StringBuffer();

String      会员名=rs.getString("logname"),

sex=rs.getString("sex"),

age=rs.getString("age"),

email=rs.getString("email"),

phone=rs.getString("phone"),

address=rs.getString("address"),

message=rs.getString("message");

str.append("      会员名字: "+会员名+"\n");

str.append("      性别: "+sex+"\n");

str.append("      年龄:"+age+"\n");

str.append("email:"+email+"\n");

str.append("      电话: "+phone+"\n");

str.append("      地址: "+address+"\n");

str.append("      主要简历和交友条件: "+message+"\n");

//      为了能显示会员信息中的原始 HTML 标记信息, 信息做回压流处理:

String content=new String(str);

handle.setContent(content);

out.print(handle.getContent());

```

```

    }

}

catch(SQLException e1)

    { out.print("<BR>    查找失败");

    }

%>

</FONT>

</BODY>

</HTML>

```

7.4.5 留言板

登录的用户可以在该页面进行公共留言和私人留言。如果选择公共留言，那么该会员的留言可以被所有的会员查看；如果选择私人留言，留言将送给某个特定的会员。这时必须要输入对方的会员名字才能留言，如果该名字不是注册的会员名，就不能实现留言。该模块由两个页面组成，一个留言主页面：**message.jsp**，会员在该页面输入留言，然后选择留言方式（公共留言或私人留言）后，提交给 **leaveword.jsp** 页面，该页面将留言写入数据库中的 **publicword** 表（公共留言）或 **secrecword** 表（私人留言）。

该模块需要两个 **beans**：**PublicWord.java**，**SecretWord.java**，一个负责公共留言，另一个负责私人留言。当把一个留言送给一个特定的会员时，要求有留言时间，我们要求留言的时间是唯一的，所以 **beans** 中负责留言的方法：**addItem** 声明为 **synchronized**，这样，会员留言的时间是唯一的，这个时间取从 **1970** 年到留言时所走过的毫秒数。将来可以根据留言时间删除这一留言。

负责公共留言的 beans

PublicWord.java:

```
package tom.jiafei;

import java.sql.*;

public class PublicWord
{String logname="",
    message="";

    String backNews;// 用来留言是否成功。

    Connection con;

    Statement sql;

    ResultSet rs;

    public PublicWord()
    { // 加载桥接器:

        try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            }

        catch(ClassNotFoundException e){}

    }

    // 设置属性值、获取属性值的方法:

    public void setLogname(String name)

    { logname=name;

    }

    public String getLogname()

    {return logname;

    }
```

```

public String getMessage()

{return message;

}

public void setMessage(String m)

{ message=m;

}

public String getBackNews()

{return backNews;

}

public void setBackNews(String s)

{backNews=s;

}

// 添加记录到数据库的 wordpad 表:

public void addItem()

{try{

    con=DriverManager.getConnection("jdbc:odbc:friend","", "");

    sql=con.createStatement();

    String s=""+logname+"",""+message+"";

    String condition="INSERT INTO wordpad VALUES"+"("+s+")";

    sql.executeUpdate(condition);

    backNews="    添加成功了";

    con.close();

}

```



```
catch(SQLException e)
```

```
{//    由于表 wordpad 和 member 表通过字段 logname 做了关联，所以如果输入的 logname
```

```
//    不对，就会出现异常
```

```
backNews="    你没有登录，不能留言";
```

```
}
```

```
}
```

```
}
```

负责私人留言的 beans

SecretWord.java:

```
package tom.jiafei;
```

```
import java.sql.*;
```

```
public class SecretWord
```

```
{String logname="",
```

```
time="", //    留言时间。
```

```
message="";
```

```
String backNews;//    返回留言是否成功的信息。
```

```
Connection con;
```

```
Statement sql;
```

```
ResultSet rs;
```

```
public SecretWord()
```

```
{ //    加载桥接器:
```

```

    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        }

    catch(ClassNotFoundException e){}

}

// 设置属性值、获取属性值的方法：

public void setLogname(String name)

{ logname=name;

}

public String getLogname()

{return logname;

}

public String getMessage()

{return message;

}

public void setMessage(String m)

{ message=m;

}

public String getBackNews()

{return backNews;

}

public void setBackNews(String s)

{backNews=s;

}

```

```
public void setTime(String t)
```

```
{time=t;
```

```
}
```

```
public String getTime()
```

```
{return time;
```

```
}
```

```
// 添加记录到数据库的 wordpad 表:
```

```
// 我们要求留言的时间是唯一的, 所以下面的方法声明为 synchronized
```

```
public synchronized void addItem()
```

```
{try{
```

```
    con=DriverManager.getConnection("jdbc:odbc:friend","","");
```

```
    sql=con.createStatement();
```

```
    String s=""+logname+"", "+"+message+"", "+"+time+"";
```

```
    String condition="INSERT INTO secretwordpad VALUES"+"("+s+")";
```

```
    sql.executeUpdate(condition);
```

```
    backNews="    添加成功了";
```

```
    con.close();
```

```
}
```

```
catch(SQLException e)
```

```
{//    由于表 wordpad 和 member 表通过字段 logname 做了关联, 所以如果输入  
对方的 logname
```

```
//    不存在, 就会出现异常
```

```
    backNews="    该会员不存在, 不能留言给他(她)";
```

```

    }

}

}

```

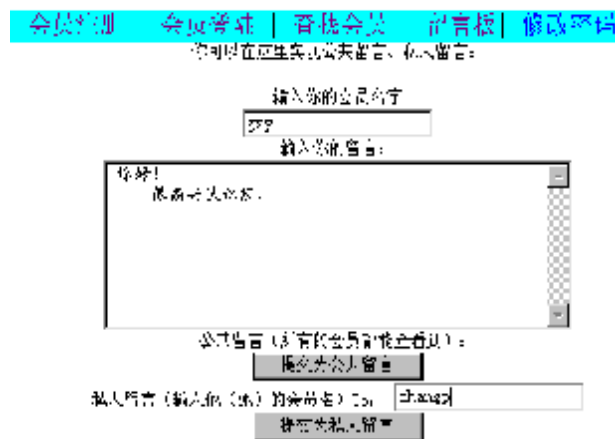


图 7.11 留言板

留言板的主页面（效果如图 7.11 所示）

message.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

```

```

<%@ page import="java.io.*" %>

<%@ page import="tom.jiafei.Login" %>

<jsp:useBean id="login" class="tom.jiafei.Login" scope="session" >

</jsp:useBean>

```

<%!//处理字符串的一个常用方法:

```

    public String getString(String s)

    { if(s==null) s="";

    try {byte a[]=s.getBytes("ISO-8859-1");

        s=new String(a);

    }

    catch(Exception e)

    {

    }

    return s;

}

```

```

%>

```

<% //如果客户直接进入该页面将被转向登录页面。

```

    if(session.isNew())

    {response.sendRedirect("login.jsp");

    }

```

// 如果没有成功登录将被转向登录页面

```

String success=login.getSuccess();

    if(success==null)

```

```

        {success="";
    }

    if(!(success.equals("ok")))

        {response.sendRedirect("login.jsp");

        }

    %>

```

<HTML>

<BODY >

<%@ include file="head.txt" %>

<CENTER>

你可以在这里实现公共留言、私人留言。

<%String str=response.encodeURL("leaveword.jsp");

%>

<FORM action="<%=str%>" method=post>

输 入 你 的 会 员 名 字
<Input Type=text name=logname

value=<%=login.getLogname()%>>

输入你的留言:

<TextArea name="message" Rows="8" Cols="50">

</TextArea>

公共留言（所有的会员都能查看到）:

<Input type=submit name="submit" value="提交为公共留言">

私人留言（输入他（她）的会员名） To:

<Input type =text name=person>

```

<BR><Input type=submit name="submit" value="提交为私人留言">

</FORM>

</CENTER>

</BODY>

</HTML>

```

进行留言操作的 JSP 页面

leaveword.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<jsp:useBean id="login" class="tom.jiafei.Login" scope="session" >

</jsp:useBean>

<jsp:useBean id="publicbean" class="tom.jiafei.PublicWord" scope="page" >

</jsp:useBean>

<jsp:useBean id="secretbean" class="tom.jiafei.SecretWord" scope="page" >

</jsp:useBean>

<%!//处理字符串的一个常用方法:

    public String getString(String s)

    { if(s==null) s="";

      try {byte a[]=s.getBytes("ISO-8859-1");

          s=new String(a);

      }

      catch(Exception e)

```

```

        { }

    return s;

}

%>

<HTML>

<BODY bgcolor=pink ><Font size=1>

<% //获取提交键的的串值:

    String s=request.getParameter("submit");

    s=getString(s);

    // 根据 s 的不同情况分开处理:

    if(s.equals(" 提交为公共留言"))

    { // 获取提交的留言:

        String ms=request.getParameter("message");

        ms=getString(ms);

        publicbean.setLogname(login.getLogname());

        publicbean.setMessage("'" + login.getLogname() + "    的留言: " + ms);

        // 留言:

        publicbean.addItem();

        out.print(publicbean.getBackNews());

    }

else if(s.equals(" 提交为私人留言"))

    { // 获取会员的名字:

        String name=request.getParameter("person");

```



```

        name=getString(name);

String ms=request.getParameter("message");

ms=getString(ms);

if(name.equals(""))

    {out.print("        您没有输入他（她）的名字，不能留言给人家");

    }

else

    { secretbean.setLogname(name);

        secretbean.setMessage("'" + login.getLogname() + "        留言给你:" + ms);

        //        留言时间：

        long n=System.currentTimeMillis();

        String time=String.valueOf(n);

        secretbean.setTime(time);

        //        留言：

        secretbean.addItem();

        out.print(secretbean.getBackNews());

    }

}

%>

</FONT>

</BODY>

</HTML>

```

7.4.6 查看公共留言

在该页面可以分页显示所有会员的留言。在这个页面中要使用前面已经使用过的一些 beans，**PageNumber.java**，负责分页显示数据；**HandleMessage.java**，负责处理原始的 HTML 和 JSP 信息。

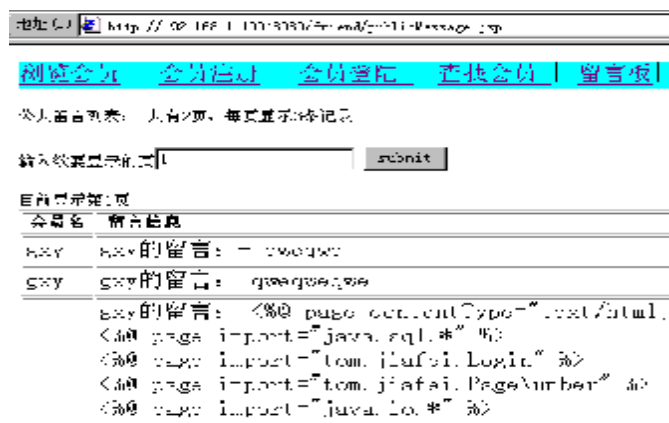


图 7.12 显示公共留言

查看公共留言页面（效果如图 7.12 所示）

publicMessage.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="tom.jiafei.Login" %>
```

```

<%@ page import="tom.jiafei.PageNumber" %>

<%@ page import="tom.jiafei.HandleMessage" %>

<jsp:useBean id="handlePage" class="tom.jiafei.PageNumber" scope="session" >

</jsp:useBean>

<jsp:useBean id="login" class="tom.jiafei.Login" scope="session" >

</jsp:useBean>

<jsp:useBean id="handle" class="tom.jiafei.HandleMessage" scope="page" >

</jsp:useBean>

<% //如果客户直接进入该页面将被转向登录页面。

    if(session.isNew())

        {response.sendRedirect("login.jsp");

        }

    // 如果没有成功登录将被转向登录页面

    String success=login.getSuccess();

        if(success==null)

            {success="";

            }

    if(!(success.equals("ok")))

        {response.sendRedirect("login.jsp");

        }

%>

<HTML>

<BODY ><Font size=1>

```

```
<%@ include file="head.txt" %>
```

```
<P> 公共留言列表:
```

```
<%! // 声明一个共享的连接对象:
```

```
    Connection con=null;
```

```
    // 显示数据库记录的方法:
```

```
    public void showList(ResultSet rs,javax.servlet.jsp.JspWriter out,int
```

```
n,tom.jiafei.HandleMessage h)
```

```
    {try
```

```
    {
```

```
        out.print("<Table Border>");
```

```
        out.print("<TR>");
```

```
        out.print("<TH width=50>"+ "<Font size=1>"+ "会员名"+ "</FONT>");
```

```
        out.print("<TH width=70>"+ "<Font size=1>"+ "留言信息
```

```
        "+ "</FONT>");
```

```
        out.print("</TR>");
```

```
        for(int i=1;i<=n;i++)
```

```
        {
```

```
            String logname=rs.getString("logname");
```

```
            String message=rs.getString("public");
```

```
            if(logname==null)
```

```
                {logname="";
```

```
                }
```

```
            if(message==null)
```

```
{message="";  
}
```

// 为了能显示原始的 HTML 或 JSP 文件格式的信息,需对信息进行回压流处理:

```
h.setContent(message);  
message=h.getContent();  
  
// 将信息显示在表格中:  
out.print("<TR>");  
  
out.print("<TD >"+logname+"</TD>");  
out.print("<TD >"+message+"</TD>");  
out.print("</TR>");  
  
rs.next();  
}  
out.print("</Table>");  
}  
catch(Exception e1) {}  
}  
%>  
  
<% Statement sql=null;  
  
ResultSet rs=null;  
  
int rowCount=0; // 总的记录数。  
  
String logname="";  
  
// 第一个客户负责建立连接对象:
```

```

if(con==null)

{ try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

    catch(ClassNotFoundException e)

    {out.print(e);

    }

try

{con=DriverManager.getConnection("jdbc:odbc:friend","", "");

sql=

```

```

con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);

```

```

rs=sql.executeQuery("SELECT * FROM wordpad"); //          返回可滚动的
结果集。

```

```

rs.last(); //          将游标移动到最后一行。

```

```

int number=rs.getRow(); //          获取最后一行的行号。

```

```

rowCount=number; //          获取记录数。

```

```

handlePage.setPageSize(3); //          设置每页显示的记录数。

```

```

handlePage.setShowPage(1); //          设置欲显示的页码数。

```

```

handlePage.setPageCount(rowCount,handlePage.getPageSize()); //          计算
总页数。

```

```

out.print("          共有"+handlePage.getPageCount()+"页， ");

```

```

out.print("          每页显示"+ handlePage.getPageSize()+"条记录");

```

```

    }

    catch(SQLException e)

    {out.print(e);

    }

}

// 其它客户通过同步块使用这个连接：

else

{ synchronized(con)

{ try { sql=

```

```

con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);

```

```

        rs=sql.executeQuery("SELECT * FROM wordpad"); //          返回可滚动的
结果集。

```

```

        rs.last(); //          将游标移动到最后一行。

```

```

        int number=rs.getRow(); //          获取最后一行的行号。

```

```

        rowCount=number; //          获取记录数。

```

```

        handlePage.setPageSize(3); //          设置每页显示的记录数。

```

```

        handlePage.setShowPage(1); //          设置欲显示的页码数。

```

```

        handlePage.setPageCount(rowCount,handlePage.getPageSize()); //          计
算总页数。

```

```

        out.print("          共有"+handlePage.getPageCount()+"页， ");

```

```

        out.print("          每页显示"+ handlePage.getPageSize()+"条记录");

```

```

    }

    catch(SQLException e)

        {out.print(e);

        }

    }

}

%>

<%-- 选择显示某页的表单 --%>

<Form action="" method="post" >

    输入欲要显示的页<Input type="text" name="ok" value="1">

    <Input type="submit" value="submit">

</FORM>

<% // 获取表单提交的信息:

    String s=request.getParameter("ok");

    if(s==null)

        {s="1";

        }

    int m=Integer.parseInt(s);

    handlePage.setShowPage(m);

    out.print("        目前显示第"+handlePage.getShowPage()+"页");

    int n=handlePage.getShowPage();

    //        将游标移到:

    rs.absolute((n-1)*handlePage.getPageSize()+1);

```


secretMessage.jsp :

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="java.io.*" %>

<%@ page import="tom.jiafei.Login" %>

<%@ page import="tom.jiafei.PageNumber" %>

<%@ page import="tom.jiafei.HandleMessage" %>

<jsp:useBean id="handlePage" class="tom.jiafei.PageNumber" scope="session" >

</jsp:useBean>

<jsp:useBean id="login" class="tom.jiafei.Login" scope="session" >

</jsp:useBean>

<jsp:useBean id="handle" class="tom.jiafei.HandleMessage" scope="page" >

</jsp:useBean>

<% //如果客户直接进入该页面将被转向登录页面。

    if(session.isNew())

        {response.sendRedirect("login.jsp");

        }

    // 如果没有成功登录将被转向登录页面

    String success=login.getSuccess();

        if(success==null)

            {success="";

            }

    if(!(success.equals("ok")))
```

```

        {response.sendRedirect("login.jsp");
    }

%>

<HTML>

<BODY ><Font size=1>

<%@ include file="head.txt" %>

<P> 公共留言列表:

<%! // 声明一个共享的连接对象:

    Connection con=null;

    // 显示数据库记录的方法:

    public void showList(ResultSet rs,javax.servlet.jsp.JspWriter out,int
n,tom.jiafei.HandleMessage h)

    {try

    {

        out.print("<Table Border>");

        out.print("<TR>");

        out.print("<TH width=50>"+<Font size=1>"+"        会员名"+</FONT>");

        out.print("<TH width=70>"+<Font size=1>"+"        留 言 信 息

"+</FONT>");

        out.print("<TH width=70>"+<Font size=1>"+"        留 言 时间

"+</FONT>");

        out.print("<TH width=70>"+<Font size=1>"+"        删 除 留 言

"+</FONT>");

```

```

    out.print("</TR>");

for(int i=1;i<=n;i++)
{
    String logname=rs.getString("logname");

    if(logname==null)
    {logname="";
    }

    String message=rs.getString("message");

    if(message==null)
    {message="";
    }

    String time =rs.getString("time"); //          获取该信息的留言时间

    if(time==null)
    {time="";
    }

    //          为了能显示原始的 HTML 或 JSP 文件格式的信息,需对信息进行流处理:

    h.setContent(message);

    message=h.getContent();

    //          将信息显示在表格中:

    out.print("<TR>");

    out.print("<TD >"+logname+"</TD>");

    out.print("<TD >"+message+"</TD>");

    out.print("<TD >"+time+"</TD>");

```

```

//      添加一个删除该信息的表单：

String s1="<Form action=delete.jsp method=post>";

String s2="<input type=hidden name=time value='"+time+"'>";

String s3="<input type=submit value=      删除该留言></FORM> ";

String s=s1+s2+s3;

out.print("<TD >" +s+ "</TD>");

out.print("</TR>") ;

rs.next();

}

out.print("</Table>");

}

catch(Exception e1) {}

}

%>

<% Statement sql=null;

ResultSet rs=null;

int rowCount=0; //      总的记录数。

String logname="";

//      第一个客户负责建立连接对象：

if(con==null)

{ try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

}

catch(ClassNotFoundException e)

```

```

        {out.print(e);

        }

    try

        {con=DriverManager.getConnection("jdbc:odbc:friend","", "");

        sql=

con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);

        String s=login.getLogname();

        if(s==null)

            {s="";

            }

        //      得到自己的私人留言:

        String condition="SELECT * FROM secretwordpad WHERE logname =

"+s+"";

        rs=sql.executeQuery(condition); //      返回可滚动的结果集。

        rs.last(); //      将游标移动到最后一行。

        int number=rs.getRow(); //      获取最后一行的行号。

        rowCount=number; //      获取记录数。

        handlePage.setPageSize(3); //      设置每页显示的记录数。

        handlePage.setShowPage(1); //      设置欲显示的页码数。

        handlePage.setPageCount(rowCount,handlePage.getPageSize()); //      计
算总页数。

```

```

        out.print("        共有"+handlePage.getPageCount()+"页， ");
        out.print("        每页显示"+ handlePage.getPageSize()+"条记录");
    }
    catch(SQLException e)
    {out.print(e);
    }
}
// 其它客户通过同步块使用这个连接：
else
{ synchronized(con)
{ try { sql=

con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);

String s=login.getLogname();

if(s==null)

{s="";

}

// 得到自己的私人留言：

String condition="SELECT * FROM secretwordpad WHERE logname =

"+s+"";

rs=sql.executeQuery(condition); // 返回可滚动的结果集。

rs.last(); // 将游标移动到最后一行。

```

int number=rs.getRow(); // 获取最后一行的行号。

rowCount=number; // 获取记录数。

handlePage.setPageSize(3); // 设置每页显示的记录数。

handlePage.setShowPage(1); // 设置欲显示的页码数。

handlePage.setPageCount(rowCount,handlePage.getPageSize()); // 计

算总页数。

out.print(" 共有"+handlePage.getPageCount()+"页， ");

out.print(" 每页显示"+ handlePage.getPageSize()+"条记录");

}

catch(SQLException e)

{out.print(e);

}

**catch(IOException ee){}
}**

}

%>

<%-- 选择显示某页的表单 --%>

<Form action="" method="post" >

输入欲要显示的页<Input type="text" name="ok" value="1">

<Input type="submit" value="submit">

</FORM>

<% // 获取表单提交的信息:

String s=request.getParameter("ok");


```

        if(s==null)

            {s="1";

            }

        int m=Integer.parseInt(s);

        handlePage.setShowPage(m);

        out.print("        目前显示第"+handlePage.getShowPage()+"页");

        int n=handlePage.getShowPage();

        //        将游标移到:

        rs.absolute((n-1)*handlePage.getPageSize()+1);

        showList(rs,out,handlePage.getPageSize(),handle); //        显示该页的内容。

    %>

</FONT>

</BODY>

</HTML>

```

删除私人留言的页面

delete.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="tom.jiafei.Login" %>

<jsp:useBean id="login" class="tom.jiafei.Login" scope="session" >

</jsp:useBean>

<% //如果客户直接进入该页面将被转向登录页面。

```

```

if(session.isNew())

    {response.sendRedirect("login.jsp");

    }

// 如果没有成功登录将被转向登录页面

String success=login.getSuccess();

    if(success==null)

        {success="";

        }

if(!(success.equals("ok")))

    {response.sendRedirect("login.jsp");

    }

%>

<HTML>

<BODY ><Font size=1>

<%@ include file="head.txt" %>

<% // 获取提交的信息的时间:

    String time=request.getParameter("time");

        if(time==null)

            {time="";

            }

    byte b[]=time.getBytes("ISO-8859-1");

    time=new String(b);

Connection con=null;

```

```

Statement sql=null;

ResultSet rs=null;

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

catch(ClassNotFoundException event){}

try

{con=DriverManager.getConnection("jdbc:odbc:friend","","");

    sql=con.createStatement();

    //      删除操作：

    String s=login.getLogname();

    String condition1=

        "DELETE FROM secretwordpad WHERE logname ='+"+"'"+s+"'";

    String condition2=

        "AND time ='+"+"'"+time+"'";

    String condition=condition1+condition2;

    sql.executeUpdate(condition);

    out.print("      删除了该留言");

    con.close();

}

catch(SQLException event)

{out.print("'" +event);

}

```

%>

</BODY>

</HTML>

7.4.8 修改密码

在该页面输入会员名和正确密码后，可以修改密码。

图 7.14 修改密码

修改密码页面

modifyPassword.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import="java.sql.*" %>
```

```
<%@ page import="tom.jiafei.Login" %>
```

```
<jsp:useBean id="login" class="tom.jiafei.Login" scope="session" >
```

```
</jsp:useBean>
```

```
<% //如果客户直接进入该页面将被转向登录页面。
```

```

if(session.isNew())

    {response.sendRedirect("login.jsp");

    }

// 如果没有成功登录将被转向登录页面

String success=login.getSuccess();

    if(success==null)

        {success="";

        }

    if(!(success.equals("ok")))

        {response.sendRedirect("login.jsp");

        }

%>

<HTML>

<BODY bgcolor=cyan ><Font size=1>

<%@ include file="head.txt" %>

<P>修改密码,密码长度不能超过 30 个字符:

<%String str=response.encodeURL("modifyPassword.jsp");

%>

<FORM action="<%=str%>" Method="post">

<BR>输入您的会员名:

<BR><Input type=text name="logname" value="<%=login.getLogname()%>" >

<BR>输入您的密码:

<BR><Input type=password name="password">

```


输入您的新的密码:

<Input type=text name="newPassword1">

请再输入一次新密码:

<Input type=text name="newPassword2">

<Input type=submit name="g" value="提交">

</FORM>

<%!//处理字符串的一个常用方法:

```
public String getString(String s)
{ if(s==null) s="";
  try {byte a[]=s.getBytes("ISO-8859-1");
      s=new String(a);
  }
  catch(Exception e)
  { }
  return s;
}
```

%>

<% // 获取提交的会员名:

```
String logname=request.getParameter("logname");
    logname=getString(logname);
// 获取提交的密码:
String password=request.getParameter("password");
    password=getString(password);
```

// 获取提交的新密码:

```
String newPassword1=request.getParameter("newPassword1");
```

```
newPassword1=getString(newPassword1);
```

// 获取提交的新密码:

```
String newPassword2=request.getParameter("newPassword2");
```

```
newPassword2=getString(newPassword2);
```

```
try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
}
```

```
catch(ClassNotFoundException event){}
```

// 验证身份:

```
Connection con=null;
```

```
Statement sql=null;
```

```
boolean modify=false;
```

```
boolean ifEquals=false;
```

```
ifEquals=(newPassword1.equals(newPassword2))&&(newPassword1.length()<=30);
```

```
if(ifEquals==true)
```

```
{ try{ con=DriverManager.getConnection("jdbc:odbc:friend","", "");
```

```
sql=con.createStatement();
```

```
boolean bo1=logname.equals(login.getLogname()),
```

```
bo2=password.equals(login.getPassword());
```

```
if(bo1&&bo2)
```

```
{// 修改密码:
```

```

        modify=true;

        out.print("                您的密码已经更新");

        String c="UPDATE member SET password =
"+"""+newPassword1+"""+
        " WHERE logname = "+"""+logname+""";

        sql.executeUpdate(c);

    }

    con.close();

}

catch(SQLException e1) {}

}

else

    { out.print("        你两次输入的密码不一致或长度过大");

    }

    if(modify==false&&ifEquals==true)

    { out.print("<BR> 您没有输入密码帐号或<BR>您输入的帐号或密码不正确
"+logname+": "+password);

    }

    %>

</FONT>

</BODY>

</HTML>

```


7.4.9 修改个人信息

在该页面输入会员名和正确密码可以修改除密码和会员名以外的其他个人注册信息。

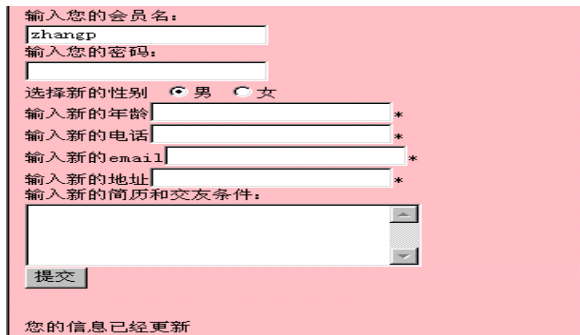


图 7.15 修改个人信息

修改个人信息页面

ModifyMessage.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="tom.jiafei.Login" %>

<jsp:useBean id="login" class="tom.jiafei.Login" scope="session" >

</jsp:useBean>

<% //如果客户直接进入该页面将被转向登录页面。

if(session.isNew())

    {response.sendRedirect("login.jsp");
```

```

    }

    // 如果没有成功登录将被转向登录页面

    String success=login.getSuccess();

        if(success==null)

            {success="";

            }

    if(!(success.equals("ok")))

        {response.sendRedirect("login.jsp");

        }

%>

<HTML>

<BODY bgcolor=pink ><Font size=1>

<%@ include file="head.txt" %>

<%String str=response.encodeURL("modifyMessage.jsp");

%>

<P>修改您的基本信息：性别、年龄、电话、电子邮件、地址、简历。

<FORM action="<%=str%>" Method="post">

<BR>输入您的会员名：

<BR><Input type=text name="logname" value="<%=login.getLogname()%>" >

<BR>输入您的密码：

<BR><Input type=password name="password">

<BR>选择新的性别

<Input type=radio name="sex" checked="o" value=" 男">男

```

```

<Input type=radio name="sex" value=" 女">女
<BR>输入新的年龄<Input type=text name="age" >*
<BR>输入新的电话<Input type=text name="phone" >*
<BR>输入新的 email<Input type=text name="email" >*
<BR>输入新的地址<Input type=text name="address" >*
<BR>输入新的简历和交友条件:

<BR><TextArea name="message" Rows="4" Cols="32">

</TextArea>

<BR><Input type=submit name="g" value="提交">

</FORM>

```

<%!//处理字符串的一个常用方法:

```

public String getString(String s)
{ if(s==null) s="?";

  try {byte a[]=s.getBytes("ISO-8859-1");

    s=new String(a);

  }

  catch(Exception e)

  {s="?";

  }

  return s;

}

```

%>

<% // 获取提交的用户名:

```
String logname=request.getParameter("logname");

    logname=getString(logname);

// 获取提交的密码:

String password=request.getParameter("password");

    password=getString(password);

// 获取新的性别:

String sex=request.getParameter("sex");

    sex=getString(sex);

String age=request.getParameter("age");

    age=getString(age);

// 获取新 email:

String email=request.getParameter("email");

    email=getString(email);

// 获取新电话:

String phone=request.getParameter("phone");

    phone=getString(phone);

// 获取新地址:

String address=request.getParameter("address");

    address=getString(address);

String message=request.getParameter("message");

    message=getString(message);

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }
```

```

catch(ClassNotFoundException event){}

// 验证身份:

Connection con=null;

Statement sql=null;

boolean modify=false;

try{ con=DriverManager.getConnection("jdbc:odbc:friend","","");

    sql=con.createStatement();

    boolean bo1=logname.equals(login.getLogname()),

        bo2=password.equals(login.getPassword());

    if(bo1&&bo2)

        {//      修改信息:

            String c1="UPDATE member SET sex = '"+sex+"'"+

                " WHERE logname = '"+logname+"'";

            String c2="UPDATE member SET age = '"+age+"'"+

                " WHERE logname = '"+logname+"'";

            String c3="UPDATE member SET email = '"+email+"'"+

                " WHERE logname = '"+logname+"'";

            String c4="UPDATE member SET phone = '"+phone+"'"+

                " WHERE logname = '"+logname+"'";

            String c5="UPDATE member SET address = '"+address+"'"+

                " WHERE logname = '"+logname+"'";

            String c6="UPDATE member SET message = '"+message+"'"+

                " WHERE logname = '"+logname+"'";

```

```

        sql.executeUpdate(c1);

        sql.executeUpdate(c2);

        sql.executeUpdate(c3);

        sql.executeUpdate(c4);

        sql.executeUpdate(c5);

        sql.executeUpdate(c6);

        out.print("<BR>        您的信息已经更新");
    }
else
    {out.print("<BR>        您还没有输入密码或您输入的密码或用户名有错误");
    }

    con.close();
}

catch(SQLException e1)
    { out.print("<BR>        更新失败");
    }

%>

</FONT>

</BODY>

</HTML>

```

第8章 网上书店

这一章讲述如何用 **JSP** 技术建立一个简单的电子商务应用系统：网上书店。我们将采用 **Sun** 公司倡导的 **JSP+Javabeans** 模式。

8.1 系统设计

注册：新用户填写表单，包括用户名、**e-mail** 地址等信息。如果输入的用户名已经被其他用户注册使用，系统提示用户更改自己的用户名

用户登录：输入用户名、密码。如果用户输入的用户名或密码有错误，系统将显示错误信息；如果登录成功，就将一个成功登录的信息赋值给用户，同时用户被链接到“订购图书”页面。

浏览图书书目：成功登录的用户可以分页浏览图书书目，并将想要订购的图书提交到填写订单页面。如果用户直接进入该页面或没有成功登录就进入该页面，将被链接到“用户登录”页面。

订购图书：成功登录的用户可以在该页面订购所需要的图书，如果用户直接进入该页面或没有成功登录就进入该页面，将被链接到“用户登录”页面。

查看订单：成功登录的用户可以在该页面查看自己的订单，如果用户直接进入该页面或没有成功登录就进入该页面，将被链接到“用户登录”页面。

修改订单：成功登录的用户可以在该页面修改或删除自己的订单，如果用户直接进入该页面或没有成功登录就进入该页面，将被链接到“用户登录”页面。

修改密码：成功登录的用户可以在该页面修改自己的登录密码，如果用户直接进入该页面或没有成功登录就进入该页面，将被链接到“用户登录”页面。

修改注册信息：成功登录的用户可以在该页面修改自己的注册信息，比如联系电话、

通信地址等，如果用户直接进入该页面或没有成功登录就进入该页面，将被链接到“用户登录”页面。

8.2 数据库设计及连接

我们采用 JDBC-ODBC 桥接器方式访问数据库。我们用 Access 建立一个数据库 shop.mdb，并将该数据库设置成一个名字是 shop 的数据源。该库有如下的表。

(1)注册信息表：user

用户的注册信息存入数据库 shop.mdb 的 user 表中。user 表结构如图 8.1 所示。

(2)书目表：book

book 表用来存放图书书目。book 表结构如图 8.2 所示。

(3)订单表：orderform

该表存放各个用户的订购信息。orderform 表结构如图 8.3 所示。



字段名称	数据类型	说明
logname	文本	用户登陆名称
realname	文本	用户的真实姓名
password	文本	口令
email	文本	电子邮件地址
phone	文本	电话
address	文本	邮寄地址

图 8.1 注册信息表



字段名称	数据类型	说明
id	自动编号	
order_number	文本	订阅号
book_name	文本	书名
author	文本	作者
publisher	文本	出版社
time	日期/时间	出版时间
price	文本	单价
category	文本	分类

图 8.2 书目表

orderform: 表			
	字段名称	数据类型	说明
▶	logname	文本	已注册的用户名
	realname	文本	真实姓名
▼	order_number	文本	图书订购号
	bookname	文本	订购的图书的名字
	mount	文本	订购的数量
	phone	文本	联系电话
	address	文本	邮寄地址
字段属性			

图 8.3 订单表

(4) 表 user 和表 orderform 之间的关联关系

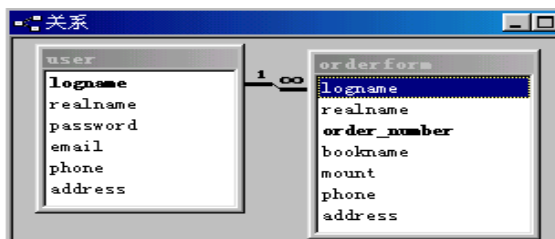


图 8.4 表的关联关系

8.3 页面管理

所有的页面将包括一个导航条，该导航条由注册、登录、浏览、订购、修改密码、修改个人信息组成。为了减少代码的编写，其它页面通过使用 JSP 的 `<%@ include>` 标签将导航条文件：`head.txt` 嵌入自己的页面。我们将所有的 `beans` 存放在 JSP 引擎的 `classes` 目录中，所有的 JSP 页面以及导航条文件存放在 JSP 引擎的 `webapps/Root` 目录中。

`head.txt`:

```
<table align="center" border="0" width="740" height="18" bgcolor=yellow
cellspacing="1">
<tr>
<td width="100%">
<a
href="<%=response.encodeURL(" http://192.168.1.100:8080/showBookList.jsp ")%>"> 书
目浏览</a> |
<a
href="<%=response.encodeURL(" http://192.168.1.100:8080/userRegister.jsp ")%>">用户
注册</a> |
<a href="<%=response.encodeURL(" http://192.168.1.100:8080/userLogin.jsp ")%>">
用户登录 </a> |
<a href="<%=response.encodeURL(" http://192.168.1.100:8080/buybook.jsp ")%>">订
```

购图书 |

<a

href="<%=response.encodeURL(" http://192.168.1.100:8080/modifyForm.jsp ")%>">修改

订单 |

<a

href="<%=response.encodeURL(" http://192.168.1.100:8080/showOrderForm.jsp ")%>">

查看订单|

<a

href="<%=response.encodeURL(" http://192.168.1.100:8080/modifyPassword.jsp ")%>">

修改密码 |

<a

href="<%=response.encodeURL(" http://192.168.1.100:8080/modifyMessage.jsp ")%>">

修改个人信息 |</td>

</tr>

</table>

8.4 各个页面的设计

对使用 **beans** 的页面，我们首先给出 **beans** 的代码，然后阐述页面的设计过程。

我们尽量减少不必要的 **HTML** 标签，重点体现 **JSP** 的功能。主页由导航条和一个欢迎语组成。

主页（效果如图 8.5 所示）

bookmain.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

<HTML>

<BODY bgcolor =green>

<%@ include file="head.txt" %>

<H1>

<CENTER> 欢迎光临网上书店 </CENTER>

</BODY>

</HTML>

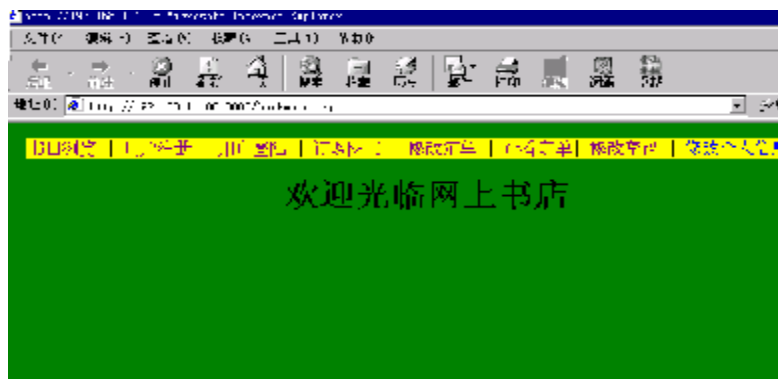


图 8.5 主页

8.4.1 用户注册

用户的注册信息需要存入数据库 shop.dbm 的 user 表中。

注册页面使用的 beans

Register.java:

```
import java.sql.*;

public class Register
{
    String logname,
        realname,
        password,
        email,
        phone,
        address;

    String message;

    Connection con;

    Statement sql;

    ResultSet rs;

    public Register()
    {
        // 加载桥接器:

        try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            }

        catch(ClassNotFoundException e){}

    }

    // 设置属性值、获取属性值的方法:

    public void setLogname(String name)

    {
        logname=name;

    }
}
```

```
public String getLogname()

{return logname;

}

public void setRealname(String name)

{ realname=name;

}

public String getRealname()

{return realname;

}

public void setPassword(String pw)

{ password=pw;

}

public String getPassword()

{return password;

}

public void setEmail(String em)

{ email=em;

}

public String getEmail()

{return email;

}

public void setPhone(String ph)

{ phone=ph;
```

```

    }

    public String getPhone()

    {return phone;

    }

    public void setAddress(String ad)

    { address=ad;

    }

    public String getAddress()

    {return address;

    }

    public String getMessage()

    {return message;

    }

    // 添加记录到数据库的 user 表:

    public void addItem()

    {try{

        con=DriverManager.getConnection("jdbc:odbc:shop","","");

        sql=con.createStatement();

        String s=

        """+logname+"""+", """+realname+"""+", """+password+"""+", """+

        email+"""+", """+phone+"""+", """+address+""";

        String condition="INSERT INTO user VALUES"+"("+s+")";

        sql.executeUpdate(condition);

```

```

        message="    注册成功了";

        con.close();

    }

    catch(SQLException e)

    {message="    你还没有注册，或该用户已经存在，请你更换一个名字";

    }

}

}

```

地址 http://192.168.1.100:8080/userRegister.jsp

[书目浏览](#) | [用户注册](#) | [用户登陆](#) | [订购图书](#) |

输入您的信息，带*号项必须填写。

登陆名称 *

真实姓名 *

设置密码 *

电子邮件 *

联系电话 *

通信地址 *

注册成功了

图 8.6 用户注册

注册页面（效果如图 8.6 所示）

userRegister.jsp:


```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="Register" %>

<%! //处理字符串的方法:

public String codeString(String s)

{ String str=s;

    try{byte b[]=str.getBytes("ISO-8859-1");

        str=new String(b);

        return str;

    }

    catch(Exception e)

    { return str;

    }

}

%>

<HTML>

<BODY ><Font size=1>

<%@ include file="head.txt" %>

<Font size=1>

<%String str=response.encodeURL("userRegister.jsp");

%>

<P>输入您的信息，带*号项必须填写:

<FORM action="<%=str%>" Method="post">

<BR>登录名称<Input type=text name="logname">*

```

**
真实姓名<Input type=text name="realname">***

**
设置密码<Input type=password name="password">***

**
电子邮件<Input type=text name="email">***

**
联系电话<Input type=text name="phone">***

**
通信地址<Input type=text name="address">***

**
<Input type=submit name="g" value="提交">**

</Form>

<jsp:useBean id="login" class="Register" scope="request" >

</jsp:useBean>

<% // 提交信息后，进行注册操作：

String logname="",realname="",password="",email="",phone="",address="";

if(!(session.isNew()))

{ logname=request.getParameter("logname");

if(logname==null)

{logname="";

}

logname=codeString(logname);

realname=request.getParameter("realname");

if(realname==null)

{realname="";

}

realname=codeString(realname);

password=request.getParameter("password");

```

        if(password==null)

        {password="";

        }

password=codeString(password);

email=request.getParameter("email");

        if(email==null)

        {email="";

        }

email=codeString(email);

phone=request.getParameter("phone");

        if(phone==null)

        {phone="";

        }

phone=codeString(phone);

address=request.getParameter("address");

        if(address==null)

        {address="";

        }

address=codeString(address);

    }

%>

<% // 为了以后处理汉字方便, 我们采用了第 1 种方式初始化 beans

    if(!(logname.equals(""))&&!(address.equals(""))&&!(phone.equals("")))

```

```

        &&!(realname.equals(""))&&!(password.equals(""))))

    {%>

        <jsp:setProperty name= "login" property="logname"
value="<%=logname%>" />

        <jsp:setProperty name= "login" property="realname"
value="<%=realname%>" />

        <jsp:setProperty name= "login" property="password"
value="<%=password%>" />

        <jsp:setProperty name= "login" property="email" value="<%=email%>"
/>

        <jsp:setProperty name= "login" property="phone"
value="<%=phone%>" />

        <jsp:setProperty name= "login" property="address"
value="<%=address%>" />

        <%
            login.addItem();
        }
        else
        {out.print(" 你还没有填写信息，或信息填写不完整");
        }
    %>

    <% // 返回注册信息

        if(!(session.isNew()))

```

```

    {
%>

    <jsp:getProperty name= "login" property="message" />

<%

    }

%>

</Body>

</HTML>

```

8.4.2 用户登录

用户可在该页面输入自己的用户名和密码，系统将对用户名和密码进行验证，如果身份正确将被连接到订购图书页面，否则提示用户输入的密码或用户名不正确。

登录页面使用的 beans

Login.java:

```

import java.sql.*;

public class Login
{
    String logname,
        realname,
        password,
        phone,
        address;

    String success="false",message="";

    Connection con;

```

```

Statement sql;

ResultSet rs;

public Login()
{ //    加载桥接器：

    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        }

    catch(ClassNotFoundException e){}

}

// 设置属性值、获取属性值的方法：

public void setLogname(String name)

{ logname=name;

}

public String getLogname()

{return logname;

}

public void setPassword(String pw)

{ password=pw;

}

public String getPassword()

{return password;

}

public void setRealname(String name)

{ realname=name;

```

```

    }

    public String getRealname()
    {return realname;
    }

    public void setPhone(String ph)
    { phone=ph;
    }

    public String getPhone()
    {return phone;
    }

    public void setAddress(String ad)
    { address=ad;
    }

    public String getAddress()
    {return address;
    }

    public String getSuccess()
    {return success;
    }

    // 查询数据库的 user 表:

    public String getMessage()
    {try{

        con=DriverManager.getConnection("jdbc:odbc:shop","","");
    }

```

```

sql=con.createStatement();

String condition=

"SELECT * FROM user WHERE logname = '"+logname+"'";

rs=sql.executeQuery(condition);

int rowcount=0;

String ps=null;

while(rs.next())

{ rowcount++;

    logname=rs.getString("logname");

    realname=rs.getString("realname");

    ps=rs.getString("password");

    phone=rs.getString("phone");

    address=rs.getString("address");

}

if((rowcount==1)&&(password.equals(ps)))

{ message="ok";

    success="ok";

}

else

{message="      输入的用户名或密码不正确";

    success="false";

}

con.close();

```



```

        return message;
    }
    catch(SQLException e)
    { message="    输入的用户名或密码不正确";
      success="false";
      return message;
    }
}
}
}

```

登录页面（效果如图 8.7 所示）

userLogin.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="Login" %>

<%! //处理字符串的方法:

public String codeString(String s)

{ String str=s;

  try{byte b[]=str.getBytes("ISO-8859-1");

    str=new String(b);

    return str;

  }

  catch(Exception e)

  { return str;

```

```

        }

    }

%>

<HTML>

<BODY ><Font size=1>

<%@ include file="head.txt" %>

<Font size=1>

<%String string=response.encodeURL("userLogin.jsp");

%>

<P>输入用户名和密码:

<FORM action="<%=string%>" Method="post">

<BR>登录名称<Input type=text name="logname">

<BR>输入密码<Input type=password name="password">

<BR><Input type=submit name="g" value="提交">

</Form>

<jsp:useBean id="login" class="Login" scope="session" >

</jsp:useBean>

<% // 提交信息后, 验证信息是否正确:

    String message="",

        logname="",

        password="";

    if(!(session.isNew()))

        {logname=request.getParameter("logname");

```

```

        if(logname==null)

            {logname="";

            }

        logname=codeString(logname);

        password=request.getParameter("password");

        if(password==null)

            {password="";

            }

        password=codeString(password);

    }

%>

<%

    if(!(logname.equals("")))

    {

%>

        <jsp:setProperty name= "login" property="logname"

value="<%=logname%>" />

        <jsp:setProperty name= "login" property="password"

value="<%=password%>" />

        <%

            message=login.getMessage();

            if(message==null)

                {message="";

```

```

    }

}

%>

<% if(!(session.isNew()))

    { if(message.equals("ok"))

        { String str=response.encodeURL("buybook.jsp");

            response.sendRedirect(str);

        }

    else

        {out.print(message);

        }

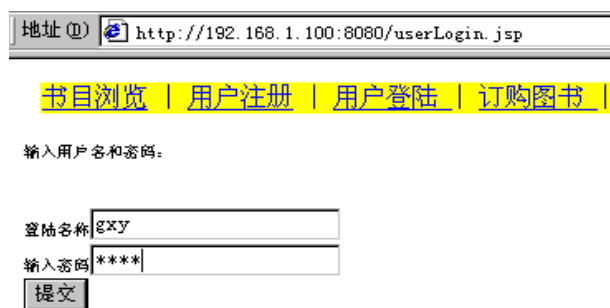
    }


%>

</Body>

</HTML>

```



地址 (U)  http://192.168.1.100:8080/userLogin.jsp

[书目浏览](#) | [用户注册](#) | [用户登陆](#) | [订购图书](#) |

输入用户名和密码。

登陆名称

输入密码

图 8.7 用户登录

8.4.3 用户订购

成功登录的用户可以在该页面订购图书。输入正确的用户名和密码之后，就可以订购图书了。用户将订购的图书存入订 货单，如果用户已经订购了该图书，就必须到修改订 单页面修改订单后才能再订购该书。

订购页面使用的 beans (该 beans 负责查阅用户准备订购的图书)

BuyBook.java:

```
import java.sql.*;

public class BuyBook
{
    long id=0;

    String order_number,

        book_name;

    Connection con;

    Statement sql;

    ResultSet rs;

    public BuyBook()
    { //    加载桥接器:

        try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            }

        catch(ClassNotFoundException e){}
```

```

    }

// 设置属性值、获取属性值的方法：

public void setId(long n)

    {id=n;

    }

public long getId()

    {return id;

    }

public void setBook_name(String name)

    { book_name=name;

    }

public String getBook_name()

    {return book_name;

    }

public void setOrder_number(String number)

    { order_number=number;

    }

public String getOrder_number()

    {return order_number;

    }

// 通过书的 id 号查询数据库的 book 表：

public StringBuffer getMessageBybook_id()

    {

```

```

StringBuffer buffer=new StringBuffer();

try{

    con=DriverManager.getConnection("jdbc:odbc:shop","","");

    sql=con.createStatement();

    String condition="SELECT * FROM book WHERE id = "+id;

    rs=sql.executeQuery(condition);

    buffer.append("<Table Border><FONT size=1> ");

    buffer.append("<TR>");

    buffer.append("<TH width=50>"+<Font size=1>"+id"+</FONT>");

    buffer.append("<TH width=50>"+<Font size=1>"+订购号"+</FONT>");

    buffer.append("<TH width=70>"+<Font size=1>"+书名"+</FONT>");

    buffer.append("<TH width=60>"+<Font size=1>"+作者"+</FONT>");

    buffer.append("<TH width=60>"+<Font size=1>"+出版社"+</FONT>");

    buffer.append("<TH width=50>"+<Font size=1>"+出版 时间

"+</FONT>");

    buffer.append("<TH width=20>"+<Font size=1>"+价钱"+</FONT>");

    buffer.append("<TH width=50>"+<Font size=1>"+分类"+</FONT>");

    buffer.append("</TR>");

    while(rs.next())

    { order_number=rs.getString(2);

    book_name=rs.getString(3);

    String 作者=rs.getString(4);

    String 出版社=rs.getString(5);

```

```

        Date            时间=rs.getDate(6);

        String          价格=rs.getString("price");

        String          分类=rs.getString("category");

        buffer.append("<TR>");

        buffer.append("<TD >"+"<Font
size=1>" +rs.getLong(1)+"</FONT>");

        buffer.append("<TD >"+"<Font
size=1>" +order_number+"</FONT>");

        buffer.append("<TD >"+"<Font size=1>" +book_name+"</FONT>");

        buffer.append("<TD >"+"<Font size=1>" +        作者+"</FONT>");

        buffer.append("<TD >"+"<Font size=1>" +        出版社+"</FONT>");

        buffer.append("<TD >"+"<Font size=1>" +        时间+"</FONT>");

        buffer.append("<TD >"+"<Font size=1>" +        价格+"</FONT>");

        buffer.append("<TD >"+"<Font size=1>" +        分类+"</FONT>");

        buffer.append("</TR>");

    }

    buffer.append("</TABLE>");

    buffer.append("</FONT>");

    con.close();

    return buffer;

}

catch(SQLException e)

    { return buffer;

```



```
    }  
  }  
}
```

用户订购页面使用的 beans(该 beans 负责填写订购单):

OrderForm.java:

```
import java.sql.*;  
  
public class OrderForm  
{ String logname, // 用户名。  
    realname, // 真实姓名。  
    order_number, // 图书订购号  
    phone,  
    address,  
    book_name, // 书名。  
    mount; // 数量。  
  
    Connection con;  
  
    Statement sql;  
  
    ResultSet rs;  
  
    public OrderForm()  
    { // 加载桥接器:  
        try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
            }  
  
        catch(ClassNotFoundException e){}
```

```
}  
  
// 设置属性值、获取属性值的方法：  
  
public void setLogname(String name)  
  
    { logname=name;  
  
    }  
  
public String getLogname()  
  
    {return logname;  
  
    }  
  
public void setRealname(String name)  
  
    { realname=name;  
  
    }  
  
public String getRealname()  
  
    {return realname;  
  
    }  
  
public void setOrder_number(String number)  
  
    { order_number=number;  
  
    }  
  
public String getOrder_number()  
  
    {return order_number;  
  
    }  
  
public void setBook_name(String name)  
  
    { book_name=name;  
  
    }
```

```

public String getBook_name()

    {return book_name;

    }

public void setPhone(String ph)

    {   phone=ph;

    }

public String getPhone()

    {return phone;

    }

public void setAddress(String ad)

    {   address=ad;

    }

public String getAddress()

    {return address;

    }

public void setMount(String n)

    {   mount=n;

    }

public String getMount()

    {return mount;

    }

// 向数据库的 orderform 订单表添加订购记录:

public String setOrderBook()

```

```

{ try{con=DriverManager.getConnection("jdbc:odbc:shop","","");

    sql=con.createStatement();

    String s=

"""+logname+"""+","+""""+realname+"""+","+""""+order_number+"""+","+""""+

book_name+"""+","+""""+mount+"""+","+""""+phone+"""+","+""""+address+""";

    String condition="INSERT INTO orderform VALUES"+"("+s+")";

    sql.executeUpdate(condition);

    con.close();

    return "    该书被添加到你的订单";

}

catch(SQLException e)


{ return "    你已经订购了该书，请去修改订单后再订购";

}

}

}

```

地址  http://192.168.1.100:8080/buybook.jsp

[书目浏览](#) | [用户注册](#) | [用户登陆](#) | [订购图书](#) | [修改定单](#) | [查看定单](#)

输入要订购的书的序列号:

查询到如下记录:

id	订购号	书名	作者	出版社	出版时间	价格	分类
2	TP124	java 2 实用教程	耿祥义	清华大学出版社	2001-10-01	39	计算机图书

如果准备订购该书，请填写下列定单，点击“提交定单”按钮

他的用户名 *

他的密码 *

订购数量 (单位: 册)

该书被添加到你的定单

订购图书页面（效果如图 8.8 所示）

buybook.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="BuyBook" %>

<%@ page import="Login" %>

<%@ page import="OrderForm" %>

<jsp:useBean id="login" class="Login" scope="session" >

</jsp:useBean>

<jsp:useBean id="book" class="BuyBook" scope="session" >

</jsp:useBean>

<jsp:useBean id="orderform" class="OrderForm" scope="page" >

</jsp:useBean>

<%! //处理字符串的方法:
```

```

public String codeString(String s)

{ String str=s;

    try{byte b[]=str.getBytes("ISO-8859-1");

        str=new String(b);

        return str;

    }

    catch(Exception e)

        { return str;

        }

}

%>

<HTML>

<BODY ><Font size=1>

<%@ include file="head.txt" %>

<% //如果客户直接进入该页面将被转向登录页面。

    if(session.isNew())

        {response.sendRedirect("userLogin.jsp");

        }

// 如果没有成功登录将被转向登录页面

String success=login.getSuccess();

    if(success==null)

        {success="";

        }

}

```

```

    if(!(success.equals("ok")))

        {response.sendRedirect("userLogin.jsp");

        }

%>

<%String str=response.encodeURL("buybook.jsp");

%>

<FORM action="<%=str%>" Method="post" >

    <P> 输入要订购的书的序列号:

    <Input type=text name="id">

    <Input type=submit name="g" value=" 提交">

</Form>

<jsp:setProperty name= "book" property="id" param="id" />

    查询到如下记录: <BR>

<% StringBuffer b=book.getMessageBybook_id();

%>

<%=b%>

<P>如果准备订购该书, 请填写订单, 点击"添加到订单"按钮<BR>

<%if((book.getId())!=0)

    {%> <FORM action="<%=str%>" method=post>

        <BR>      您    的 用      户 名      <Input TYPE=text name=logname

value="<%=login.getLogname()%>" >*

        <BR>      您的密码<Input TYPE=password name=password>*

        <BR>      订购数量<Input TYPE=text name=mount value=1>(单位: 册)

```

```

<input type=submit name="k" value="    提交订单">

</FORM>

<%}

%>

<% if((book.getId())!=0)

    { String name=request.getParameter("logname");//    获取在表单中提交的用户名。

        if(name==null)

            {name="";

            }

        name=codeString(name);

        String word=request.getParameter("password");//    获取在表单中提交的密码。

        if(word==null)

            {word="";

            }

        word=codeString(word);

        String mount=request.getParameter("mount");//    获取在表单中提交的密码。

        mount=codeString(mount);

        //    判断提交的名字和密码是否正确:

        //    如果正确就初始化 orderform 的值，并添加数据到订单。

        if((name.equals(login.getLogname()))&&(word.equals(login.getPassword())))

        {

            %>

<jsp:setProperty name= "orderform" property="logname"

```



```

value="<%=login.getLogname()%>"/>

    <jsp:setProperty name= "orderform" property="realname"
value="<%=login.getRealname()%>"/>

    <jsp:setProperty name= "orderform"
        property="order_number"
value="<%=book.getOrder_number()%>"/>

    <jsp:setProperty name= "orderform"
        property="book_name"
value="<%=book.getBook_name()%>"/>

    <jsp:setProperty name= "orderform" property="mount"
value="<%=mount%>"/>

    <jsp:setProperty name= "orderform" property="phone"
value="<%=login.getPhone()%>"/>

    <jsp:setProperty name= "orderform" property="address"
value="<%=login.getAddress()%>"/>

    <% String ms=orderform.setOrderBook();

        out.print("<BR>" + ms);

    }

else

    { out.print("<BR>" + "    您必须输入正确的密码和用户名");

    }

}

%>

```

```
</Font>

</BODY>

</HTML>
```

8.4.4 查看订单

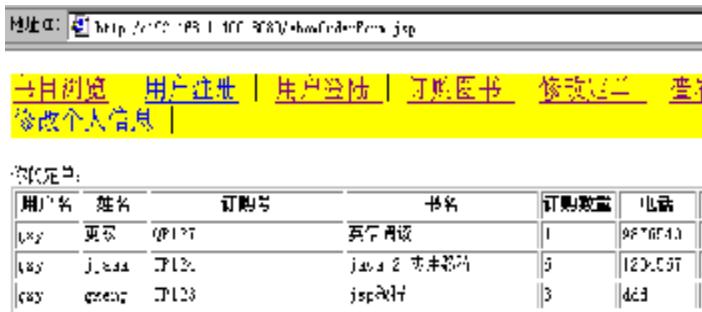


图 8.9 显示订单

查看订单页面（效果如图 8.9 所示）

showOrderForm.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="Login" %>

<jsp:useBean id="login" class="Login" scope="session" >

</jsp:useBean>

<% //如果客户直接进入该页面将被转向登录页面。

if(session.isNew())
```

```

        {response.sendRedirect("userLogin.jsp");
    }

    // 如果没有成功登录将被转向登录页面

    String success=login.getSuccess();

        if(success==null)

            {success="";

            }

    if(!(success.equals("ok")))

        {response.sendRedirect("userLogin.jsp");

        }

%>

<HTML>

<BODY ><Font size=1>

<P>你的订单:

<% String logname=login.getLogname();

        if(logname==null)

            {logname="";

            }

    Connection con;

    Statement sql;

    ResultSet rs;

    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

```

```

catch(ClassNotFoundException e){}

try{ con=DriverManager.getConnection("jdbc:odbc:shop","", "");

    sql=con.createStatement();

    String condition="SELECT * FROM orderform WHERE logname =
"+"""+logname+""";

    rs=sql.executeQuery(condition);

    out.print("<Table Border>");

    out.print("<TR>");

    out.print("<TH width=50>"+<Font size=1>"+"      用户名");
    out.print("<TH width=50>"+<Font size=1>"+"      姓名");
    out.print("<TH width=160>"+<Font size=1>"+"      订购号");
    out.print("<TH width=160>"+<Font size=1>"+"      书名");
    out.print("<TH width=60>"+<Font size=1>"+"      订购数量");
    out.print("<TH width=60>"+<Font size=1>"+"      电话");
    out.print("<TH width=160>"+<Font size=1>"+"      地址");

    out.print("</TR>");

    while(rs.next())

    { out.print("<TR>");

        out.print("<TD >"+<Font size=1>"+rs.getString(1)+"</TD>");
        out.print("<TD >"+<Font size=1>"+rs.getString(2)+"</TD>");
        out.print("<TD >"+<Font size=1>"+rs.getString(3)+"</TD>");
        out.print("<TD >"+<Font size=1>"+rs.getString(4)+"</TD>");
        out.print("<TD >"+<Font size=1>"+rs.getString(5)+"</TD>");

```

```

        out.print("<TD >"+"<Font size=1>" +rs.getString(6)+"</TD>");

        out.print("<TD >"+"<Font size=1>" +rs.getString(7)+"</TD>");

        out.print("</TR>") ;

    }

    out.print("</Table>");

    con.close();

}

catch(SQLException e)

{

}

%>

</BODY>

</HTML>

```

8.4.5 修改订单

根据书的订购号来修改或删除订购单中的条款。在 **modifyForm.jsp** 页面点击提交删除连接到 **deleteForm.jsp** 删除订单的相应条款；点击提交修改连接到 **changeForm.jsp** 修改订购数量。

地址: http://192.168.1.100:8080/modifyForm.jsp

书目浏览 | 用户注册 | 用户登陆 | 订购图书 | 修改定单 | 查看定单

你的定单:

用户名	姓名	订购号	书名	订购数量
EXP	张源	TP125	数据库技术	125
EXP	张源	QF127	英语阅读	1
EXP	jjana	TP124	java 2 应用教程	1

修改定单:

输入订购号: *

输入订购号: *

输入新数量: *

图 8.10 选择修改方式

选择修改方式的页面（效果如图 8.10 所示）

modifyForm.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="Login" %>

<jsp:useBean id="login" class="Login" scope="session" >

</jsp:useBean>

<% //如果客户直接进入该页面将被转向登录页面。

    if(session.isNew())

        {response.sendRedirect("userLogin.jsp");

        }

    // 如果没有成功登录将被转向登录页面

    String success=login.getSuccess();

        if(success==null)

            {success="";

            }

    if(!(success.equals("ok")))

        {response.sendRedirect("userLogin.jsp");
```

```

    }

%>

<HTML>

<BODY ><Font size=1>

<%@ include file="head.txt" %>

<P>你的订单:

<% String logname=login.getLogname();

    if(logname==null)

        {logname="";

        }

    Connection con;

    Statement sql;

    ResultSet rs;

    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

    catch(ClassNotFoundException e){}

    try{ con=DriverManager.getConnection("jdbc:odbc:shop","","");

        sql=con.createStatement();

        String condition="SELECT * FROM orderform WHERE logname =

"+logname+"";

        rs=sql.executeQuery(condition);

        out.print("<Table Border>");

        out.print("<TR>");

```

```

        out.print("<TH width=50>"+"<Font size=1>"+"      用户名");
        out.print("<TH width=50>"+"<Font size=1>"+"      姓名");
        out.print("<TH width=160>"+"<Font size=1>"+"      订购号");
        out.print("<TH width=160>"+"<Font size=1>"+"      书名");
        out.print("<TH width=60>"+"<Font size=1>"+"      订购数量");
        out.print("<TH width=60>"+"<Font size=1>"+"      电话");
        out.print("<TH width=160>"+"<Font size=1>"+"      地址");
        out.print("</TR>");

while(rs.next())
{
    out.print("<TR>");

        out.print("<TD >"+"<Font size=1>" +rs.getString(1)+"</TD>");
        out.print("<TD >"+"<Font size=1>" +rs.getString(2)+"</TD>");
        out.print("<TD >"+"<Font size=1>" +rs.getString(3)+"</TD>");
        out.print("<TD >"+"<Font size=1>" +rs.getString(4)+"</TD>");
        out.print("<TD >"+"<Font size=1>" +rs.getString(5)+"</TD>");
        out.print("<TD >"+"<Font size=1>" +rs.getString(6)+"</TD>");
        out.print("<TD >"+"<Font size=1>" +rs.getString(7)+"</TD>");

        out.print("</TR>" );

    }

    out.print("</Table>");

    con.close();

}

catch(SQLException e)

```



```

    { }

%>

<P>修改订单:

<%String str1=response.encodeURL("deletForm.jsp");

String str2=response.encodeURL("changeForm.jsp");

%>

<FORM action="<%=str1%>" method=post>

    <BR>    输入订购号<Input TYPE=text name=order_number >*

    <Input type=submit name="k" value="    提交删除">

</FORM>

<FORM action="<%=str2%>" method=post>

    <BR>    输入订购号<Input TYPE=text name=order_number >*

    <BR>    输入新定数<Input TYPE=text name=mount >*

    <Input type=submit name="p" value="    提交修改">

</FORM>

</BODY>

</HTML>

```

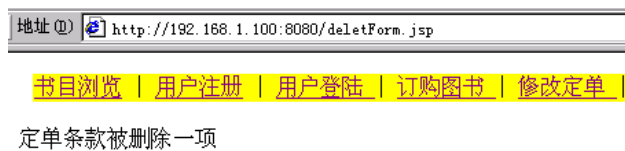


图 8.11 删除操作

删除订购单页面（效果如图 8.11 所示）

deleteForm.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="Login" %>

<jsp:useBean id="login" class="Login" scope="session" >

</jsp:useBean>

<html>

<BODY>

<% //如果客户直接进入该页面将被转向登录页面。

    if(session.isNew())

        {response.sendRedirect("userLogin.jsp");

        }

    // 如果没有成功登录将被转向登录页面

    String success=login.getSuccess();

        if(success==null)

            {success="";

            }

    if(!(success.equals("ok")))

        {response.sendRedirect("userLogin.jsp");
```

```

    }
%>
<% //获取订单号:

String order_number=request.getParameter("order_number");

    if(order_number==null)

        {order_number="";

        }

byte b[]=order_number.getBytes("ISO-8859-1");

order_number=new String(b);

Connection con=null;

Statement sql=null;

ResultSet rs=null;

    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

    catch(ClassNotFoundException e){}

    try

    {con=DriverManager.getConnection("jdbc:odbc:shop","","");

    sql=con.createStatement();

    String condition=

        "DELETE FROM orderform WHERE

order_number='"+order_number+"'";

    sql.executeUpdate(condition); //    删除。

    out.print("<BR>"+    订单条款被删除一项");

```

```

    }

    catch(SQLException e)

    { out.print("<BR>"+"    删除失败");

    }

%>

</BODY>

</HTML>

```

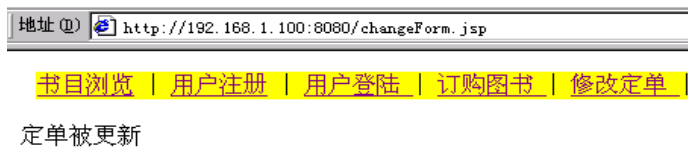


图 8.12 修改订购数量

修改订购数量页面（效果如图 8.12 所示）

changeForm.jsp

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="Login" %>

<jsp:useBean id="login" class="Login" scope="session" >

</jsp:useBean>

<html>

```

<BODY>

<%@ include file="head.txt" %>

<% //如果客户直接进入该页面将被转向登录页面。

```
if(session.isNew())  
  
{response.sendRedirect("userLogin.jsp");  
  
}
```

// 如果没有成功登录将被转向登录页面

```
String success=login.getSuccess();  
  
if(success==null)  
  
{success="";  
  
}  
  
if(!(success.equals("ok")))  
  
{response.sendRedirect("userLogin.jsp");  
  
}
```

%>

<% //获取订单号:

```
String order_number=request.getParameter("order_number");  
  
if(order_number==null)  
  
{order_number="";  
  
}
```

```
byte b[]=order_number.getBytes("ISO-8859-1");
```

```
order_number=new String(b);
```

// 获取新的定数:

```

String newMount=request.getParameter("mount");

    if(newMount==null)

        {newMount="0";

        }

    byte c[]=newMount.getBytes("ISO-8859-1");

    newMount=new String(c);

    Connection con=null;

    Statement sql=null;

    ResultSet rs=null;

    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        }

    catch(ClassNotFoundException e){}

    try

    {con=DriverManager.getConnection("jdbc:odbc:shop","","");

        sql=con.createStatement();

        String condition=

        "UPDATE orderform SET mount = "+newMount+" WHERE

order_number="+""+order_number+""";

        //    更新订单:

        sql.executeUpdate(condition);

        out.print("<BR>"+""    订单被更新");

    }

    catch(SQLException e)

```

```

        { out.print("<BR>"+"    更新失败");
        }

%>

</BODY>

</HTML>

```

8.4.6 浏览书目

用户可以在该页面分页浏览书目，同时可将准备订购的书添加到订单。

浏览书目页面所用的 beans

PageNumber.java:

```

public class PageNumber
{
    int rowCount=1, //    总的记录数。

    pageSize=1, //    每页显示的记录数。

    showPage=1, //    设置欲显示的页码数。

    pageCount=1; //    分页之后的总页数。

    public void setRowCount(int n)
    {
        rowCount=n;
    }

    public int getRowCount()
    {
        return rowCount;
    }

    public void setPageCount(int r,int p)
    {
        rowCount=r;
    }
}

```


浏览书目页面（效果如图 8.13 所示）

showBookList.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="Login" %>

<%@ page import="PageNumber" %>

<%@ page import="java.io.*" %>

<jsp:useBean id="handlePage" class="PageNumber" scope="session" >

</jsp:useBean>

<jsp:useBean id="login" class="Login" scope="session" >

</jsp:useBean>

<% //如果客户直接进入该页面将被转向登录页面。

    if(session.isNew())

        {response.sendRedirect("userLogin.jsp");

        }

    // 如果没有成功登录将被转向登录页面
```

```

String success=login.getSuccess();

    if(success==null)

        {success="";

        }

    if(!(success.equals("ok")))

        {response.sendRedirect("userLogin.jsp");

        }

%>

<HTML>

<BODY ><Font size=1>

<%@ include file="head.txt" %>

<P> 图书目录:

<%! // 声明一个共享的连接对象:

    Connection con=null;

    // 显示数据库记录的方法:

    public void showList(ResultSet rs,javax.servlet.jsp.JspWriter out,int n,String
buybook)

    {try

    {

        out.print("<Table Border>");

        out.print("<TR>");

        out.print("<TH width=50>"+<Font size=1>"+id"+</FONT>");

        out.print("<TH width=50>"+<Font size=1>"+      订购号"+</FONT>");

```

```

        out.print("<TH width=70>"+ "<Font size=1>"+ "
        书名"+ "</FONT>");
        out.print("<TH width=60>"+ "<Font size=1>"+ "
        作者"+ "</FONT>");
        out.print("<TH width=60>"+ "<Font size=1>"+ "
        出版社"+ "</FONT>");
        out.print("<TH width=50>"+ "<Font size=1>"+ "
        出 版 时 间
        "+ "</FONT>");

        out.print("<TH width=20>"+ "<Font size=1>"+ "
        价钱"+ "</FONT>");
        out.print("<TH width=50>"+ "<Font size=1>"+ "
        分类"+ "</FONT>");
        out.print("<TH width=50>"+ "<Font size=1>"+ "
        添 加 到 订 单
        "+ "</FONT>");

        out.print("</TR>");
        for(int i=1;i<=n;i++)
        { out.print("<TR>");

            String id=rs.getString(1);// 图书 id 号。

            out.print("<TD >"+id+"</TD>");

            out.print("<TD >"+rs.getString(2)+"</TD>");

            out.print("<TD >"+rs.getString(3)+"</TD>");

            out.print("<TD >"+rs.getString(4)+"</TD>");

            out.print("<TD >"+rs.getString(5)+"</TD>");

            out.print("<TD >"+rs.getDate(6)+"</TD>");

            out.print("<TD >"+rs.getString(7)+"</TD>");

            out.print("<TD >"+rs.getString(8)+"</TD>");

            // 在本书的的后面显示一个表单，该表单将内容提交到 buybook.jsp,
            // 将 id 号提交给 buybook.jsp,以便订购该书:

```

```

String s1="<Form action="+buybook+" method=get>";

String s2="<input type=hidden name=id value="+id+">";

String s3="<input type=submit value=      订购></FORM> ";

String s=s1+s2+s3;

out.print("<TD >" +s+ "</TD>");

out.print("</TR>") ;

rs.next();

}

out.print("</Table>");

}

catch(Exception e1) {}

}

%>

<% Statement sql=null;

ResultSet rs=null;

int rowCount=0; // 总的记录数。

// 第一个客户负责建立连接对象：

if(con==null)

{ try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

}

catch(ClassNotFoundException e)

{out.print(e);

}

```

```

try

    {con=DriverManager.getConnection("jdbc:odbc:shop","", "");

    sql=

con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);

    rs=sql.executeQuery("SELECT * FROM book");//      返回可滚动的结果集。

    rs.last();//      将游标移动到最后一行。

    int number=rs.getRow();//      获取最后一行的行号。

    rowCount=number; //      获取记录数。

    handlePage.setPageSize(3); //      设置每页显示的记录数。

    handlePage.setPageCount(rowCount,handlePage.getPageSize());//      计算总
页数。

    out.print("      共有"+handlePage.getPageCount()+"页， ");
    out.print("      每页显示"+ handlePage.getPageSize()+"条记录");
    }

catch(SQLException e)

    {out.print(e);

    }

}

// 其它客户通过同步块使用这个连接：

else

    { synchronized(con)

```

```
{ try { sql=
```

```
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
```

```
rs=sql.executeQuery("SELECT * FROM book"); // 返回可滚动的结果集。
```

```
rs.last(); // 将游标移动到最后一行。
```

```
int number=rs.getRow();// 获取最后一行的行号。
```

```
rowCount=number; // 获取记录数。
```

```
handlePage.setPageSize(3); // 设置每页显示的记录数。
```

```
handlePage.setPageCount(rowCount,handlePage.getPageSize());// 计算总页数。
```

```
out.print(" 共有"+handlePage.getPageCount()+"页， ");
```

```
out.print(" 每页显示"+ handlePage.getPageSize()+"条记录");
```

```
}
```

```
catch(SQLException e)
```

```
{out.print(e);
```

```
}
```

```
}
```

```
}
```

```
%>
```

```
<%-- 选择显示某页的表单 --%>
```

```
<%String str=response.encodeURL("showBookList.jsp");
```

```

String buybook=response.encodeURL("buybook.jsp");

%>

<Form action="<%=str%>" method="post" >

    显示下一页: <Input Type="hidden" name="a" value="next">

    <Input type=submit value="next">

</FORM>

<Form action="<%=str%>" method="post" >

    显示上一页: <Input Type="hidden" name="a" value="previous">

    <Input type="submit" value="previous">

</FORM>

<Form action="<%=str%>" method="post" >

    输入欲要显示的页<Input type="text" name="a" value="1">

    <Input type="submit" value="submit">

</FORM>

<% // 获取表单提交的信息:

    String s=request.getParameter("a");

    if(s==null)

        {s="1";

        }

    if(s.equals("next"))

        { int n=handlePage.getShowPage(); //      获取目前的页数。

          n=(n+1); //      将页数增 1。

          if(n>handlePage.getPageCount())

```

```

        { n=1;

        }

        handlePage.setShowPage(n);           //           显示该页。

        out.print("           目前显示第"+handlePage.getShowPage()+"页");

        //           将游标移到:

        rs.absolute((n-1)*handlePage.getPageSize()+1);

        showList(rs,out,handlePage.getPageSize(),buybook); //           显示第该页的内容。

    }

    else if(s.equals("previous"))

    { int n=handlePage.getShowPage(); //           获取目前的页数。

      n=(n-1); //           将页数减 1。

      if(n<=0)

      { n=handlePage.getPageCount();

      }

      handlePage.setShowPage(n);           //           显示该页。

      out.print("           目前显示第"+handlePage.getShowPage()+"页");

      rs.absolute((n-1)*handlePage.getPageSize()+1); //           移动游标。

      showList(rs,out,handlePage.getPageSize(),buybook); //           显示第该页的内容。

    }

    else

    { int m=Integer.parseInt(s);

      handlePage.setShowPage(m);

      out.print("           目前显示第"+handlePage.getShowPage()+"页");

```



```

        int n=handlePage.getShowPage();

        rs.absolute((n-1)*handlePage.getPageSize()+1); //      移动游标。

        showList(rs,out,handlePage.getPageSize(),buybook); //      显示该页的内容。
    }

    %>

</FONT>

</BODY>

</HTML>

```

8.4.7 修改密码

输入用户名和正确密码可以修改密码。

修改密码页面（效果如图 8.14 所示）

modifyPassword.jsp:

```

<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.sql.*" %>

<%@ page import="Login" %>

<jsp:useBean id="login" class="Login" scope="session" >

</jsp:useBean>

<% //如果客户直接进入该页面将被转向登录页面。

    if(session.isNew())

        {response.sendRedirect("userLogin.jsp");

        }

    // 如果没有成功登录将被转向登录页面

```

```

String success=login.getSuccess();

    if(success==null)

        {success="";

        }

    if(!(success.equals("ok")))

        {response.sendRedirect("userLogin.jsp");

        }

%>

<HTML>

<BODY bgcolor=pink ><Font size=1>

<%@ include file="head.txt" %>

<P>修改密码,密码长度不能超过 30 个字符:

<%String str=response.encodeURL("modifyPassword.jsp");

%>

<FORM action="<%=str%>" Method="post">

<BR>输入您的用户名:

<BR><Input type=text name="logname" value="<%=login.getLogname()%>" >

<BR>输入您的密码:

<BR><Input type=password name="password">

<BR>输入您的新的密码:

<BR><Input type=text name="newPassword1">

<BR>请再输入一次新密码:

<BR><Input type=text name="newPassword2">

```


<Input type=submit name="g" value="提交">

</FORM>

<%!//处理字符串的一个常用方法:

```
public String getString(String s)
{ if(s==null) s="";
  try {byte a[]=s.getBytes("ISO-8859-1");
      s=new String(a);
  }
  catch(Exception e)
  { }
  return s;
}
```

%>

<%

String logname=request.getParameter("logname");// 获取提交的用户名。

logname=getString(logname);

String password=request.getParameter("password");// 获取提交的密码。

password=getString(password);

String newPassword1=request.getParameter("newPassword1");// 获取提交的新
密码 1。

newPassword1=getString(newPassword1);

String newPassword2=request.getParameter("newPassword2");// 获取提交的新
密码 2。

```

        newPassword2=getString(newPassword2);

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

catch(ClassNotFoundException event){}

// 验证身份:

Connection con=null;

Statement sql=null;

boolean modify=false;

boolean ifEquals=false;

ifEquals=(newPassword1.equals(newPassword2))&&(newPassword1.length()<=30);

if(ifEquals==true)

    { try{ con=DriverManager.getConnection("jdbc:odbc:shop","","");

        sql=con.createStatement();

        boolean bo1=logname.equals(login.getLogname()),

            bo2=password.equals(login.getPassword());

        if(bo1&&bo2)

            {//          修改密码:

            modify=true;

            out.print("          您的密码已经更新");

            String c="UPDATE user SET password =

"+"""+newPassword1+"""+

            " WHERE logname = "+"""+logname+""";

```

```

        sql.executeUpdate(c);

    }

    con.close();

}

catch(SQLException e1) {}

}

else

{ out.print("    你两次输入的密码不一致或长度过大");

}

if(modify==false&&ifEquals==true)

{ out.print("<BR> 您没有输入密码帐号或<BR>您输入的帐号或密码不正确

"+logname+": "+password);

}

%>

</FONT>

</BODY>

</HTML>

```

图 8.14 修改密码

8.4.8 修改个人信息

输入用户名和正确密码可以修改除密码和用户名以外的个人信息。

图 8.15 修改信息

修改个人信息页面（效果如图 8.15 所示）

modifyMessage.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import="java.sql.*" %>
```

```

<%@ page import="Login" %>

<jsp:useBean id="login" class="Login" scope="session" >

</jsp:useBean>

<% //如果客户直接进入该页面将被转向登录页面。

    if(session.isNew())

        {response.sendRedirect("userLogin.jsp");

        }

    // 如果没有成功登录将被转向登录页面

    String success=login.getSuccess();

        if(success==null)

            {success="";

            }

        if(!(success.equals("ok")))

            {response.sendRedirect("userLogin.jsp");

            }

%>

<HTML>

<BODY bgcolor=pink ><Font size=1>

<%@ include file="head.txt" %>

<%String str=response.encodeURL("modifyMessage.jsp");

%>

<P>修改您的基本信息：真实姓名、电子邮件、电话、邮寄地址。

<FORM action="<%=str%>" Method="post">

```


输入您的用户名:

<Input type=text name="logname" value="<%=login.getLogname()%>" >

输入您的密码:

<Input type=password name="password">

输入新的姓名<Input type=text name="realname" >

输入新的 email<Input type=text name="email" >

输入新的电话<Input type=text name="phone" >

输入新的地址<Input type=text name="address" >

<Input type=submit name="g" value="提交">

</FORM>

<%!//处理字符串的一个常用方法:

```
public String getString(String s)
{
    if(s==null) s="";

    try {byte a[]=s.getBytes("ISO-8859-1");

        s=new String(a);

    }

    catch(Exception e)

    { }

    return s;

}

%>
```

<%= String logname=request.getParameter("logname"); // 获取提交的用户名。

logname=getString(logname);


```

String password=request.getParameter("password"); //    获取提交的密码。

    password=getString(password);

String realname=request.getParameter("realname"); //    获取新姓名。

    realname=getString(realname);

String email=request.getParameter("email"); //    获取新 email:

    email=getString(email);

String phone=request.getParameter("phone"); //    获取新电话:

    phone=getString(phone);

String address=request.getParameter("address"); //    获取新地址:

    address=getString(address);

    try{Class.forName('sun.jdbc.odbc.JdbcOdbcDriver');

        }

    catch(ClassNotFoundException event){}

// 验证身份:

Connection con=null;

Statement sql=null;

boolean modify=false;

try{ con=DriverManager.getConnection("jdbc:odbc:shop","","");

    sql=con.createStatement();

    boolean bo1=logname.equals(login.getLogname()),

        bo2=password.equals(login.getPassword());

    if(bo1&&bo2)

        {//            修改信息:

```

```

        modify=true;

        out.print("<BR>        您的信息已经更新");

        String c1="UPDATE user SET realname = '"+realname+"'+"
            " WHERE logname = '"+logname+"'";

        String c2="UPDATE user SET email = '"+email+"'+"
            " WHERE logname = '"+logname+"'";

        String c3="UPDATE user SET phone = '"+phone+"'+"
            " WHERE logname = '"+logname+"'";

        String c4="UPDATE user SET address = '"+address+"'+"
            " WHERE logname = '"+logname+"'";

        sql.executeUpdate(c1);

        sql.executeUpdate(c2);

        sql.executeUpdate(c3);

        sql.executeUpdate(c4);

    }

    else

        {out.print("<BR>        您还没有输入密码或您输入的密码或用户名有错误");

        }

    con.close();

}

catch(SQLException e1)

    { out.print("<BR>        更新失败");

    }

```

%>

</BODY>

</HTML>

第9章 Java Servlet

我们已经知道，SUN 公司以 **Java Servlet** 为基础，推出了 **Java Server Page**。JSP 提供了 **Java Servlet** 的几乎所有好处，当一个客户请求一个 JSP 页面时，JSP 引擎根据 JSP 页面生成一个 Java 文件，即一个 **servlet**。这一章，将对 **servlet** 做一个较详细的介绍，这不仅对于深刻理解 JSP 有一定的帮助，而且通过学习 **servlet**，还能使我们选择使用 **JSP+javaBeans+servlet** 的模式来开发我们的 Web 应用程序。

我们已经知道，用 JSP 支持 **JavaBeans** 这一特点，可以有效地管理页面的静态部分和页面的动态部分。另外，我们也可以在一個 JSP 页面中调用一个 **servlet** 完成动态数据的处理，而让 JSP 页面本身处理静态的信息。因此，开发一个 Web 应用有两种模式可以选择：

(1) **JSP+javaBeans**

(2) **JSP+javaBeans+servlet**

9.1 Servlet 工作原理

servlet 由支持 **servlet** 的服务器：**servlet** 引擎，负责管理运行。当多个客户请求一个 **servlet** 时，引擎为每个客户启动一个线程而不是启动一个进程，这些线程由 **servlet** 引擎服务器来管理，与传统的 CGI 为每个客户启动一个进程相比较，效率要高的多。

9.1.1 Servlet 的生命周期

学习过 Java 语言的人对 **Java Applet**(Java 小应用程序)都很熟悉，一个 **Java Applet** 是 **java.applet.Applet** 类的子类，该子类的对象由客户端的浏览器负责初始化和运行。**servlet** 的运行机制和 **Applet** 类似，只不过它运行在服务器端。一个 **servlet** 是 **javax.servlet**

包中 **HttpServlet** 类的子类，由支持 **servlet** 的服务器完成该子类的对象，即 **servlet** 的初始化。

Servlet 的生命周期主要有下列三个过程组成：

- (1) 初始化 **servlet**。**servlet** 第一次被请求加载时，服务器初始化这个 **servlet**，即创建一个 **servlet** 对象，这对象调用 **init** 方法完成必要的初始化工作。
- (2) 诞生的 **servlet** 对象再调用 **service** 方法响应客户的请求。
- (3) 当服务器关闭时，调用 **destroy** 方法，消灭 **servlet** 对象。

init 方法只被调用一次，即在 **servlet** 第一次被请求加载时调用该方法。当后续的客户请求 **servlet** 服务时，Web 服务将启动一个新的线程，在该线程中，**servlet** 调用 **service** 方法响应客户的请求，也就是说，每个客户的每次请求都导致 **service** 方法被调用执行。

9.1.2 **init** 方法：

该方法是 **HttpServlet** 类中的方法，我们可以在 **servlet** 中重写这个方法。

方法描述：

```
public void init(ServletConfig config) throws ServletException
```

servlet 第一次被请求加载时，服务器初始化一个 **servlet**，即创建一个 **servlet** 对象，这个对象调用 **init** 方法完成必要的初始化工作。该方法在执行时，**servlet** 引擎会把一个 **ServletConfig** 类型的对象传递给 **init()** 方法，这个对象就被保存在 **servlet** 对象中，直到 **servlet** 对象被消灭，这个 **ServletConfig** 对象负责向 **servlet** 传递服务设置信息，如果传递失败就会发生 **ServletException**，**servlet** 就不能正常工作。

我们已经知道，当多个客户请求一个 **servlet** 时，引擎为每个客户启动一个线程，那么 **servlet** 类的成员变量被所有的线程共享。

9.1.3 service 方法

该方法是 **HttpServlet** 类中的方法，我们可以在 **servlet** 中直接继承该方法或重写这个方法。

方法描述：

```
public void service(HttpServletRequest request HttpServletResponse response)
```

```
throw
```

```
ServletException,IOException
```

当 **servlet** 成功创建和初始化之后，**servlet** 就调用 **service** 方法来处理用户的请求并返回响应。**Servlet** 引擎将两个参数传递给该方法，一个 **HttpServletRequest** 类型的对象，该对象封装了用户的请求信息，此对象调用相应的方法可以获取封装的信息，即使用这个对象可以获取用户提交的信息。另外一个参数对象是 **HttpServletResponse** 类型的对象，该对象用来响应用户的请求。和 **init** 方法不同的是，**init** 方法只被调用一次，而 **service** 方法可能被多次的调用，我们已经知道，当后续的客户请求 **servlet** 服务时，**Servlet** 引擎将启动一个新的线程，在该线程中，**servlet** 调用 **service** 方法响应客户的请求，也就是说，每个客户的每次请求都导致 **service** 方法被调用执行，调用过程运行在不同的线程中，互不干扰。

9.1.4 destroy 方法

该方法是 **HttpServlet** 类中的方法。**servlet** 可直接继承这个方法，一般不需要重写。

方法描述：

public destroy()

当 Servlet 引擎终止服务时，比如关闭服务器等，**destroy()**方法会被执行，消灭 servlet 对象。

9.2 编译和安装 servlet

9.2.1 简单的 servlet 例子

在下面的例子 1 中，**Hello** 扩展了 **HttpServlet**。

例子 1

servlet 源文件

Hello.java:

```
import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class Hello extends HttpServlet

{ public void init(ServletConfig config) throws ServletException

    {super.init(config);

    }

    public void service(HttpServletRequest request,HttpServletResponse response)

throws IOException

    { // 获得一个向客户发送数据的输出流:

        PrintWriter out=response.getWriter();
```

```
response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME  
类型。
```

```
out.println("<HTML> <BODY>");  
  
out.println("Simple servlet");  
  
out.println("</body> </html>");  
  
}  
  
}
```

9.2.2 编译 servlet

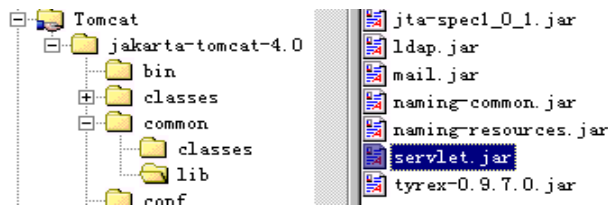


图 9.1 编译 servlet 所需要的 jar 文件

为了编译 servlet 源文件，需要 `HttpServlet`、`HttpServletRequest` 等类，JDK 内置包中并不包含这些类文件。为了能编译 servlet 源文件得到创建 servlet 用的字节码文件，需要在环境变量中包含 `servlet.jar`，这个 jar 文件在 TOMCAT 安装目录的 `common\lib` 文件下，如图 9.1 所示（也可以到 sun 公司网站下载编译 servlet 所需要的类）。

对于 window2000，用鼠标右键点击“我的电脑”，弹出菜单，然后选择属性，弹出“系统特性”对话框，再单击该对话框中的高级选项，然后点击按钮“环境变量”，编辑 classpath，添加新的环境变量的值：

D:\Tomcat\jakarta-tomcat-4.0\common\lib\servlet.jar;

对于 Win9x, 用记事本编辑 Autoexec.bat 文件, 将新的值:

D:\Tomcat\jakarta-tomcat-4.0\common\lib\servlet.jar;

加入即可(或将 **servlet.jar** 文件拷贝到 **jdk** 安装目录的 **jre\lib\ext** 中)。

该环境变量值也可以即设即用, 在 **DOS** 命令行键入:

set classpath=D:\Tomcat\jakarta-tomcat-4.0\common\lib\servlet.jar

回车确认即可。

我们将上述 **servlet** 的源文件 **Hello.java** 保存到 **F:\2000**, 然后编译生成字节码文件 **Hello.class**

9.2.3 存放 **servlet** 的目录

(1) 所有 **web** 服务目录可使用的 **servlet** 的存放位置

如果让所有 **web** 服务目录都可以使用该 **servlet**, 那么创建这个 **servlet** 的字节码文件需存放在 **Tomcat** 安装目录的 **classes** 目录中, 例如, 本书所用机器的目录就是:
D:\tomcat\Jakarta-tomcat-4.0\classes, 如图 9.1 所示。

我们已经知道, **servlet** 第一次被请求加载时, 服务器初始化一个 **servlet**, 即创建一个 **servlet** 对象, 这对象调用 **init** 方法完成必要的初始化工作。如果你对 **servlet** 的源文件进行了修改, 并将新的字节码文件存放到 **classes** 中, 如果服务器没有关闭的话, 新的 **servlet** 不会被创建, 因为, 当后续客户请求 **servlet** 服务时, 已初始化的 **servlet** 将调用 **service**

方法响应客户。

(2) 只对 **examples** 服务目录可用的 **secler** 的存放目录

examples 是 **TOMCAT** 引擎的默认 **web** 服务目录之一。

如果想让某个 **servlet** 只对 **examples** 目录可用，那么创建该 **servlet** 的字节码文件只需存放在 **webapps/example/Web-inf/classes** 目录中。

存放在该目录中的 **servlet** 和存放在上面 (1) 中所述目录中的 **servlet** 有所不同，服务器引擎首先检查 **webapps/example/Web-inf/classes** 目录中的创建该 **servlet** 的字节码文件是否被修改过，如果重新修改过，就会用消灭 **servlet**，用新的字节码重新初始化 **servlet**。

如果经常调试 **servlet**，可以把 **servlet** 放在 **webapps/example/Web-inf/classes**。需要注意的是，当用户请求 **servlet** 服务时，由于服务器引擎每次都要检查字节码文件是否被修改过，导致服务器的运行效率降低。

9.2.4 运行 **servlet**

如果一个 **servlet** 对所有的 **web** 服务目录可用，那么只要在服务器引擎启动后，在浏览器地址栏键入：

http://localhost:8080/web 服务目录/servlet/创建 servlet 类的名字

即可，例如，对于用上述 **Hello** 创建的 **servlet**，

(1) **Root** 服务目录

http://localhost:8080/servlet/Hello

(2) **friend** 目录（我们自定义的一个 **web** 服务目录）

http://localhost:8080/friend/servlet/Hello

如果是只对 **examples** 服务目录可用的 **servelt**，那么只要在服务器引擎启动后，在浏览器地址栏键入：

http://localhost:8080/examples/servlet/创建 servlet 类的名字

我们将 **Hello.class** 文件保存到 Tomcat 引擎的 **classes** 文件夹中。图 9.2 和 9.3 是在不同的 **web** 目录下运行 **servlet** 的效果。

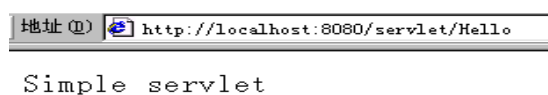


图 9.2 用 Web 根目录访问 servlet

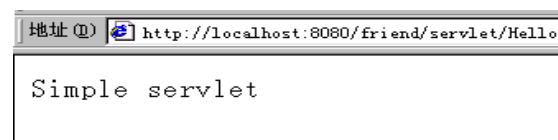


图 9.3 用 friend 目录访问

9.2.5 带包名的 servlet

在写一个 servlet 的 java 文件时，可以使用 `package` 语句给 servlet 一个包名。包名可以是一个合法的标识符，也可以是若干个标识符加“.”分割而成，如：

```
package gping;
```

```
package tom.jiafei;
```

程序如果使用了包语句，例如

```
package tom.jiafei;
```

那么在 `classes` 目录下需有如下 的子目录，例如，在 `D:\Tomcat\jakarta-tomcat-4.0\classes` 下再建立如下的目录结构。

```
\tom\jiafei
```

并将 servlet 的字节码文件存在该目录中，如图 6.15 所示。

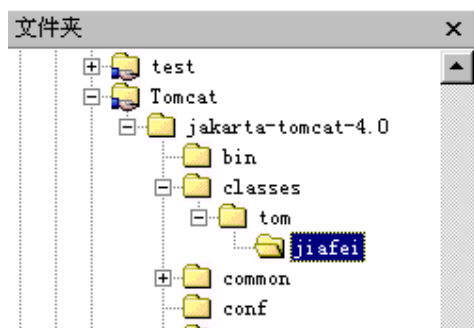


图 9.4 存放带包名的 servlet

如果 **servlet** 有包名，比如，**Hello** 的包名是 **tom.jiafei**，那么调用该 **servlet** 的 URL 是：

http://localhost:8080/web 服务目录/servlet/tom.jiafei.Hello

因为起了包名，**Hello** 的全名是 **tom.jiafei.Hello**（就好比大连的全名是：中国.辽宁.大连）。

9.3 通过 JSP 页面调用 servlet

9.3.1 通过表单向 servlet 提交数据

任何一个 Web 服务目录下的 JSP 页面都可以通过表单或超链接访问某个 **servlet**。通过 JSP 页面访问 **servlet** 的好处是，JSP 页面可以负责页面的静态信息处理，动态信息处理交给 **servlet** 去完成。

在下面的例子中，JSP 页面通过表单向 **servlet** 提交一个正实数，**servlet** 负责计算这个数的平方根返回给客户。

为了方便地调试 **servlet**，本书中，**servlet** 的字节码文件存放在 **D:\Tomcat\jakarta-tomcat-4.0\webapps\example\Web-inf\classes** 中，那么在 JSP 页面中调用 **servlet** 时，**servlet** 的 URL 是：

/examples/servlet/servletName

在下面的例子 2 中，JSP 页面通过表单提交一个正数，servlet 负责计算这个数的平方根。

例子 2

调用 servlet 的页面(该页面存放在 web 服务的根目录 Root 中)

givenumber.jsp(效果如图 9.5 所示)

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<HTML>
```

```
<BODY bgcolor=cyan><Font size=1>
```

```
<P>输入一个数，servlet 求这个数的平方根：
```

```
<FORM action="examples/servlet/Sqrt" method=get>
```

```
<Input Type=text name=number>
```

```
<Input Type=submit value=" 提交">
```

```
</FORM>
```

```
</BODY>
```

```
</HTML>
```



图 9.5 提交数字的 JSP 页面

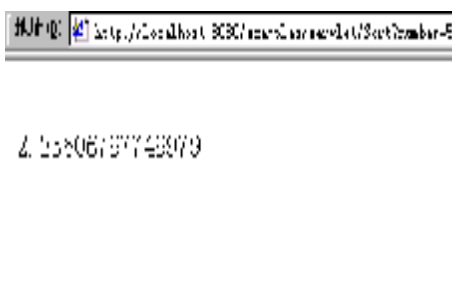


图 9.6 负责计算平方根的 servlet

servlet 源文件(效果如图 9.6 所示)

Sqrt.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Sqrt extends HttpServlet

{ public void init(ServletConfig config) throws ServletException

    {super.init(config);

    }

    public void service(HttpServletRequest request,HttpServletResponse response)
```

throws IOException

```
{ // 获得一个向客户发送数据的输出流:
```

```
    PrintWriter out=response.getWriter();
```

```
    response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME
```

类型。

```
    out.println("<HTML> <BODY>");
```

```
    String number=request.getParameter("number"); // 获取客户提交的信
```

息。

```
double n=0;

try{ n=Double.parseDouble(number);

    out.print("<BR>" + Math.sqrt(n));

}

catch(NumberFormatException e)

    { out.print("<H1>input number letter please! </H1>");

    }

out.println("</body> </html>");

}

}
```

9.3.2 通过超链接访问 servlet

我们可以在 JSP 页面中，点击一个超链接，访问 servlet。

例子 3

connection.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><Font size=1>

<A Href="/servlet/Hello" >加载 servlet<A>

</BODY>

</HTML>
```


9.4 servlet 的共享变量

我们已经知道，在 `servlet` 被加载之后，当后续的客户请求 `servlet` 服务时，引擎将启动一个新的线程，在该线程中，`servlet` 调用 `service` 方法响应客户的请求，而且 `servlet` 类中定义的成员变量，被所有的客户线程共享。在下面的例子 4 中，利用共享变量实现了一个计数器。

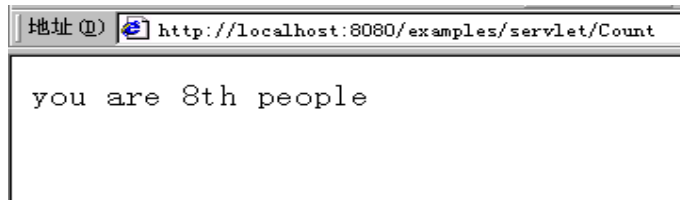


图 9.7 计数器

例子 4(效果如图 9.7 所示)

Count.java:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Count extends HttpServlet
{
    int count;

    public void init(ServletConfig config) throws ServletException
    {
        super.init(config);
```

```

    count=0;

}

public synchronized void service(HttpServletRequest request,HttpServletResponse
response)

        throws IOException

{ // 获得一个向客户发送数据的输出流:

    PrintWriter out=response.getWriter();

    response.setContentType("text/html;charset=GB2312");//    设置响应的 MIME
类型。

    out.println("<HTML> <BODY>");

    count++;                //                增加计数。

    out.println("you are "+count+"th"+" people");

    out.println("</body> </html>");

}

}

```

注:在处理多线程问题时,我们必须注意这样一个问题:当两个或多个线程同时访问同一个变量,并且一个线程需要修改这个变量。我们应对这样的问题作出处理,否则可能发生混乱.所以上述例子中的 service 方法是一个 **synchronized** 方法。

数学上有一个计算 p 的公式:

$$\frac{p}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}\dots\dots$$

下面的例子中利用成员变量被所有客户共享这一特性实现客户帮助计算 p 的值,即每当客户请求访问 **servlet** 时都参与了一次 p 的计算。

客户通过点击一个 **JSP** 页面的超链接访问一个计算 p 的 **servlet**

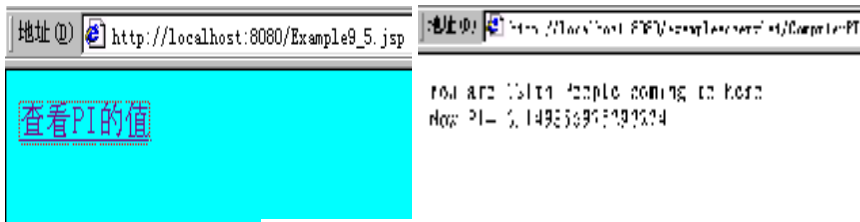


图 9.8 用 servlet 计算 PI

例子 5(效果如图 9.8 所示)

JSP 页面

Example9_5.jsp:

```
<%@ page contentType="text/html; charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan>

<A Href="examples/servlet/ComputerPI" >查看 PI 的值<A>

</BODY>

</HTML>
```

servlet 源文件

```
import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;
```

```

public class ComputerPI extends HttpServlet

{ double sum=0,i=1,j=1;

  int number=0;

  public void init(ServletConfig config) throws ServletException

  {super.init(config);

  }

  public synchronized void service(HttpServletRequest
request,HttpServletResponse response)

      throws IOException

  { // 获得一个向客户发送数据的输出流:

    PrintWriter out=response.getWriter();

    response.setContentType("text/plain");// 设置响应的 MIME 类型为纯文本。

    number++;

    sum=sum+i/j;

    j=j+2;

    i=-i;

    out.println("You are "+number+"th People coming to here");

    out.println("Now PI= "+4*sum);

  }

}

```

9.5 HttpServlet 类

9.5.1 doGet 方法和 doPost 方法

HttpServlet 除了 **init**、**service**、**destroy** 方法外，该类还有两个很重要的方法：**doGet** 和 **doPost**，用来处理客户的请求并作出响应。

当服务器引擎第一次接受到一个 **servlet** 请求时，会使用 **init** 方法初始化一个 **servlet**，以后每当服务器再接受到一个 **servlet** 请求时，就会产生一个新线程，并在这个线程中调用 **service** 方法检查 HTTP 请求类型（**Get**、**Post** 等），并在 **service** 方法中根据用户的请求方式，对应地再调用 **doGet** 或 **doPost** 方法。因此，在 **servlet** 类中，我们不必重写 **service** 方法来响应客户，直接继承 **service** 方法即可。我们可以在 **servlet** 类中重写 **doPost** 或 **doGet** 方法来响应用户的请求，这样可以增加响应的灵活性，并降低服务器的负担。

如果不论用户请求类型是 **Post** 还是 **Get**，服务器的处理过程完全相同，那么我们可以只在 **doPost** 方法中编写处理过程，而在 **doGet** 方法中再调用 **doPost** 方法即可，或只在 **doGet** 方法中编写处理过程，而在 **doPost** 方法中再调用 **doGet** 方法（见例子 6）。

如果根据请求的类型进行不同的处理，就需在两个方法中编写不同的处理过程（见例子 7）。

在下面的例子 6 中，用户可以通过两个表单向 **servlet** 提交一个正数，其中一个表单的提交方式是 **post**，另一个表单的方式是 **get**。无论用户用那种方式，服务器的 **servlet** 都计算这个数的全部因数，返回给用户。而在下面的例子 7 中，如果使用 **post** 方式提交正数，**servlet** 计算这个数的全部因数，如果使用 **get** 方式，**servlet** 求出小于这个数的全部素数。

例子 6(效果如图 9.9 所示)

提交正数的 JSP 页面

Example9_6.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><Font size=1>

<P>输入一个数，提交给 servlet(Post 方式):

<FORM action="examples/servlet/ComputerFactor" method=post>

    <Input Type=text name=number>

    <Input Type=submit value=" 提交">

</FORM>

<P>输入一个数，提交给 servlet(Get 方式):

<FORM action="examples/servlet/ComputerFactor" method=get>

    <Input Type=text name=number>

    <Input Type=submit value=" 提交">

</FORM>

</BODY>

</HTML>
```

servlet 源文件

ComputerFactor.java:

```
import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class ComputerFactor extends HttpServlet

{
```

```
public void init(ServletConfig config) throws ServletException
```

```
{super.init(config);
```

```
}
```

```
public void doPost(HttpServletRequest request,HttpServletResponse response)
```

```
throws ServletException,IOException
```

```
{ // 获得一个向客户发送数据的输出流:
```

```
PrintWriter out=response.getWriter();
```

```
response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME
```

类型。

```
out.println("<HTML>");
```

```
out.println("<BODY>");
```

```
String number=request.getParameter("number"); // 获取客户提交的信息。
```

```
double n=0;
```

```
try{ n=Double.parseDouble(number);
```

```
out.println("<H1> factors of "+n+" :</H1>");
```

```
// 求 n 的全部因数:
```

```
for(int i=1;i<=n;i++)
```

```
{ if(n%i==0)
```

```
out.println(i);
```

```
}
```

```
}
```

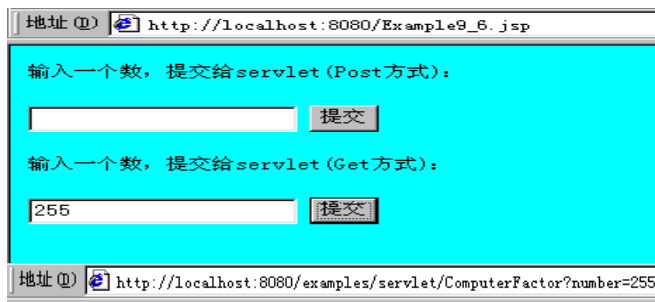
```
catch(NumberFormatException e)
```

```

        { out.print("<H1>input number letter please! </H1>");
        }
    }

    public void doGet(HttpServletRequest request,HttpServletResponse response)
        throws ServletException,IOException
    {
        doPost(request,response);
    }
}

```



factors of 255.0 :

1 3 5 15 17 51 85 255

图 9.9 Post、Get 处理方式相同

例子 7(效果如图 9.10 所示)

提交正数的 JSP 页面

Example9_7.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><Font size=1>

<P>输入一个数，提交给 servlet(Post 方式):

<FORM action="examples/servlet/ComputerFactorandPrimNumber" method=post>

  <Input Type=text name=number>

  <Input Type=submit value=" 提交">

</FORM>

<P>输入一个数，提交给 servlet(Get 方式):

<FORM action="examples/servlet/ComputerFactorandPrimNumber" method=get>

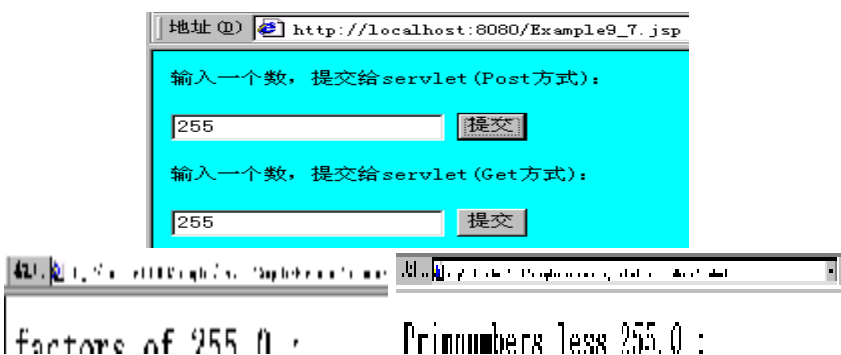
  <Input Type=text name=number>

  <Input Type=submit value=" 提交">

</FORM>

</BODY>

</HTML>
```



sevlet 源文件

ComputerFacorandPrimNumber.java:

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class ComputerFactorandPrimNumber extends HttpServlet

{

public void init(ServletConfig config) throws ServletException

{super.init(config);

}

public void doPost(HttpServletRequest request,HttpServletResponse response)

throws ServletException,IOException

```
{ // 获得一个向客户发送数据的输出流:
```

```
    PrintWriter out=response.getWriter();
```

```
    response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME  
    类型。
```

```
    out.println("<HTML>");
```

```
    out.println("<BODY>");
```

```
    String number=request.getParameter("number"); // 获取客户提交的信息。
```

```
    double n=0;
```

```
    try{ n=Double.parseDouble(number);
```

```
        out.println("<H1> factors of "+n+" :</H1>");
```

```
        // 求 n 的全部因数:
```

```
        for(int i=1;i<=n;i++)
```

```
            { if(n%i==0)
```

```
                out.println(i);
```

```
            }
```

```
        }
```

```
    catch(NumberFormatException e)
```

```
        { out.print("<H1>input number letter please! </H1>");
```

```
        }
```

```
    }
```

```
public void doGet(HttpServletRequest request,HttpServletResponse response)
```

throws ServletException,IOException

```
{ PrintWriter out=response.getWriter();
```

```
response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME
```

类型。

```
out.println("<HTML>");
```

```
out.println("<BODY>");
```

```
String number=request.getParameter("number"); // 获取客户提交的信息。
```

```
double n=0;
```

```
try{ n=Double.parseDouble(number);
```

```
out.println("<H1> Primnumbers less "+n+" :</H1>");
```

```
// 求小于 n 的全部素数:
```

```
int j=1;
```

```
for(int i=1;i<n;i++)
```

```
{ for(j=2;j<i;j++)
```

```
{if(i%j==0)
```

```
break;
```

```
}
```

```
if(j>=i)
```

```
{ out.println(i);
```

```
}
```

```
}
```

```
}
```

```

        catch(NumberFormatException e)

            { out.print("<H1>input number letter please! </H1>");

              }

        }

    }
}

```

9.5.2 处理 HTTP 请求头及表单信息

有关 **HTTP** 请求头的和表单的介绍，可参见第 3 章。

在下面的例子 9 中，**servlet** 显示请求的 **HTTP** 头的值和表单提交的信息（可参考对比第 3 章例子 4）。

例子 8

提交信息的 JSP 页面

Example9_8.jsp:

```

<HTML>

<BODY bgcolor=cyan><FONT size=1>

<%@ page contentType="text/html;charset=GB2312" %>

<FORM action="examples/servlet/GetMessages" method=post name=form>

    <INPUT type="text" name="boy">

    <INPUT TYPE="submit" value="enter" name="submit">

</FORM>

</FONT>

</BODY>

```

</HTML>

处理 HTTP 请求头的 sevlet 源文件

GetMessages.java :

```
import java.io.*;

import java.util.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class GetMessages extends HttpServlet
{

    public void init(ServletConfig config) throws ServletException
    {super.init(config);

    }

    public void doPost(HttpServletRequest request,HttpServletResponse response)
        throws ServletException,IOException
    { // 获得一个向客户发送数据的输出流:

        PrintWriter out=response.getWriter();

        response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME
类型。

        out.println("<HTML>");

        out.println("<BODY>");

        // 客户使用的协议是:

        out.println("<BR>Protocol:");
```

```
String protocol=request.getProtocol();

out.println(protocol);

//    获取接受客户提交信息的 servlet:

out.println("<BR>Accept servlet:");

String path=request.getServletPath();

out.println(path);

//    客户提交的信息的长度:

out.println("<BR>message Length:");

int length=request.getContentLength();

out.println(length);

//    客户提交信息的方式:

out.print("<BR> Method:");

String method=request.getMethod();

out.println(method);

//    获取 HTTP 头文件中 User-Agent 的值:

out.println("<BR> User-Agent:");

String header1=request.getHeader("User-Agent");

out.println(header1);

//    获取 HTTP 头文件中 accept 的值:

out.println("<BR> accept:");

String header2=request.getHeader("accept");

out.println(header2);

//    获取 HTTP 头文件中 Host 的值:
```

```
    out.println("<BR> Host:");

    String header3=request.getHeader("Host");

    out.println(header3);

    //    获取 HTTP 头文件中 accept-encoding 的值:

    out.println("<BR> accept-encoding:");

    String header4=request.getHeader("accept-encoding");

    out.println(header4);

    //    获取客户的 IP 地址:

    out.println("<BR> client IP:");

    String IP=request.getRemoteAddr();

    out.println(IP);

    //    获取客户机的名称:

    out.println("<BR> client name:");

    String clientName=request.getRemoteHost();

    out.println(clientName);

    //    获取服务器的名称:

    out.println("<BR> server name:");

    String serverName=request.getServerName();

    out.println(serverName);

    //    获取服务器的端口号:

    out.println("<BR> ServerPort:");

    int serverPort=request.getServerPort();

    out.println(serverPort);
```



```

//    获取客户端提交的所有参数的名字:

out.println("<BR>Parameter Names");

Enumeration enum=request.getParameterNames();

    while(enum.hasMoreElements())

        {String s=(String)enum.nextElement();

            out.println(s);

        }

//    文本框 text 提交的信息:

out.println("<BR> text:");

String str=request.getParameter("boy");

    out.println(str);

out.println("</BODY>");

out.println("</HTML>");

}

public void doGet(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

{

    doPost(request,response);

}

}

```

下面的例子 9 用 servlet 实现用户注册。用户通过一个 JSP 页面提交姓名和 email 地址实现注册。当 servlet 获取这些信息后，首先检查散列表对象中是否已经存在这个名字，

该散列表存储了已经注册的用户的名字。如果目前准备注册的用户提交的名字在散列表中已经存在，就提示客户更换名字，否则将检查客户是否提供了书写正确的 email 地址，如果提供了书写正确 email 地址将允许注册（仅仅要求 email 地址中不允许出现空格）。

例子 9(效果如图 9.11 所示)

提交注册名字的 JSP 页面

Example9_9.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan><Font size=1 >

  <FORM action="examples/servlet/LoginByServlet" method=post >

    <P>   输入你的姓名:

    <INPUT type="text" name="name" value="abc">

    <BR>

    <P>   输入你的 e-mail 地址:

    <INPUT type="text" name="address" value="   ookk@sina.com">

    <P>   点击送出按钮:

    <BR>

    <INPUT TYPE="submit" value="   送出" name=submit>

  </FORM>

</FONT>

</BODY>

</HTML>
```

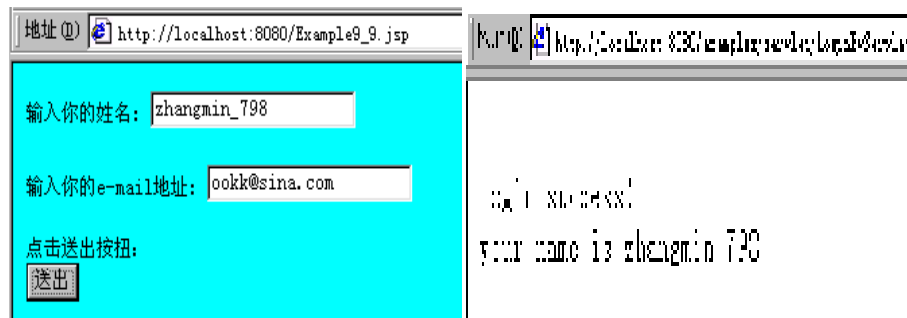


图 9.11 用 servlet 实现注册

sevlet 源文件

LoginByServlet.java:

```
import java.io.*;

import java.util.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class LoginByServlet extends HttpServlet

{ Hashtable hashtable=new Hashtable();

    public void init(ServletConfig config) throws ServletException

    {super.init(config);

    }

}
```

```
public synchronized void doPost(HttpServletRequest request,HttpServletResponse  
response)
```

```
throws ServletException,IOException
```

```
{ // 获得一个向客户发送数据的输出流:
```

```
PrintWriter out=response.getWriter();
```

```
response.setContentType("text/html;charset=GB2312");
```

```
out.println("<HTML>");
```

```
out.println("<BODY>");
```

```
// 获取用户提交的名字:
```

```
String person_name=request.getParameter("name"),
```

```
name_found=null;
```

```
if(person_name==null)
```

```
{person_name="";
```

```
}
```

```
// 在散列表查找是否已存在该名字:
```

```
name_found=(String)hashtable.get(person_name);
```

```
if(name_found==null)
```

```
{ String person_email=request.getParameter("address");
```

```
if(person_email==null)
```

```
{person_email="";
```

```
}
```

```
StringTokenizer fenxi=new StringTokenizer(person_email," @");
```

```
int n=fenxi.countTokens();
```

```

        if(n>=3)

            {out.print("<BR>"+ "there are exists illegal letters in your
email");

                }

        else

            { hashtable.put(person_name,person_name);

                out.print("<BR>"+ "login success!");

                out.print("<BR>"+ "your name is "+person_name);

            }

        }

    else

        {out.print("<BR>"+ "This name has exist ");

            }

    out.println("</BODY>");

    out.println("</HTML>");

}

public synchronized void doGet(HttpServletRequest request,HttpServletResponse
response)

    throws ServletException,IOException

{

    doPost(request,response);

}

```

}

9.5.3 设置响应的 HTTP 头

有关响应的 HTTP 头介绍，可参见第 3 章。

在下面的例子 10 中，servlet 设置响应头：Refresh 的头值是 2，那么该 servlet 在 2 秒钟后自动刷新，即 servlet 在 2 秒钟后重新调用 service 方法响应用户。

例子 10(效果如图 9.12 所示)

调用 servlet 的 JSP 页面

Example9_10:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan>

<A Href="examples/servlet/DateServlet" >查看时间</A>

</BODY>

</HTML>
```

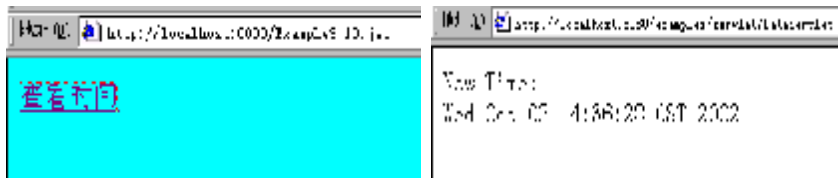


图 9.12 用 servlet 显示时间

sevlet 源文件

DateServlet.java:

```
import java.io.*;

import java.util.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class DateServlet extends HttpServlet

{ public void init(ServletConfig config) throws ServletException

    {super.init(config);

    }

    public void doPost(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

    { //    获得一个向客户发送数据的输出流:

        PrintWriter out=response.getWriter();

        response.setContentType("text/html;charset=GB2312");//    设置响应的 MIME

类型。

        out.println("<HTML>");

        out.println("<BODY>");

        response.setHeader("Refresh","2"); //    设置 Refresh 的值。

        out.println("Now Time:");

        out.println("<BR>" +new Date());

        out.println("</BODY>");

        out.println("</HTML>");
```

```

    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)

        throws ServletException, IOException

    {

        doPost(request, response);

    }

}

```

下面例子 11 实现 servlet 的重定向，客户访问 servlet: Day；如果访问的时间是在 22 点之后，就被重定向到 servlet: Night，提醒用户休息。Day 和 Night 被存放在 examples/Web-inf/classes 中。

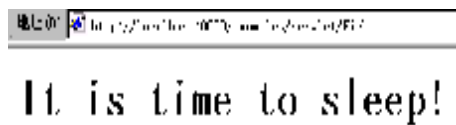


图 9.13 servlet 重定向

例子 11(效果如图 9.13 所示)

Day.java:

```

import java.io.*;

import java.util.*;

```



```

import javax.servlet.*;

import javax.servlet.http.*;

public class Day extends HttpServlet
{

    public void init(ServletConfig config) throws ServletException

        {super.init(config);

        }

    public void doPost(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

    { //    获得一个向客户发送数据的输出流:

        PrintWriter out=response.getWriter();

        response.setContentType("text/html;charset=GB2312");//    设置响应的 MIME
类型。

        out.println("<HTML>");

        out.println("<BODY>");

        Calendar calendar=Calendar.getInstance(); //    创建一个日历对象。

        calendar.setTime(new Date());//    用当前时间初始化日历时间。

        int hour=calendar.get(Calendar.HOUR_OF_DAY),

        minute=calendar.get(Calendar.MINUTE),

        second=calendar.get(Calendar.SECOND);

        if(hour>=22)

            {response.sendRedirect("Night"); //    重定向。

            }
    }

```

```

        else

            { out.print("Now time :");

                out.print(hour+":"+minute+":"+second);

            }

        out.println("</BODY>");

        out.println("</HTML>");

    }

    public void doGet(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

    {

        doPost(request,response);

    }

}

```

Night.java:

```

import java.io.*;

import java.util.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class Night extends HttpServlet

{

    public void init(ServletConfig config) throws ServletException

```

```
{super.init(config);
```

```
}
```

```
public void doPost(HttpServletRequest request,HttpServletResponse response)
```

```
throws ServletException,IOException
```

```
{ // 获得一个向客户发送数据的输出流:
```

```
PrintWriter out=response.getWriter();
```

```
response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME
```

类型。

```
out.println("<HTML>");
```

```
out.println("<BODY>");
```

```
out.println("<H1> It is time to sleep");
```

```
out.println("</BODY>");
```

```
out.println("</HTML>");
```

```
}
```

```
public void doGet(HttpServletRequest request,HttpServletResponse response)
```

```
throws ServletException,IOException
```

```
{
```

```
doPost(request,response);
```

```
}
```

```
}
```

9.6 用 servlet 读写文件

这节内容涉及到的文件操作及输入、输出流的内容可参见第 4 章。

9.6.1 读取文件的内容

在下面的例子 12 中，通过一个 JSP 页面显示给用户一些 JSP 文件的名称，该 JSP 文件存放在 Root 服务目录下。用户可以通过 Post 或 Get 方式将文件的名称提交给一个 servlet，该 servlet 存放在服务目录 examples 下的 Web-inf/classes 中。这个 servlet 将根据提交方式的不同，分别读取 JSP 文件的源代码给客户，或显示该 JSP 文件的运行效果给客户。

例子 12(效果如图 9.14、9.15、9.16 所示)

提交文件名称的 JSP 页面

read.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>
```

```
<%@ page import ="java.io.*" %>
```

```
<%! class FileJSP implements FilenameFilter
```

```
{ String str=null;
```

```
    FileJSP(String s)
```

```
    {str="."+s;
```

```
    }
```

```
    public boolean accept(File dir,String name)
```

```
    { return name.endsWith(str);
```

```
    }
```

```
}
```

```
%>
```

```
<P>下面列出了服务器上的一些 jsp 文件
```

```

<% File dir=new File("d:/Tomcat/Jakarta-tomcat-4.0/webapps/root/");

FileJSP file_jsp=new FileJSP("jsp");

String file_name[]=dir.list(file_jsp);

for(int i=0;i<5;i++)

    {out.print("<BR>" +file_name[i]);

    }

%>

```


 输入文件的 名字读取 JSP 文件的 源代码 内容:

```

<FORM action="examples/servlet/ReadFileServlet" method=post>

    <Input type="text" name="name">

    <Input type=submit value="    提交">

</FORM>

```


输入文件的 名字显示该 JSP 文件的 运行效果:

```

<FORM action="examples/servlet/ReadFileServlet" method=get>

    <Input type="text" name="name">

    <Input type=submit value="    提交">

</FORM>

```

读取文件的 servlet 源文件

ReadFileServlet:

```

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

```

```
public class ReadFileServlet extends HttpServlet
```

```
{
```

```
public void init(ServletConfig config) throws ServletException
```

```
{super.init(config);
```

```
}
```

//doPost 方法使用了回压流来读取 JSP 文件的源代码：

```
public void doPost(HttpServletRequest request,HttpServletResponse response)
```

```
throws ServletException,IOException
```

```
{ // 获取提交的文件的 名字：
```

```
String name=request.getParameter("name");
```

```
// 获得一个向客户发送数据的输出流：
```

```
PrintWriter out=response.getWriter();
```

```
response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME
```

类型。

```
out.println("<HTML>");
```

```
out.println("<BODY>");
```

```
File f=new File("d:/Tomcat/Jakarta-tomcat-4.0/webapps/root",name);
```

```
try{ FileReader in=new FileReader(f) ;
```

```
PushbackReader push=new PushbackReader(in);
```

```
int c;
```

```
char b[]=new char[1];
```

```
while ( (c=push.read(b,0,1))!=-1)// 读取 1 个字符放入字符数组 b。
```

```
{ String s=new String(b);
```

```
if(s.equals("<"))    //
```

回压的条件

```
{  push.unread('&');
```

```
    push.read(b,0,1); //push
```

读出被回压的字符字节,放入数组 b.

```
    out.print(new String(b));
```

```
    push.unread('L');
```

```
    push.read(b,0,1); //push
```

读出被回压的字符字节,放入数组 b.

```
    out.print(new String(b));
```

```
    push.unread('T');
```

```
    push.read(b,0,1); //push
```

读出被回压的字符字节,放入数组 b.

```
    out.print(new String(b));
```

```
}
```

```
else if(s.equals(">"))    //
```

回压的条件

```
{  push.unread('&');
```

```
    push.read(b,0,1); //push
```

读出被回压的字符字节,放入数组 b.

```
    out.print(new String(b));
```

```
    push.unread('G');
```

```
    push.read(b,0,1); //push
```

读出被回压的字符字节,放入数组 b.

```
    out.print(new String(b));
```

```
    push.unread('T');
```

```
    push.read(b,0,1); //push
```

读出被回压的字符字节,放入数组 b.

```
    out.print(new String(b));
```

```
}
```

```

        else if(s.equals("\n"))

            { out.print("<BR>");

              }

        else

            {out.print(new String(b));

              }

        }

    push.close();

    }

    catch(IOException e){}

    out.println("</BODY>");

    out.println("</HTML>");

    }

//doGet 方法将显示 JSP 源文件运行的效果

public void doGet(HttpServletRequest request,HttpServletResponse response)

    throws ServletException,IOException

{ String name=request.getParameter("name");

    //    获得一个向客户发送数据的输出流:

    PrintWriter out=response.getWriter();

    response.setContentType("text/html;charset=GB2312");//    设置响应的 MIME

    类型。

    out.println("<HTML>");

    out.println("<BODY>");

```



```

File f=new File("d:/Tomcat/Jakarta-tomcat-4.0/webapps/root",name);

try{ FileReader in=new FileReader(f) ;

    BufferedReader bufferin=new BufferedReader(in);

    String str=null;

    while((str=bufferin.readLine())!=null)

        {out.print("<BR>" +str);

        }

    bufferin.close();

    in.close();

}

catch(IOException e){}

out.println("</BODY>");

out.println("</HTML>");

}

}

```

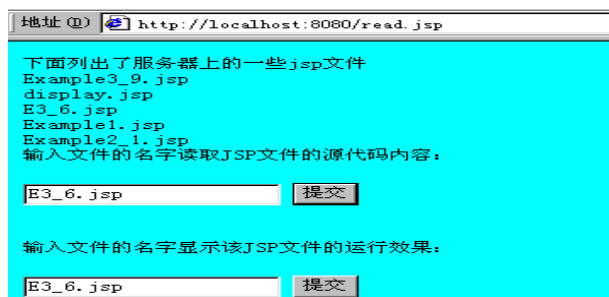


图 9.14 提交文件名字的 JSP 页面

```

<?xml?
<? page contentType="text/html; charset=GB2312" %>
<BODY>
<?>Here Are you From
<FORM action="" method=post name=form>
<INPUT type="radio" name="F" value="a" >China
<INPUT type="radio" name="R" value="b" >Russia
<INPUT type="radio" name="K" value="c" >England
<INPUT type="radio" name="F" value="d" >France
<?>What action do you prefer
<INPUT type="radio" name="P" value="a">Online

```


9.6.2 写文件

在这节，我们通过一个 **servlet** 实现小说内容的续写来说明 **servlet** 在写文件中的技巧。

在下面的例子 13 中，通过一个 **JSP** 页面显示给用户一个小说文件的已有内容，小说文件存放在服务器的 **F:/2000** 下，文件名字是 **story.txt**。**JSP** 文件存放在 **Root** 服务目录下。用户可以通过 **Post** 方式将小说的新内容提交给一个 **servlet**，该 **servlet** 存放在服务目录 **examples** 下的 **Web-inf/classes** 中。这个 **servlet** 将用户提交的内容写入小说文件的尾部。

例子 13(效果如图 9.17 所示)

提交小说内容的 JSP 页面

story.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import ="java.io.*" %>

<HTML>

<BODY>

<H4> 小说已有内容: </H4>

<Font size=1 Color=blue>

<% File f=new File("F:/2000","story.txt");

// 列出小说的内容:

try{ RandomAccessFile file=

    new RandomAccessFile(f,"r");

    String temp=null;

    while((temp=file.readUTF())!=null)
```

```

        { byte d[]=temp.getBytes("ISO-8859-1");

        temp=new String(d);

        out.print("<BR>" +temp);

    }

    file.close();

}

catch(IOException e){}

%>

```

<P> 请输入续写的新内容:

<Form action="examples/servlet/Write" method=post name=form>

<TEXTAREA name="content" ROWs="12" COLS=80 WRAP="physical">

</TEXTAREA>

<INPUT type="submit" value=" 提交内容" name="submit">

</FORM>

</BODY>

</HTML>

地址 http://localhost:8080/story.jsp

小说已有内容:

如果你的车子停错了地方, 交通警察马上会发现。
如果不给你罚单就让你把车子开走, 算你非常幸运了
我在瑞典

请输入续写的新内容:

度假时, 在我小汽车里发现这样一个条子:
“阁下: 欢迎光临敝市。此系‘禁止停车’地段。你若留意
我市路标, 在你逗留期间您会感到愉快。本通知仅提醒注意”

提交内容

地址 http://localhost:8080/examples/servlet/Write

续写文件的 servlet 源文件：

Write.java:

```
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
public class Write extends HttpServlet  
{ // 声明一个共享的文件和共享字符串：  
  
File f=null;
```

```

String use="yes" ;

public void init(ServletConfig config) throws ServletException

{super.init(config);

}

public void doPost(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

{ //    获取提交的文件内容：

    String content=request.getParameter("content");

    //    获得一个向客户发送数据的输出流：

    PrintWriter out=response.getWriter();

    response.setContentType("text/html;charset=GB2312");//    设置响应的 MIME
类型。

    out.println("<HTML>");

    out.println("<BODY>");

    f=new File("F:/2000","story.txt");

    //    把对文件的操作放入一个同步块中，并通知

    //    其它用户该文件正在被操作中：

    if(use.startsWith("yes"))

    { synchronized(f)

        { use="using";

            try{

                RandomAccessFile file=new RandomAccessFile(f,"rw");

                file.seek(file.length()); //    定位到文件的末尾。

```

```

        file.writeUTF(content);

        file.close();

        use="yes";

        out.print("<BR>"+"contents have been Write to file");

    }

    catch(IOException e){}

}

}

//    如果该小说正在被续写，就通知客户等待：

else

    {out.print("file is writing,wait please");

    }

    out.println("</BODY>");

    out.println("</HTML>");

}

public void doGet(HttpServletRequest request,HttpServletResponse response)

    throws ServletException,IOException

{

    doPost(request,response);

}

}

```

9.7 用 servlet 访问数据库

有关数据库连接的一些知识可参见第 5 章。本节通过例子说明 **servlet** 在数据库方面的

应用。我们仍然使用第 5 章的数据源 **sun**，该数据源为 **Server** 服务器上的 **pubs** 数据库，该库有一个表：**students**。

9.7.1 数据库记录查询

在下面的例子 14 中，客户通过 **condition.jsp** 页面输入查询条件，例如，查询某个姓名的成绩、查询成绩在某个分数段范围内的学生成绩等等。用户通过 **Post** 方式提交姓名给 **servlet**；分数区间通过 **Get** 方式提交给 **servlet**。该 **servlet** 根据不同的提交方式采取相应的查询方法。

例子 14(效果如图 9.18 所示)

提交查询条件的 JSP 页面

condition.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<HTML>

<BODY bgcolor=cyan>

<Font size=1>

<FORM action="examples/servlet/Inquire" Method="post">

<P> 成绩查询

<P> 输入姓名:

<Input type=text name="name">

<Input type=submit name="g" value=" 提交">

</Form>

<FORM action="examples/servlet/Inquire" Method="get" >

<P> 根据分数查询名单:<BR>
```


英语分数在

<Input type=text name="englishmin" value=1>

和

<Input type=text name="englishmax" value=100>

之间

 数学分数在

<Input type=text name="mathmin" value=1>

和

<Input type=text name="mathmax" value=100>

之间

<Input type=submit value=" 提交">

</Form>

</BODY>

</HTML>

The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/condition.jsp`. The page content is as follows:

成绩查询

输入姓名:

根据分数查询名单:

英语分数在 和 之间

数学分数在 和 之间

Below the form, there is a table with the following data:

Number	Name	Math	English	Physics
2007	zhanglin	80	90	87
2008	Liu	82	80	86
2009	Jia	80	80	88

图 9.18 使用 servlet 查询数据库

负责查询的 servlet 源文件：

Inquire.java:

```
import java.io.*;  
  
import javax.servlet.*;  
  
import javax.servlet.http.*;  
  
import java.sql.*;  
  
public class Inquire extends HttpServlet  
  
{ public void init(ServletConfig config) throws ServletException  
  
    {super.init(config);  
  
    }  
  
// 通过 Post 方法按名字查询记录：  
  
public void doPost(HttpServletRequest request,HttpServletResponse response)  
  
    throws ServletException,IOException
```

```

{ PrintWriter out=response.getWriter();

response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME 类
型。

out.println("<HTML>");
out.println("<BODY>");
// 获取提交的姓名:

String name=request.getParameter("name");

String number,xingming;

Connection con=null;

Statement sql=null;

ResultSet rs=null;

int math,english,physics;

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    }

catch(ClassNotFoundException e){}

try

{   con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

    sql=con.createStatement();

    String condition="SELECT * FROM students WHERE          姓 名 =

"+"""+name+""";

    rs=sql.executeQuery(condition);

out.print("<Table Border>");

    out.print("<TR>");

```

```

out.print("<TH width=100>"+ "Number");

out.print("<TH width=100>"+ "Name");

out.print("<TH width=50>"+ "Math");

out.print("<TH width=50>"+ "English");

out.print("<TH width=50>"+ "Phsics");

out.print("</TR>");

while(rs.next())

{ out.print("<TR>");

  number=rs.getString(1);

  out.print("<TD >"+number+"</TD>");

  xingming=rs.getString(2);

  out.print("<TD >"+xingming+"</TD>");

  math=rs.getInt("      数学成绩");

  out.print("<TD >"+math+"</TD>");

  english=rs.getInt("      英语成绩");

  out.print("<TD >"+english+"</TD>");

  physics=rs.getInt("      物理成绩");

  out.print("<TD >"+physics+"</TD>");

  out.print("</TR>") ;

}

out.print("</Table>");

con.close();

}

```

```

catch(SQLException e)

{

}

out.println("</BODY>");

out.println("</HTML>");

}

// 通过 Get 方法按成绩查询记录:

public void doGet(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

{
    PrintWriter out=response.getWriter();

    response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME 类
    型。

    out.println("<HTML>");

    out.println("<BODY>");

    // 获取提交的分数的最大值和最小值:

    String englishmax=request.getParameter("englishmax");

    String englishmin=request.getParameter("englishmin");

    String mathmax=request.getParameter("mathmax");

    String mathmin=request.getParameter("mathmin");

    String number,xingming;

    Connection con=null;

    Statement sql=null;

    ResultSet rs=null;

```

```

int math,english,physics;

try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

}

catch(ClassNotFoundException e){}

try

{   con=DriverManager.getConnection("jdbc:odbc:sun","sa","");

    sql=con.createStatement();

    String eCondition="          英语成绩 <= "+englishmax+" AND "+"    英语成绩
>= "+englishmin;

    String mCondition="          数学成绩 <= "+mathmax+" AND "+"  数学成绩 >=
"+mathmin;

    String condition="SELECT * FROM students WHERE "+mCondition+"
and "+eCondition;

    rs=sql.executeQuery(condition);

    out.print("<Table Border>");

    out.print("<TR>");

    out.print("<TH width=100>"+ "Number");

    out.print("<TH width=100>"+ "Name");

    out.print("<TH width=50>"+ "Math");

    out.print("<TH width=50>"+ "English");

    out.print("<TH width=50>"+ "Phsics");

    out.print("</TR>");

```

```

while(rs.next())

{ out.print("<TR>");

  number=rs.getString(1);

  out.print("<TD >" + number + "</TD>");

  xingming=rs.getString(2);

  out.print("<TD >" + xingming + "</TD>");

  math=rs.getInt("      数学成绩");

  out.print("<TD >" + math + "</TD>");

  english=rs.getInt("      英语成绩");

  out.print("<TD >" + english + "</TD>");

  physics=rs.getInt("      物理成绩");

  out.print("<TD >" + physics + "</TD>");

  out.print("</TR>") ;

}

out.print("</Table>");

con.close();

}

catch(SQLException e)

{

}

out.println("</BODY>");

out.println("</HTML>");

}

```

```
}
```

9.7.2 使用共享连接

数据库操作中，建立连接是耗时最大的操作之一。如果客户访问的是同一数据库，那么，为每个客户都建立一个连接是不合理的。我们已经知道，`servlet` 的成员变量是被所有用户共享的。这样，我们可以把 `Connection` 对象作为一个成员变量被所有的客户共享，也就是说第一个访问数据库的客户负责建立连接，以后所有的客户共享这个连接，每个客户都不要关闭这个共享的连接。下面的 `servlet` 使用共享连接查询数据库的所有记录。

例子 15

使用共享连接的 `servlet` 源文件

`ShareInquire.java`:

```
import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

import java.sql.*;

public class ShareInquire extends HttpServlet

{ Connection con=null; // 共享连接。

    public void init(ServletConfig config) throws ServletException

    {super.init(config);

        // 加载 JDBC-ODBC 桥接器：

        try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            }

        catch(ClassNotFoundException e){}
```



```
}
```

// 通过 Post 方法按名字查询记录:

```
public void doPost(HttpServletRequest request,HttpServletResponse response)
```

```
    throws ServletException,IOException
```

```
{ PrintWriter out=response.getWriter();
```

```
    response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME 类
```

型。

```
    out.println("<HTML>");
```

```
    out.println("<BODY>");
```

```
    Statement sql=null;
```

```
    ResultSet rs=null;
```

```
    if(con==null)
```

```
    { try
```

```
        { // 第一个用户负责建立连接 con。
```

```
            con=DriverManager.getConnection("jdbc:odbc:sun","sa","");
```

```
            sql=con.createStatement();
```

```
            String condition="SELECT * FROM students";
```

```
            rs=sql.executeQuery(condition);
```

```
            out.print("<Table Border>");
```

```
                out.print("<TR>");
```

```
                out.print("<TH width=100>"+ "Number");
```

```
                out.print("<TH width=100>"+ "Name");
```

```
                out.print("<TH width=50>"+ "Math");
```

```

        out.print("<TH width=50>"+ "English");

        out.print("<TH width=50>"+ "Phsics");

        out.print("</TR>");

while(rs.next())

    { out.print("<TR>");

        out.print("<TD >"+rs.getString(1)+"</TD>");

        out.print("<TD >"+rs.getString(2)+"</TD>");

        out.print("<TD >"+rs.getInt("        数学成绩")+ "</TD>");

        out.print("<TD >"+rs.getInt("        英语成绩")+ "</TD>");

        out.print("<TD >"+rs.getInt("        物理成绩")+ "</TD>");

        out.print("</TR>") ;

    }

    out.print("</Table>");

}

catch(SQLException e)

    {

    }

}

// 其它客户通过同步块使用这个连接：

else

{ synchronized(con)

    {try{ sql=con.createStatement();

        String condition="SELECT * FROM students";

```

```

rs=sql.executeQuery(condition);

out.print("<Table Border>");

out.print("<TR>");

out.print("<TH width=100>"+ "Number");

out.print("<TH width=100>"+ "Name");

out.print("<TH width=50>"+ "Math");

out.print("<TH width=50>"+ "English");

out.print("<TH width=50>"+ "Phsics");

out.print("</TR>");

while(rs.next())

{ out.print("<TR>");

  out.print("<TD >"+rs.getString(1)+"</TD>");

  out.print("<TD >"+rs.getString(2)+"</TD>");

  out.print("<TD >"+rs.getInt("      数学成绩")+"</TD>");

  out.print("<TD >"+rs.getInt("      英语成绩")+"</TD>");

  out.print("<TD >"+rs.getInt("      物理成绩")+"</TD>");

  out.print("</TR>") ;

}

out.print("</Table>");

}

catch(SQLException e)

{

}

```

```

    }

    }

    out.println("</BODY>");

    out.println("</HTML>");

}

public void doGet(HttpServletRequest request, HttpServletResponse response)

        throws ServletException, IOException

{
    doPost(request, response);
}

}

```

9.8 会话管理

9.8.1 获取用户的会话

我们已经知道，HTTP 协议是一种无状态协议。一个客户向服务器发出请求（request）然后服务器返回响应（response），连接就被关闭了。在服务器端不保留连接的有关信息，因此当下一次连接时，服务器已没有以前的连接信息了，无法判断这一次连接和以前的连接是否属于同一客户。因此，必须使用客户的会话，记录有关连接的信息。

一个 servlet 使用 `HttpServletRequest` 对象 `request` 调用 `getSession` 方法获取用户的会话对象：

```
HttpSession session=request.getSession(true);
```

一个用户在不同的 servlet 中获取的 session 对象是完全相同的，不同的用户的 session

对象互不相同。有关会话对象常用方法可参见第 4 章。

在下面的例子 16 中，有两个 servlet，Boy 和 Girl。客户访问 Boy 时，将一个字符串对象，存入自己的会话中，然后访问 Girl，在 Girl 中再输出自己的 session 对象中的字符串对象。

例子 16(效果如图 9.19 所示)

Boy.java:

```
import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class Boy extends HttpServlet

{ public void init(ServletConfig config) throws ServletException

    {super.init(config);

    }

    public void doPost(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

    { //    获得一个向客户发送数据的输出流:

        PrintWriter out=response.getWriter();

        response.setContentType("text/html;charset=GB2312");//    设置响应的 MIME

类型。

        out.println("<HTML>");

        out.println("<BODY>");

        HttpSession session=request.getSession(true); //    获取客户的会话对象
```

```

        session.setAttribute("name","Zhoumin");

        out.println(session.getId());           //          获取会话的 Id.

        out.println("</BODY>");

        out.println("</HTML>");

    }

    public void doGet(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

    { doPost(request,response);

    }

}

```

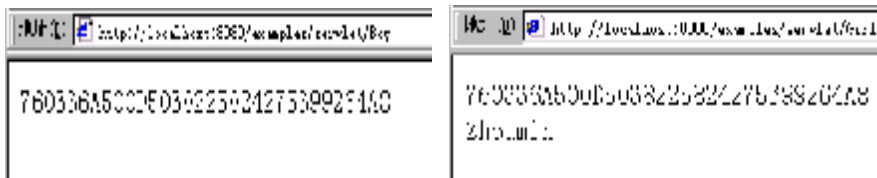


图 9.19 在 servlet 中使用会话对象

Girl.java:

```

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class Girl extends HttpServlet

{ public void init(ServletConfig config) throws ServletException

```

```
{super.init(config);
```

```
}
```

```
public void doPost(HttpServletRequest request,HttpServletResponse response)
```

```
throws ServletException,IOException
```

```
{ // 获得一个向客户发送数据的输出流:
```

```
PrintWriter out=response.getWriter();
```

```
response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME
```

类型。

```
out.println("<HTML>");
```

```
out.println("<BODY>");
```

```
HttpSession session=request.getSession(true); // 获取客户的会话对象
```

```
session.setAttribute("name","Zhoumin");
```

```
out.println(session.getId()); // 获取会话的 Id.
```

```
String s=(String)session.getAttribute("name"); // 获取会话中存储的数据。
```

```
out.print("<BR>" +s);
```

```
out.println("</BODY>");
```

```
out.println("</HTML>");
```

```
}
```

```
public void doGet(HttpServletRequest request,HttpServletResponse response)
```

```
throws ServletException,IOException
```

```
{ doPost(request,response);
```

```
}
```

```
}
```

9.8.2 购物车

用户通过一个 JSP 页面：**choice.jsp** 选择商品，提交给 **servlet: AddCar**，该 **servlet** 负责将商品添加到用户的 **session** 对象中（相当于用户的一个购物车），并将 **session** 对象中的商品显示给用户。用户可以不断地从 **choice.jsp** 页面提交商品给 **AddCar**。用户通过 **remove.jsp** 页面选择要从购物车中删除的商品提交给 **servlet: RemoveGoods**，该 **servlet** 负责从用户的购物车（用户的 **session** 对象）删除商品。

例子 17(效果如图 9.20、9.21 所示)

负责选择商品的 JSP 页面

choice.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.util.*" %>

<%@ page import="Car1" %>

<HTML>

<BODY bgcolor=cyan><Font size=1>

<P>这里是第一百货商场，选择您要购买的商品添加到购物车：

<FORM action="examples/servlet/AddCar" method=post name=form>

    <Select name="item" value="      没选择">

        <Option value="TV">      电视机

        <Option value="apple">    苹果

        <Option value="coke">     可口可乐

        <Option value="milk">     牛奶

        <Option value="tea">      茶叶
```


</Select>

<P>输入购买的数量:

<Input type=text name="mount">

<P>选择计量单位:

<INPUT type="radio" name="unit" value=" 个">个

<INPUT type="radio" name="unit" value=" 公斤">公斤

<INPUT type="radio" name="unit" value=" 台">台

<INPUT type="radio" name="unit" value=" 瓶">瓶

<Input type=submit value="提交添加">

</BODY>

</HTML>

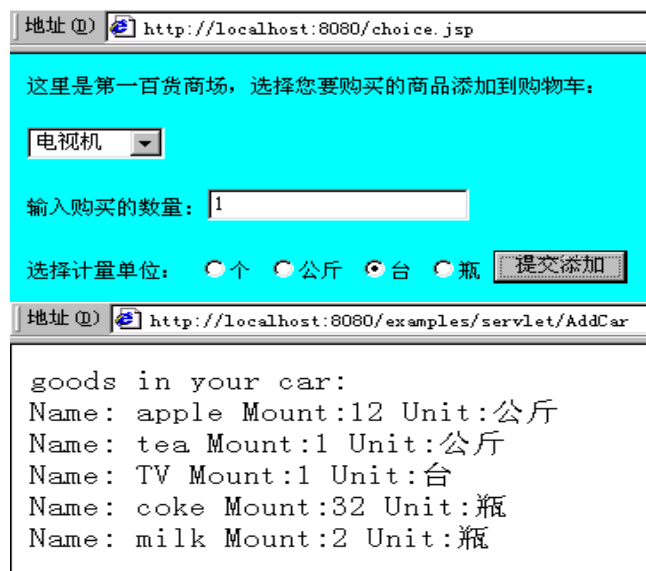


图 9.20 使用 servlet 添加商品

负责添加商品的 servlet

AddCar.java:

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class AddCar extends HttpServlet
{
    public void init(ServletConfig config) throws ServletException
    {super.init(config);
    }

    public void doPost(HttpServletRequest request,HttpServletResponse response)
        throws ServletException,IOException
    { // 获得一个向客户发送数据的输出流:

        PrintWriter out=response.getWriter();

        response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME
    }
}
```

类型。

```
out.println("<HTML>");
```

```
out.println("<BODY>");
```

```
HttpSession session=request.getSession(true); // 获取客户的会话对象
```

```
String item =request.getParameter("item"), // 获取客户选择的商品名
```

称。

```
mount=request.getParameter("mount"), // 获取客户购买的数量。
```

```
unit =request.getParameter("unit"); // 获取商品的计量单位。
```

```
// 将客户的购买信息存入客户的 session 对象中。
```

```
String str="Name: "+item+" Mount:"+mount+" Unit:"+unit;
```

```
session.setAttribute(item,str);
```

```
// 将购物车中的商品显示给客户：
```

```
out.println(" goods in your car: ");
```

```
Enumeration enum=session.getAttributeNames();
```

```
while(enum.hasMoreElements())
```

```
{ String name=(String)enum.nextElement();
```

```
out.print("<BR>" +(String)session.getAttribute(name));
```

```
}
```

```
}
```

```
public void doGet(HttpServletRequest request,HttpServletResponse response)
```

```
throws ServletException,IOException
```

```
{
```

```
doPost(request,response);
```

```
}
```

```
}
```

选择删除商品的 JSP 页面

remove.jsp:

```
<%@ page contentType="text/html;charset=GB2312" %>

<%@ page import="java.util.*" %>

<%@ page import="Car1" %>

<HTML>

<BODY bgcolor=cyan><Font size=1>

<P>选择要从购物车中删除的商品:

<FORM action="examples/servlet/RemoveGoods" method=post name=form>

    <Select name="item" value="    没选择">

        <Option value="TV">        电视机

        <Option value="apple">    苹果

        <Option value="coke">    可口可乐

        <Option value="milk">    牛奶

        <Option value="tea">    茶叶

    </Select>

<Input type=submit value="提交删除">

</FONT>

</BODY>

</HTML>
```



```
goods in your car:
Name: apple Mount:10 Unit:公斤
Name: tea Mount:1 Unit:公斤
Name: TV Mount:null Unit:null
Name: coke Mount:8 Unit:瓶
Name: milk Mount:21 Unit:瓶
```

图 9.21 使用 servlet 删除商品

负责删除商品的 servlet

RemoveGoods.java

```
import java.io.*;

import java.util.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class RemoveGoods extends HttpServlet

{public void init(ServletConfig config) throws ServletException

    {super.init(config);
```

```
}
```

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
    throws ServletException, IOException
```

```
{ // 获得一个向客户发送数据的输出流:
```

```
    PrintWriter out=response.getWriter();
```

```
    response.setContentType("text/html;charset=GB2312");// 设置响应的 MIME
```

类型。

```
    out.println("<HTML>");
```

```
    out.println("<BODY>");
```

```
    HttpSession session=request.getSession(true); // 获取客户的会话对象
```

```
    String item =request.getParameter("item"); // 获取要删除的商品名称。
```

```
        session.removeAttribute(item); // 删除商品。
```

```
    // 将购物车中的商品显示给客户:
```

```
    out.println("<H3>Now goods in your car:</H3> ");
```

```
    Enumeration enum=session.getAttributeNames();
```

```
    while(enum.hasMoreElements())
```

```
    { String name=(String)enum.nextElement();
```

```
        out.print("<BR>"+(String)session.getAttribute(name));
```

```
    }
```

```
}
```

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
    throws ServletException, IOException
```

```
{ doPost(request,response);
```

```
}  
  
}
```

9.8.3 猜数字

在第 3 章、第 6 章讲述 JSP 内置对象以及 **javabeans** 时，曾分别举过猜数字的例子。在这里，我们再使用 **servlet** 来实现猜数字这个小游戏，这样，我们就用 3 种方式实现了这个小游戏：直接由 JSP 页面来实现、通过 **javabeans** 来实现、通过 **servlet** 来实现。

当客户访问 **servlet: GetNumber** 时，随机分配给客户一个 1 到 100 之间的数，然后将这个数字存在客户的 **session** 对象中。客户在表单里输入一个数，来猜测分配给自己的那个数字。客户输入一个数字后，提交给 **servlet: Result**，该 **servlet** 负责判断这个数是否和客户的 **session** 对象中存放的那个数字相同，如果相同就连接到 **servlet: Success**；如果不相同就连接到 **servlet: Large** 或 **Small**。然后，客户在这些 **servlet** 中重新提交数字到 **Result**。

servlet 源文件

GetNumber.java:

```
import java.io.*;  
  
import java.util.*;  
  
import javax.servlet.*;  
  
import javax.servlet.http.*;  
  
public class GetNumber extends HttpServlet  
{ public void init(ServletConfig config) throws ServletException  
    {super.init(config);  
    }  
}
```

```

public void doPost(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

{ response.setContentType("text/html");

    ServletOutputStream out=response.getOutputStream();

    out.print("A number between 1 and 100 to you,guess it out please! ");

    HttpSession session=request.getSession(true);

    session.setAttribute("count",new Integer(0));

    int number=(int)(Math.random()*100)+1;    //        获取一个随机数。

    session.setAttribute("save",new Integer(number));

    out.print("<FORM action=Result method=post name=form>");

    out.print("<INPUT type=text name=boy >");

    out.print("<INPUT type=submit value=Enter>");

    out.print("</FORM>");

}

public void doGet(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

{ doPost(request,response);

}

}

```

Result.java:

```

import java.io.*;

import java.util.*;

```



```

import javax.servlet.*;

import javax.servlet.http.*;

public class Result extends HttpServlet

{ public void init(ServletConfig config) throws ServletException

    {super.init(config);

    }

    public void doPost(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

    { response.setContentType("text/html");

      ServletOutputStream out=response.getOutputStream();

      HttpSession session=request.getSession(true);

      String str=request.getParameter("boy");

      if(str==null)

          {str="0";

          }

      int guessNumber=Integer.parseInt(str);

      Integer integer=(Integer)session.getAttribute("save");

      int realnumber=integer.intValue();

      if(guessNumber==realnumber)

          { int n=((Integer)session.getAttribute("count")).intValue();

            n=n+1;

            session.setAttribute("count",new Integer(n));

            response.sendRedirect("Success");

```

```

    }

    else if(guessNumber>realnumber)

    { int n=((Integer)session.getAttribute("count")).intValue();

      n=n+1;

      session.setAttribute("count",new Integer(n));

      response.sendRedirect("Larger");

    }

    else if(guessNumber<realnumber)

    { int n=((Integer)session.getAttribute("count")).intValue();

      n=n+1;

      session.setAttribute("count",new Integer(n));

      response.sendRedirect("Smaller");

    }

  }

  public void doGet(HttpServletRequest request,HttpServletResponse response)

      throws ServletException,IOException

  { doPost(request,response);

  }

}

```

Larger.java:

```

import java.io.*;

import java.util.*;

```

```

import javax.servlet.*;

import javax.servlet.http.*;

public class Larger extends HttpServlet

{ public void init(ServletConfig config) throws ServletException

    {super.init(config);

    }

    public void doPost(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

    { response.setContentType("text/html");

      ServletOutputStream out=response.getOutputStream();

      out.print("Larger ,try again!"); //    所猜的数比实际的数大，请再猜。

      out.print("<BR><FORM action=Result  method=post name=form>");

      out.print("<INPUT type=text name=boy >");

      out.print("<INPUT type=submit value=Enter>");

      out.print("</FORM>");

    }

    public void doGet(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

    { doPost(request,response);

    }

}

```

Smaller.java

```

import java.io.*;

import java.util.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class Smaller extends HttpServlet

{

    public void init(ServletConfig config) throws ServletException

    {super.init(config);

    }

    public void doPost(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

    { response.setContentType("text/html");

        ServletOutputStream out=response.getOutputStream();

        out.print("Smaller ,try again!"); //    所猜的数比实际的数小，请再猜。

        out.print("<BR><FORM action=Result method=post name=form>");

        out.print("<INPUT type=text name=boy >");

        out.print("<INPUT type=submit value=Enter>");

        out.print("</FORM>");

    }

    public void doGet(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

    { doPost(request,response);

    }

```

```
}
```

Success.java

```
import java.io.*;

import java.util.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class Success extends HttpServlet

{

    public void init(ServletConfig config) throws ServletException

    {super.init(config);

    }

    public void doPost(HttpServletRequest request,HttpServletResponse response)

        throws ServletException,IOException

    { response.setContentType("text/html");

        ServletOutputStream out=response.getOutputStream();

        HttpSession session=request.getSession(true);

        int count=((Integer)session.getAttribute("count")).intValue();

        int num=((Integer)session.getAttribute("save")).intValue();

        long startTime=session.getCreationTime();

        long endTime=session.getLastAccessedTime();

        long spendTime=(endTime-startTime)/1000;

        out.println("Congratulatuon! You are right");
```

```
        out.println("after just"+count+"tries") ;  
        out.println("you spend"+spendTime+"Seconds");  
        out.println("That Number is"+num);  
    }  
    public void doGet(HttpServletRequest request,HttpServletResponse response)  
        throws ServletException,IOException  
    { doPost(request,response);  
    }  
}
```