第 3 章　JSP 隐含对象

在 JSP 页面中有一些不用声明就可以在 JSP 脚本中使用的对象，这就是 JSP 页面的隐含对象。隐含对象是 JSP 引擎自动创建的 Java 类实例，是 Servlet 类的对象，它们能与 Servlet 环境交互。隐含对象可以实现很多功能，如从客户端获得数据、向客户端发回数据、控制传输数据的有效域和异常处理等。

JSP 可以使用的内部对象有：

request：客户端请求，此请求会包含来自 GET/POST 请求的参数。

response：网页传回客户端的响应。

pageContext：网页的属性是在这里管理的。

session：与请求有关的会话。

application：Servlet 正在执行的内容，Servlet 上下文。

out：用来传送响应的输出流。

config：Servlet 的配置对象。

page：JSP 网页本身。

exception：针对错误网页，未捕捉的例外。

下面列举一些可以利用它们来做的事：

①不必使用表达式，你可以直接存取内部 out 对象来打印一些东西到
response: <% out.println("Hello"); %>

②不必直接传送参数，你可以借助请求对象来取得客户端输入的参数值：
<% String name=request.getParameter("name"); out.println(name); %>。

③完成页面的重定向：<% response.sendRedirect("/hello.jsp");%>

④在错误页面中显示出错信息：<% String st=exception.getMessage();%>

下面分别介绍这些隐含对象。由于篇幅限制，本章有些例子说明页面的功能、关键代码和运行效果，完整程序请参看本书第 3 章源代码，程序中的重要部分均已作出注释。

3.1　request 对象

HTTP 通信协议是在客户端与服务器之间一种传递请求与响应信息的通信协议。在 JSP 页面中，隐含对象 request 对象代表的是来自客户端的请求，那么调用该对象的方法就可以得到客户端提交的信息。例如，我们在 FORM 表单中填写的信息等。它是实现 javax.servlet.HttpServletRequest 接口类的一个实例，通过它可以查看请求参数、请求类型（GET，POST，HEAD 等）以及请求的 HTTP 头（Cookie，Referer 等）。严格说来，如果请求所用的是 HTTP 之外的其他协议，request 可以是 ServletRequest 的子类（而不是 HttpServletRequest），但在实践中几乎不会用到。request 对象的方法有很多，有些是从 javax.servlet.ServletRequest 接口中实现的，这些函数与协议类型无关，有些是 javax.servlet.HttpServletRequest 中的方法，它们只

支持 HTTP 协议。

request 对象的主要方法（协议无关的）如下：

```
public abstract String getContentType()  //返回请求数据类型
public abstract int getContentLength()  //返回请求数据长度
public abstract String getProtocol()  //返回请求协议和版本
public abstarct String getScheme()  // 返回使用的 URL 类型
public abstract String getRemoteAddr() //返回发出请求的 IP 地址
public abstract String getRemoteHost() //返回完整主机名
public abstract ServletInputStream getInputStream()  //返回
public abstract String getParameter(String name) //返回指定参数值
public abstract String[] getParameterValues(String name) //返回指定参数的一组值
public abstract Enumeration getParameterNames() //返回所有参数名
public abstract Object getAttributes(String name) //取得 request 对象中的属性，为了与
                    //同一个请求的 JSP 页面通信，还有 setAttribute()，removeAttribute()等
```

request 对象的主要方法（只支持 Http 协议的）如下：

```
public abstract String getQueryString() //
public abstract Enumeration getHeaders() //请求头信息值的枚举
public abstract String getHeader(String name)// 返回指定名字请求头信息值
public abstract Enumeration getHeaderNames()//请求头名字的枚举
public abstract String getMethod() //返回 get 或 post 等
public abstract StringBuffer getRequestUrl() //返回请求的 URL 地址
public abstract long getDateHeader(String name)// 返回某个日期型请求头的值
```

request 对象还有其他的方法，可以查阅相关手册。request 对象中比较重要的方法是：getParameter()、getParameterNames()、getParameterValues()和 getHeader()，通过调用这几个方法来获取请求对象中所包含的参数的值和请求头的信息。下面举例说明函数的用法。

3.1.1　获得客户端提交的信息

一个 html 文档 example3_1.jsp 提交一个表单给 tree.jsp 页面，tree.jsp 通过 request.getParameter()接收表单中的客户输入，包括文本条的内容和按钮的值，并将有关信息送回客户端。

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>
```

example3_1.jsp 的运行结果如图 3-1 所示。

图 3-1　example3_1.jsp 的输出结果

tree.jsp：

```
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY bgcolor=green><FONT size=1>
<P>文本框提交的信息为：
    <%String textContent=request.getParameter("boy");
    %>
<BR>
    <%=textContent%>
<P> 按钮的名字为：
    <%String buttonName=request.getParameter("submit");
    %>
<BR>
    <%=buttonName%>
</FONT>
</BODY>
</HTML>
```

下面例子中，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。要注意 JSP 页面中的 String textContent=request.getParameter("girl")语句。由于 JSP 引擎在运行这个 JSP 页面时，客户端还没有提交数据，textContent 也还没有创建，而 JSP 页面中的 number=Double.parseDouble(textContent)语句需要使用 textContent，此时 textContent 对象为 null，Java 解释器会提示出现空指针异常 NullPointerException。为了避免使用空对象，在 JSP 页面中使用了如下代码：

```
if(textContent==null)
            {textContent="";
            }
```

example3_2.jsp 关键代码如下，运行结果如图 3-2 所示。

```
<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");
        }
%>
```

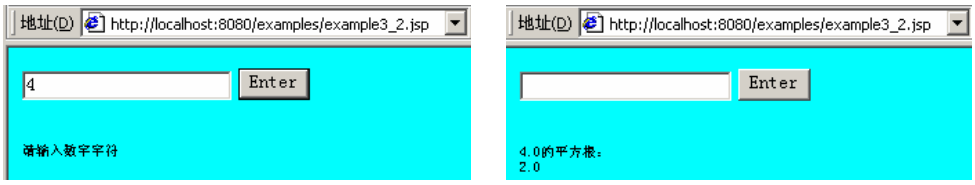


图 3-2　example3_2.jsp 的输出结果

3.1.2　处理中文信息

JSP 页面中出现汉字问题的主要原因是：客户端和服务器用 gb2312 对汉字编码，而 HTTP 传输汉字用 ISO8859_1 编码。服务器端将 JSP 转换为 Servlet 时自动将汉字转换为 ISO8859_1 编码，客户端发送汉字时也自动转换为 ISO8859_1 编码，因此服务器需要指示客户端文档的编码方式，如图 3-3 所示。

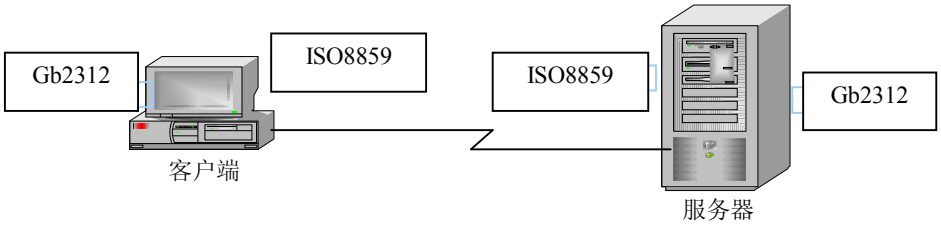
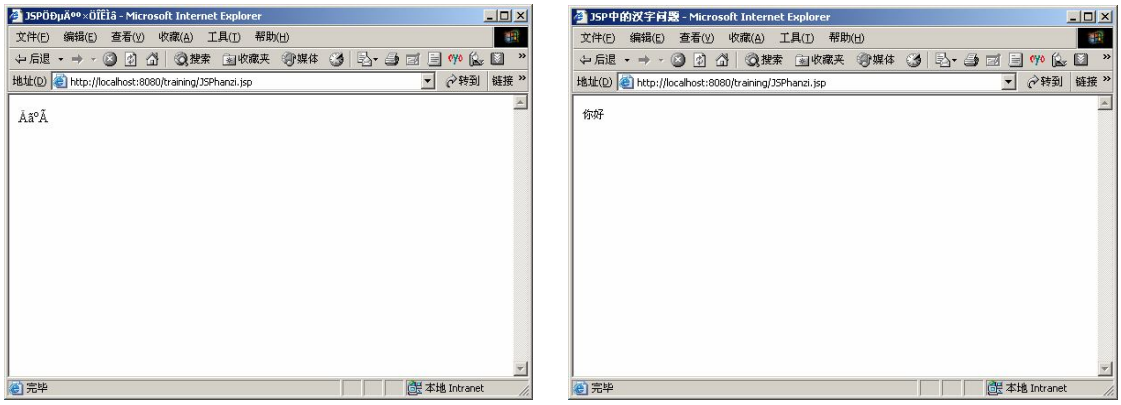


图 3-3　汉字传输和处理时的编码示意图

JSP 页面的汉字问题主要指 JSP 文件的汉字问题和表单处理的汉字问题。

（1）对于 JSP 文件的汉字问题，需要在 JSP 中使用 page 指令告诉客户端汉字的编码方式。<%@ page contentType="text/html;charset=gb2312"%>在没有特别指示下，浏览器中无法正确显示汉字。每个要显示汉字的 JSP 文档都应该包含该语句。如 hanzi.jsp 中没有上述语句，被显示为乱码。在其中加入<%@ page contentType="text/html; charset=gb2312"%>指令后“你好”被正常显示，运行结果如图 3-4（a）（b）所示。



（a） （b）

图 3-4 JSP 页面的汉字问题

hanzi.jsp：

```
<%--JSP 的汉字问题 JSPhanzi.jsp--%>
<html>
<head>
<title>JSP 中的汉字问题</title>
</head>
<body>
<%
    String HelloMsg="你好";
    out.println(HelloMsg);
%>
</body>
</html>
```

（2）表单处理的汉字问题涉及客户端和服务器的完整交互过程，在客户端和服务器端是 gb2312，传输用 ISO8859_1 编码。因此，服务器端必须指示客户端汉字的显示编码，用指令<%@ page contentType="text/html;charset=gb2312"%>，同时在服务器端接收到客户端数据时需要转换为 gb2312 后进行处理，方法是：

```
<% String name=request.getParameter("tt");
       name=new String(name.getBytes("ISO-8859-1"), "gb2312");
%>
```

或

```
<% String name=request.getParameter("tt");
       byte   b[]=name.getBytes("ISO-8859-1");
       name= new String(b);   //在 JSP 页面中默认中文编码为 GBK
%>
```

通过上述两种方法之一，提交的任何信息（无论是中文还是西文）都能正确显示。下面的例子是对 example3_1.jsp 的改进（example3_1.jsp 不变，只改动 tree.jsp 为 tree3_3.jsp），在 example3_1.jsp 中输入中文时不能正确显示。下面加入中文处理的相关语句，中文就能正确显示了，结果如图 3-5 所示。

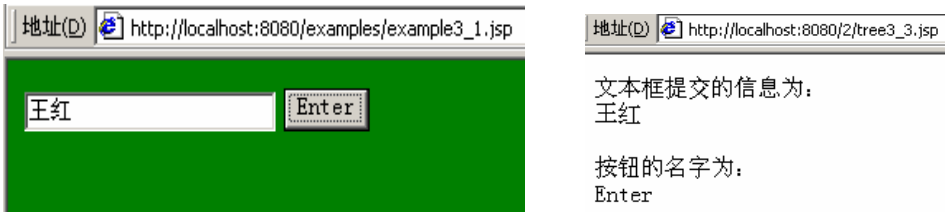


图 3-5　带中文处理的 example3_1.jsp 的输出结果

tree3_3.jsp 关键代码：

```
<%@ page contentType="text/html;charset=GB2312" %>
<P>文本框提交的信息为：
    <%String textContent=request.getParameter("boy");
      byte   b[]=textContent.getBytes("ISO-8859-1");
      textContent=new String(b);
    %>
<BR>
    <%=textContent%>
<P>  按钮的名字为：
    <%String buttonName=request.getParameter("submit");
      byte   c[]=buttonName.getBytes("ISO-8859-1");
      buttonName=new String(c);
    %>
<BR>
    <%=buttonName%>
```

3.1.3　其他常用方法

当客户端访问一个 JSP 页面时，它会提交一个 HTTP 请求给 Web 服务器的 JSP 引擎，它包含请求行、头域和可能的信息体。如：

GET/hello.htm HTTP/1.1

Host:www.sina.com.cn Http

其中首行是请求行，规定了向页面提交请求的方式，如 post、get 等，以及请求页面的文件名和使用的协议。第二行是头信息，头的信息有很多，主要包括以下几个：

user-agent：该标题最常用，返回可定制响应的客户机软件，如果客户机是 IE，则为 IFrame。

Referer：该标题返回访问服务程序文档的 URL。

Host：返回请求文档的 Internet 主机和端口，如"localhost:8080"，可用来解析含混不清的 URL 地址。

Accept：返回客户机优先接受的，以“，”隔开的 MIME 类型清单。如"text/html,text/xml,text/*"等，服务程序以此确定响应 ContentType。

Accept-Charset：客户机在响应中接受的字符集。以此检查该客户机的兼容性。

Accept-Encoding：响应可接受的内容编码，如"compress","deflate","gzip"主要用于压缩。

Accept-language：优先使用的自然语言，可用此把请求送到网页的不同语言版本。

Authorization：提供身份证名。

Connection：表示连接是否保持，通常为"keep-alive"。

Pragma：http1.1 以此实现向后兼容，常为"Cache Control:no-cache"。

一个请求还包含信息体，即 HTML 标记的组成部分，也可能是包括各种各样用于提交信息的表单等。

下面举例说明 request 对象其他方法的使用，例 example3_4..jsp 提交请求给 tree2.jsp，tree2.jsp 调用 request 对象的各种方法，运行结果如图 3-6 所示。

图 3-6　example3_4.jsp 的输出结果

example3_4.jsp 关键代码：

```
<FORM action="tree2.jsp" method=post name=form>
    <INPUT type="text" name="boy">
    <INPUT TYPE="submit" value="enter" name="submit">
</FORM>
```

tree2.jsp 关键代码：

```
<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);
%>
<BR>接受客户提交信息的长度：
<% int length=request.getContentLength();
   out.println(length);
%>
<BR>客户提交信息的方式：
<% String method=request.getMethod();
   out.println(method);
%>
<BR>获取 HTTP 头文件中 User-Agent 的值：
```

```
    <% String header1=request.getHeader("User-Agent");
        out.println(header1);
    %>
<BR>获取 HTTP 头文件中 accept 的值：
    <% String header2=request.getHeader("accept");
        out.println(header2);
    %>
<BR>获取 HTTP 头文件中 Host 的值：
    <% String header3=request.getHeader("Host");
        out.println(header3);
    %>
<BR>获取 HTTP 头文件中 accept-encoding 的值：
    <% String header4=request.getHeader("accept-encoding");
        out.println(header4);
    %>
```

3.1.4　用户注册

下面的例 login2.jsp 对 login1.jsp 中输入的用户名和 email 地址进行注册。当 login2.jsp 从 request 对象中获取客户的输入后，先在哈希表中检查是否有同名用户。若有，则告诉用户更换名字；否则检查用户的 email 地址是否正确。若不正确，则告诉用户重新输入 email 地址，否则允许注册。运行结果如图 3-7 所示。

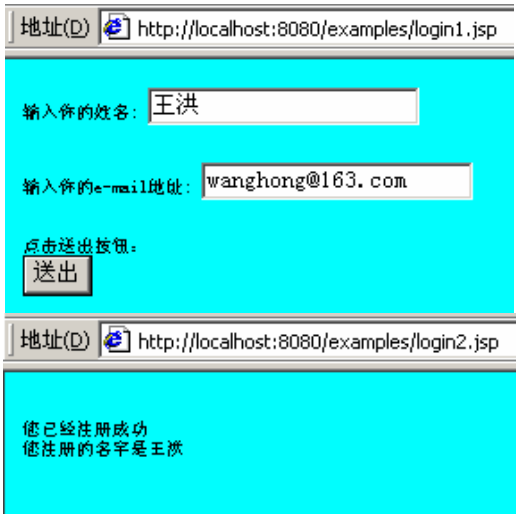


图 3-7　login1.jsp 的输出结果

哈希表是以键值对的形式存储数据的数据结构。在哈希表中键值不能重复。哈希表的构造函数是 Hashtable(int cap, float load)，cap 为初始容量，load 为负载因子，默认值是 0.75。当哈希表容量的 75%被使用时，其容量会自动增大为原来的 2 倍。

login1.jsp 关键代码：

```
<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>
```

login2.jsp 关键代码：

```
<%!Hashtable hashtable=new Hashtable();
    public synchronized void putString(String s)
    { hashtable.put(s,s);
    }
%>
<%    String person_name=request.getParameter("name"),
       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");
        }
%>
```

在上面的例子中，当服务器重新启动时就会刷新哈希表。为了保证哈希表中的数据不丢失，应该将哈希表存到文件中。客户每访问服务器时，首先从文件中读出哈希表，然后再检查信息的正确性并进行注册。对上例中 login2.jsp 的改进为 login2_2.jsp，运行结果与上例相同。login2_2.jsp 请参见源代码。

3.1.5 获取 HTML 表单中的各种数据

由于客户端经常使用表单提交数据，本节对 HTML 中的表单做一个简单介绍，并通过例子来说明表单中各种提交手段的用法。

表单是具有输入域、选择域等各种输入手段的页面，用于客户端同服务器端交互。用户在

客户端填写表单，然后“提交”，表单中的信息发送到服务器，服务器作相应的处理后返回处理情况。表单的一般格式为：

```
<form method=get|post   action="信息提交的目标页面" name="表单名"enctype="">
        表单中的标记（数据提交的手段）
</form>
```

其中，<form>是表单开始标记，</form>结束。method 取值为 GET 或 POST。GET 方法将表单信息跟在 URL 地址后传输，POST 方法将表单信息作为信息体传输。action 为表单提交的目标，通常为一个 URL。enctype 为编码方式。表单中的标记包括：<input>、<select>、<option>、<textarea>。

1．<input>的基本格式

<input>指定表单中数据输入的方式以及表单的提交键。<input>的属性主要包括：

①type：类型，指定输入方式的 GUI 对象，GUI 对象可以是 text（文本）、password（口令方式）、checkbox（多选框）、radio（单选按钮）、image（图片）、hidden（隐藏表单）、submit（提交按钮）、reset（复位按钮，将表单充置为初始状态）。

②name：名称，指定 GUI 对象的名字。

③id：标识。

④maxlength：输入最大字符数。

⑤size：输入域的宽度。

⑥value：输入域的初始值。

（1）文本框。

语法形式：<input type="text" name="me" value="hi" siae="12" align="left">

其中 align 是 text 在浏览器窗体中的对齐方式，其他同上。

下例中 text.jsp 表单的输入方式是文本框，运行结果如图 3-8 所示。

text.jsp：

```
<%@ page contentType="text/html;charset=gb2312"%>
<html>
    <head><title>表单</title><head>
<body>
<form method="POST" action="">
    <p align="center">用户登录
<p align="center">
用户名：<input type="text" name="User" size="20">    
密码：<input type="password" name="pwd" size="20"><br> <br>
</form>
</body>
</html>
```

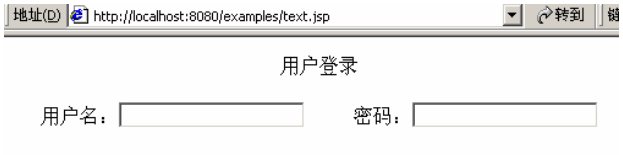


图 3-8　text.jsp 的输出结果

（2）单选按钮。

语法形式：<input type="radio" name="rad" value="red" checked="java">

其中 checked 为非空字符串，表示该单选按钮为初始被选中。注意，当一组单选按钮的 name 值相同时，这组单选按钮只能有一个被选中。服务器通过 name 指定的名字获取被选中的 radio 的 value 值。

下面的例子用单选按钮实现网上测试。客户在 radio.jsp 页面中选中几个单选按钮，将选择提交给 answer.jsp，由该页面给出成绩。运行结果如图 3-9 所示。

radio.jsp 关键代码：

```
p>诗人杜甫是中国历史上哪个朝代的人？
<form action="answer.jsp" method="post" name=form>
<input type="radio" name="girl" value="a" checked="aa">宋朝
<input type="radio" name="girl" value="b">唐朝
<input type="radio" name="girl" value="c">明朝
<input type="radio" name="girl" value="d">元朝
<br>
<p>小说《水浒》的作者是谁？
<br>
<input type="radio" name="boy" value="a">曹雪芹
<input type="radio" name="boy" value="b">罗贯中
<input type="radio" name="boy" value="c">李白
<input type="radio" name="boy" value="d">司马迁
<input type="submit" name="submit" value="提交答案">
```

answer.jsp 关键代码:

```
<%
   int num=0;
   String s1=request.getParameter("girl");
   String s2=request.getParameter("boy");
   if(s1.equals("b"))
     num++;
   if(s2.equals("b"))
     num++;
%>
<p>您的了<%=num%>分
```

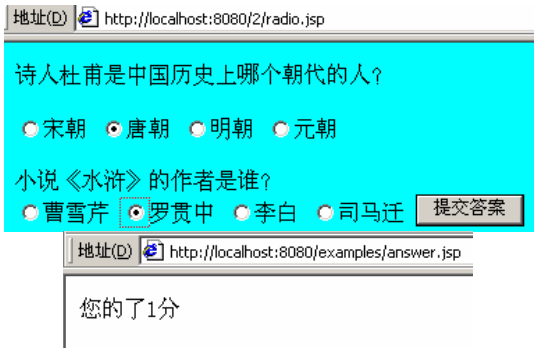


图 3-9　radio.jsp 的输出结果

（3）复选框。

语法形式：<input type="checkbox" name="ch" value="pin" checked="java">

复选框的 name 的值应该不同，JSP 页面可通过 getParameterValues()方法得到所有选中的复选框的 value 值。

（4）口令框。

语法形式：<input type="password" name="me" size=12>

当客户在口令框中输入口令时，口令框中回显的不是口令字符而是其他字符，如“*”。口令框只起到输入时不让人看见的作用，传输时不提供保密措施。

（5）提交按钮。

语法形式：<input type="submit" name="me" value="yes" size="12">

当按下提交按钮时，服务器可获得客户提交的任何数据。

（6）复置键。

语法形式：<input type="reset">

复置键将表单中的数据清空，以便重新输入。

2. <select>、<option>格式

下拉列表和滚动列表通过<select>、<option>标记来定义。其基本格式为：

```
<select>
      <option>
      <option>
            …
</select>
```

如下拉列表：

```
<select name="xiala">
      <option value="cat">小猫
      <option    value="dog">小狗
</select>
```

再如滚动列表：

```
<select name="gundong"    size=2>
      <option value="1">计算 1 到 n 的和
      <option value="2">计算 1 到 n 的平方和
<select>
```

当 select 标记中增加 size 属性时，就成为滚动列表。

下面的例子，客户在 select.jsp 中通过滚动列表选择计算的方式，通过下拉列表选择计算的项数，并将结果提交给 sum.jsp。sum.jsp 将运算结果输出。运行结果如图 3-10 所示。

select.jsp 关键代码：

```
<p>选择计算和的方式
<form action="sum.jsp" method=post name=form>
<select name="girl" size=2 >
  <option value="1">计算 1 到 n 的和
  <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>请选择 n 的值
<select name="boy">
   <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>
<input type="submit" name="submit" value="提交您的选择">
```

sum.jsp 关键代码：

```
<% int sum=0;
   String s1=request.getParameter("girl");
   String s2=request.getParameter("boy");
   int n=Integer.parseInt(s2);
   if(s1.equals("1"))
   for(int i=1;i<=n;i++)
     sum+=i;
   else
   {
      if(s1.equals("2"))
        for(int i=1;i<=n;i++)
           sum+=i*i;
      else
        for(int i=1;i<=n;i++)
           sum+=i*i*i;
    }
%>
<p>您的求和结果是<%=sum%>
```

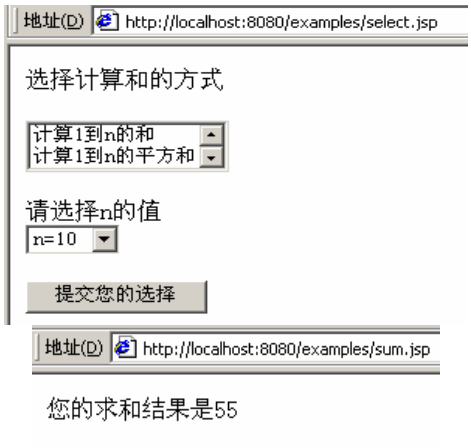


图 3-10　select.jsp 的输出结果

3．<textarea>格式

指定在表单中输入多行文本。语法形式为：

<textarea NAME="Computer" ROWS=6　COLS=64>
</textarea>

下面举一个综合例子。在例中，客户在 zonghe.jsp 中输入或选择个人相关信息，运行结果如图 3-11 所示。zonghe.jsp 请参见源代码。

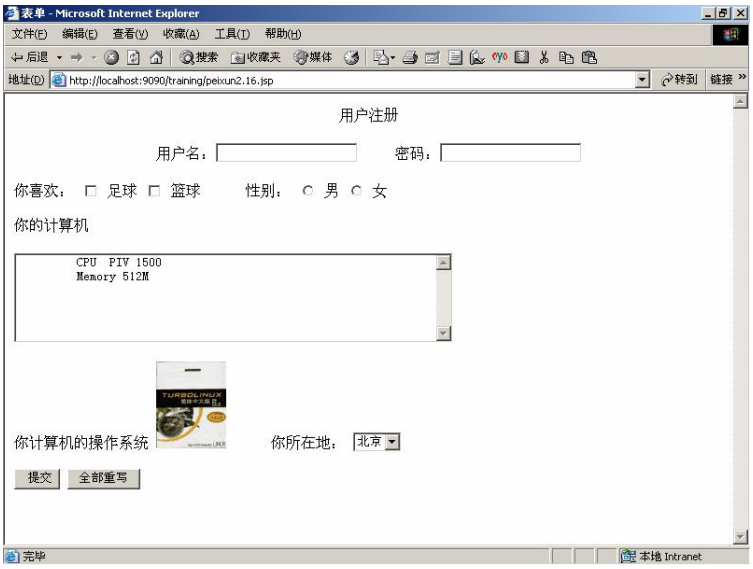


图 3-11　表单综合例运行结果

3.1.6　表格

表格由<table>和</table>标记定义。一般格式为：

<table>
<TR width="该行的宽度"border>
　　<TH　width="单元格宽度" Align="对其方式">单元格中的数据</TH>
　　…
　　<TD width="单元格宽度" Align="对其方式">单元格中的数据</TD>
</TR>
<table>

其中<tr>表示表格的一行，<th>和<td>表示行中的一个单元格。<th>和<td>的主要区别是<th>定义的单元格着重显示，而<td>定义的单元格为普通单元格，不着重显示。一行的单元格可以是着重单元格，也可以是普通单元格，还可以交替出现。Border 属性表示表格是否有边框。

下例 table.jsp 演示表格的基本用法，运行结果如图 3-12 所示。

图 3-12　table.jsp 的运行结果

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">上沿</TH>
        <TD   Valign="Bottom">下沿</TD>
        <TD   Valign="Bottom" Align="Center" >中下沿</TD>
    </TR>
    <TR >
        <TD   Valign="Top">  你好</TD>
        <TD   Valign="Bottom">hello</TD>
        <TD   Valign="Bottom" Aligin="Center" >how are you</TD>
    </TR>
  </Table>
</BODY>
</HTML>
```

下例 tableform.jsp 是表单和表格的综合使用，运行结果如图 3-13 所示。

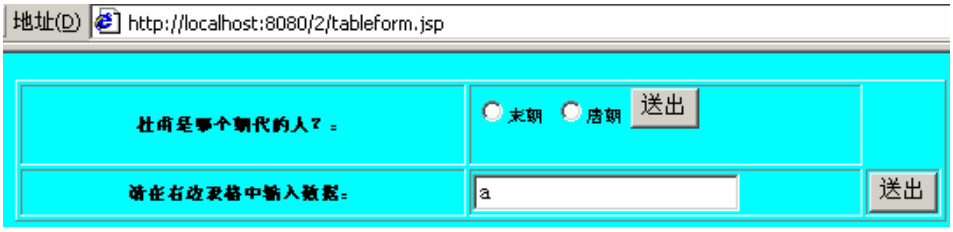


图 3-13　tableform.jsp 运行结果

tableform.jsp 关键代码：

```
  <Table align="left" Border>
    <TR>
          <TH   width=250><FONT size=1>杜甫是哪个朝代的人？：</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>请在右边表格中输入数据：</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>
```

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，服务器接到请求时会返回 HTTP 响应。响应也有结构，由状态行开始，包含一系列头和信息体。响应是实现 javax.servlet.HttpServletResponse 接口类的一个实例。“response”对象代表的是对客户端的响应，也就是说可以通过“response”对象来组织发送到客户端的数据。但是由于组织方式比较底层，所以不建议普通读者使用，需要向客户端发送文字时直接使用“out”对象即可。和 request 对象一样，response 对象的方法也有很多，有些是从 javax.servlet.ServletResponse 接口中实现的，这些函数与协议类型无关，有些是 javax.servlet.HttpServletResponse 中的方法，它们只支持 HTTP 协议。

response 对象的方法（与协议无关的）：

public void setContentType(String type)　//设定对客户端响应的 MIME 类型

public ServletOutputStream getOutputStream() //得到返回客户端的输出流

Public String getCharacterEncoding()　//得到字符编码

public java.io.PrintWriter getWriter() //得到返回客户端的打印流

response 对象的方法（只支持 HTTP 协议）：

public void sendError(int status) //转到错误码对应的页面

public void sendRedirect(String location) //页面重定向

Public void setStatus(int code) //设置状态码

void addHeader(name,value) //向 response 对象中添加头

void addCookie(cookie)　//向 response 对象中添加 Cookie

boolean containsHeader(name)　//检查 response 对象中是否含有 name 指定的头

void setHeader(name,value)　//设定头中 name 的值

response 对象中使用较多的方法是 setContentType()、sendRedirect()和 getWriter()。下面举例说明 response 对象的主要用法。

3.2.1　动态设置 contentType 属性

在 page 指令中的 contentType 属性用来指定 response 对象响应客户端请求的 MIME

（Multipurpose Internet Mail Extension protocol，多用途网际邮件扩充协议）类型。MIME 类型有很多，contentType 属性不同的值对应不同的 MIME 类型。如，当客户访问一个 JSP 页面时，如果页面的 page 指令设定页面的 contentType 属性值为 text/html，那么 JSP 引擎将按照这种属性值，以 HTML 文档的形式做出响应。ContentType 属性值和 MIME 类型的主要对应关系如表 3-1 所示。

表 3-1　contentType 属性值和 MIME 类型的主要对应关系

contentType 属性值	MIME 类型	说明
text/html	html	html 文档
text/plain	txt	文本文件
application/java	class	java 类文件
application/zip	zip	压缩文件
application/pdf	pdf	pdf 文件
image/gif	gif	图片类型
audio/basic	au	音频类型
Application/x-msexcel	Excel	电子表格
Application/msword	Word	Word 文档

但是，page 指令只能为 contentType 指定一个值，如果想动态地改变这个属性的值来以各种不同的方式做出相应，就要使用 response 对象的 setContentType()方法改变 contentType 的值。一旦这个属性值被改变了，服务器就将按照新的 MIME 类型做出响应，一直到再次改变这个属性值。需要注意，客户端一定要保证支持新的 MIME 类型。客户端要想知道自己的浏览器支持哪些 MIME 类型，可以单击资源管理器→工具→文件夹选项→文件类型，如图 3-14 所示。

图 3-14　查询 MIME 类型界面

在下例中，页面动态地将 contentType 属性值置为 application/msword。当客户单击 example3_10.jsp 页面中的按钮时，页面将保存为 Word 文档。运行结果如图 3-15 所示。

example3_10.jsp 关键代码：

```
<P>动态设置 mime 类型的例子
 <BR>setContentType 方法
 <P>当你点击按钮时当前页面就会显示在 word 文档下
  <FORM action="" method="get" name=form>
      <INPUT TYPE="submit" value="yes" name="submit">
    </FORM>
 <% String str=request.getParameter("submit");
       if(str.equals("yes"))
           {response.setContentType("application/msword;charset=GB2312");
           }
%>
```

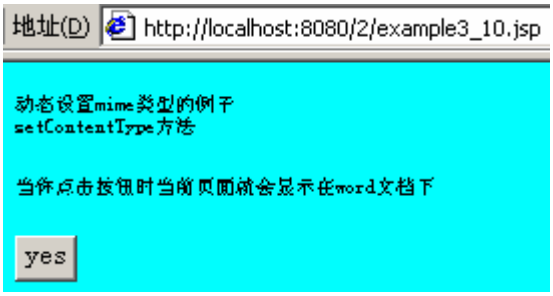


图 3-15　example3_10.jsp 的运行结果

在下例中，example3_11.jsp 将请求提交给 tree3.jsp，tree3.jsp 动态设定 contentType 属性值为 application/x-msexcel。用来把动态包含的 A.txt 文档显示为 Word 形式或 Excel 电子表格形式。需要注意，编辑 A.txt 时，回车要用
表示，输入空格时要将输入法切换到全角方式，因为在半角方式输入的多个空格符会被浏览器理解为一个。为了能用 Excel 表示文件，数据列之间要有 4 个空格。运行结果如图 3-16 所示。

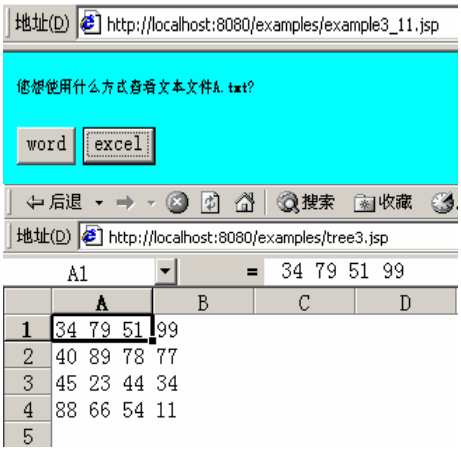


图 3-16　example3_11.jsp 的运行结果

A.txt

```
34    79    51    99<BR>
40    89    78    77<BR>
```

```
45      23      44      34<BR>
88      66      54      11<BR>
```

example3_11.jsp 关键代码：

```
  <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>
```

tree3.jsp 关键代码：

```
<%   String str1=request.getParameter("submit1");
     String str2=request.getParameter("submit2");
        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>
```

3.2.2 response 对象的 http 头

和 request 对象一样，response 对象也是有结构的，由状态行开始，包含一系列头和信息体。response 对象可以使用 addHeader()和 setHeader()方法动态添加和修改相应头的值，并将这些头发送给客户端。下面的例 example3_12.jsp 中，response 对象添加一个头 refresh,使客户端在显示该页面时，每 5 秒钟刷新一次。运行结果如图 3-17 所示。

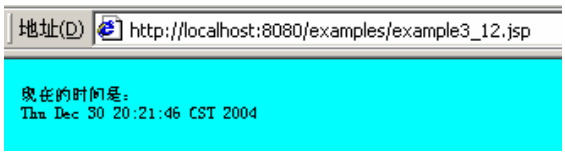


图 3-17　example3_12.jsp 的运行结果

example3_12.jsp 关键代码：

```
<%@ 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");
 %>
</FONT>
</BODY>
```

3.2.3　response 重定向

很多情况下，响应会根据客户的不同输入转向另外的 JSP 页面。如当客户输入口令不正确时，就会被引导到错误提示页面或转回初始页面让客户重新输入口令。可以使用 response 对象的 sendRedirect()方法实现页面重定向。

下面例中，客户在 example3_13.jsp 中填写表单，并提交给 tree4.jsp 页面。如果表单填写不完整，tree4.jsp 会重新定向到 example3_13.jsp，否则显示欢迎字样，运行结果如图 3-18 所示。

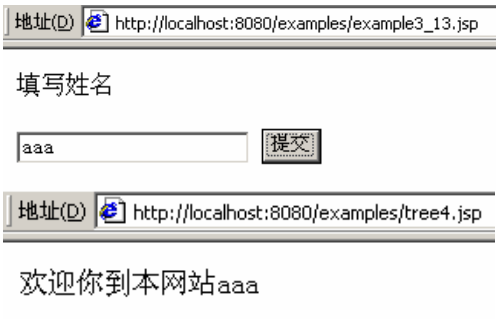


图 3-18　example3_13.jsp 的运行结果

example3_13.jsp 关键代码：

```
<p>填写姓名<br>
<form action="tree4.jsp" method=post name=form>
<input type="text" name="aa">
<input type="submit" name="submit" value="提交">
</form>
```

tree4.jsp 关键代码：

```
<% String s=request.getParameter("aa");
    if(s.equals(""))
    response.sendRedirect("example3_13.jsp");
    else
    {out.print("欢迎你到本网站");
     out.print(s);
    }
%>
```

3.2.4　response 的状态行

当服务器响应客户端请求时，它发送的首行被称为状态行。状态行包括 3 位数字的状态码和对状态码的简单描述。对 5 类状态码的大致描述如表 3-2 所示。其中，1xx 主要是试验性质的；2xx 表示请求成功；3xx 表示在请求满足之前应进一步采取的行动；4xx 表示服务器无法满足要求；5xx 服务器出现错误。

表 3-2 状态码表

状态码	描述
101	服务器正在升级协议
100	客户可以继续
201	请求成功且在服务器上创建了新资源
202	请求已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本服务器的
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	应答可以在另一个 url 中找到
304	get 方式不可用
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	得到请求但拒绝服务
404	请求的资源不可用
405	请求所用的方法不允许
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
410	请求的资源已经不可用
411	请求需要一个内容才能处理
413	请求太大，被拒绝
414	请求的 url 太大
415	请求格式被拒绝
500	服务器内部错误
501	不支持请求的部分功能
502	从代理或网关接受了不合法字符
503	http 服务暂时不可用
504	服务器在等待代理应答时发生超时
505	不支持请求的 http 版本

一般不需要修改状态行，服务器会根据执行的情况自动置状态码并发送给客户。不过可以使用 response 对象的 setStatus()方法设置响应的状态码。在下例中，example3_14.jsp 可以超链接到不同的页面，这些页面分别置不同的状态码。运行结果如图 3-19 所示。

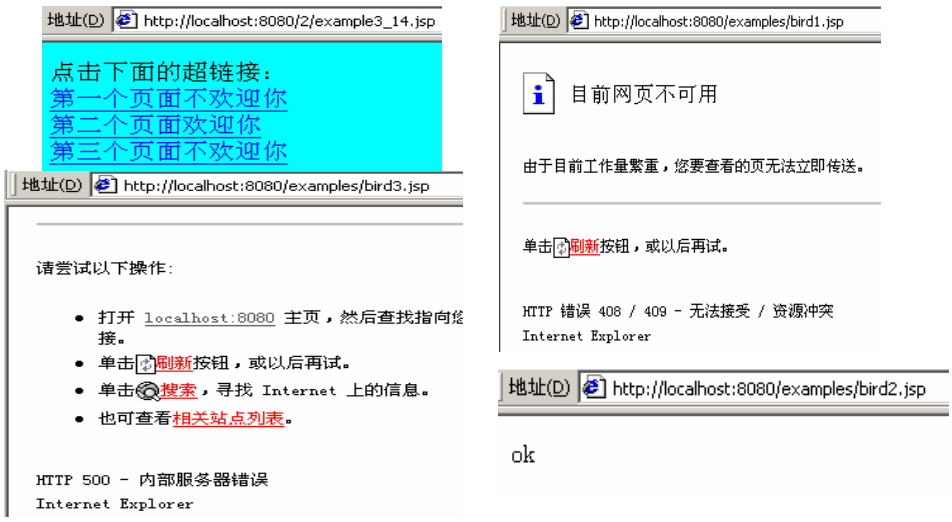


图 3-19　example3_14.jsp 的运行结果

example3_14.jsp 关键代码:

```
<P>点击下面的超链接：<BR>
    <A HREF="bird1.jsp"> 第一个页面不欢迎你
<BR>
  <A HREF="bird2.jsp"> 第二个页面欢迎你
<BR>
  <A HREF="bird3.jsp"> 第三个页面不欢迎你
```

bird1.jsp 关键代码：

```
      <% response.setStatus(408);
     out.print("不显示了");
 %>
```

bird2.jsp 关键代码：

```
 <% response.setStatus(200);
     out.println("ok");
  %>
```

bird3.jsp 关键代码：

```
<%
     response.setStatus(500);
  %>
```

3.3　out 对象

out 对象代表了向客户端发送数据的对象，与 response 对象不同，通过 out 对象发送的内容将是浏览器需要显示的内容，是文本一级的。可以通过 out 对象直接向客户端写一个由程序动态生成的 HTML 文件。它是 javax.servlet.jsp.JspWriter 类的一个实例。Out 对象是带缓存功能的 PrintWriter。JSP 允许通过 page 指令的 buffer 属性调整缓存的大小，甚至允许关闭缓存。它的重要方法是：

```
print()   //在客户端输出各种数据类型数据
println()   //在客户端换行输出各种数据类型数据
newLine()   //输出一个换行符
flush()   //清空输出缓冲区
close()   //关闭流
boolean isAutoFlush()   //检查流是否自动清缓冲
int getBufferSize()   //得到缓冲区的大小
int getRemaining()   //得到缓冲区中未用字节数
void clear()   //清缓冲区
```

因为在每个页面中都用到 out 对象，因此在这儿仅举一个例子再展示一下 out 对象的功能。例 time.jsp 用 out 对象向客户端输出信息，其中用到 Calendar 类，运行结果如图 3-20 所示。

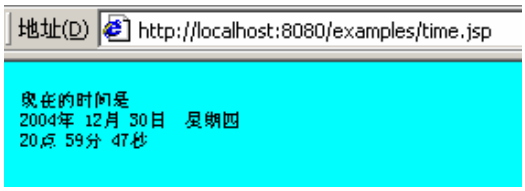


图 3-20　time.jsp 的运行结果

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><Font size=1>
<% 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<BR>
<%=年%>年
<%=月%>月
<%=日%>日
<%=星期%>
<BR>
<%=hour%>点
```

```
<%=minute%>分
<%=second%>秒
<%   if(日.equals("26"))
       {out.print("<BR><H2>今天是病毒容易发作的日子！</H2>");
       }
     if(hour>=22)
       { out.print("<BR><H2>现在时间很晚了注意休息</H2>");
       }
%>
</FONT>
</BODY>
</HTML>
```

3.4　session 对象

HTTP 是无状态（stateless）协议。客户端向服务器发送一次请求（request），服务器给与响应（response）后连接就关闭了。因此当下一次连接时，服务器已经没有以前的连接信息了。同时，服务器无法判断这一连接和以前的连接是否属于同一个客户。服务器对每一个客户端请求都没有历史记忆的特点带来了一系列问题。特别是通过在线商店购物时，服务器不能顺利地记住以前的事务就成了严重的问题。它使得“购物车”之类的应用很难实现：当我们把商品加入购物车时，服务器如何才能知道车里原先有些什么？即使服务器保存了上下文信息，我们仍旧会在电子商务应用中遇到问题。例如，当用户从选择商品的页面（由普通服务器提供）转到输入信用卡号和送达地址的页面（由支持 SSL 的安全服务器提供），服务器如何才能记住用户买了些什么？这个问题一般有四种解决方法。

第一种：Cookie。利用 javax.servlet.http.HTTPCookie 对象来存储有关购物会话的信息，后继的各个连接可以查看当前会话，然后从服务器的某些地方提取有关该会话的完整信息。这是一种优秀的，也是应用最广泛的方法。然而，即使 Servlet 提供了一个高级的、使用方便的 Cookie 接口，使用 cookie 时仍旧要做一些烦琐的细节处理，如：从其他 Cookie 中分辨出保存会话标识的 Cookie；为 Cookie 设置合适的作废时间（例如，中断时间超过 24 小时的会话一般应重置）；把会话标识和服务器端相应的信息关联起来（实际保存的信息可能要远远超过保存到 Cookie 的信息，而且像信用卡号等敏感信息永远不应该用 Cookie 来保存）。

第二种：重写 URL。你可以把一些标识会话的数据附加到每个 URL 的后面，服务器能够把该会话标识和它所保存的会话数据关联起来。这也是一个很好的方法，而且还有在浏览器不支持 Cookie 或用户已经禁用 Cookie 的情况下也有效这一优点。然而，大部分使用 Cookie 时所面临的问题同样存在，即服务器端的程序要进行许多简单但单调冗长的处理。另外，还必须十分小心地保证每个 URL 后面都附加了必要的信息（包括非直接的，如通过 Location 给出的重定向 URL）。如果用户结束会话之后又通过书签返回，则会话信息会丢失。

第三种：隐藏表单域。HTML 表单中可以包含下面这样的输入域：＜INPUT TYPE="HIDDEN" NAME="session" VALUE="..."＞。这意味着，当表单被提交时，隐藏域的名字和数据也被包含到 GET 或 POST 数据里，我们可以利用这一机制来维持会话信息。然而，这种方法有一个很大的缺点，它要求所有页面都是动态生成的，因为整个问题的核心就是每个

会话都要有一个惟一标识符。

第四种：Servlet 为我们提供的一种与众不同的方案——HttpSession API。HttpSession API 是一个基于 Cookie 或者 URL 改写机制的高级会话状态跟踪接口。如果浏览器支持 Cookie，则使用 Cookie；如果浏览器不支持 Cookie 或者 Cookie 功能被关闭，则自动使用 URL 改写方法。Servlet 开发者无需关心细节问题，也无需直接处理 Cookie 或附加到 URL 后面的信息，API 自动为 Servlet 开发者提供一个可以方便地存储会话信息的地方。

由于 JSP 页面在运行时首先被转化为 Servlet，因此在 JSP 页面中使用的会话就是 Servlet 中提供的会话机制。从一个客户打开浏览器连接到服务器开始，到客户关闭浏览器离开这个服务器结束，被称为一个会话。一个客户访问服务器时，可能会在这个服务器的几个页面之间反复连接，反复刷新一个页面或不断向一个页面提交信息等。服务器完全可以通过 session 对象知道这是同一个客户。session 对象使同一用户在访问 Web 站点时多个页面间共享信息。Session 对象是 javax.servlet.http.HttpSession 类的一个实例，它可以存储有关会话的属性。

在 JSP 页面中使用会话信息是相当简单的，主要的操作包括：查看和当前请求关联的会话对象、查看与某个会话相关的信息、在会话对象中保存信息以及会话完成或中止时释放会话对象。Session 对象的主要方法有：

String getId () //该方法返回会话的惟一标识。有时该标识被作为键-值对中的键使用
//比如会话中只保存一个值时，或保存上一次会话信息时

·void invalidate () //使 session 对象失效

·boolean isNew() //检测当前的 session 对象是否为新的。如果客户（浏览器）还没有绑定到会话则返回
//true，通常意味着该会话刚刚创建，而不是引用自客户端的请求。对于早就存在的会
//话，返回值为 false

·long getCreationTime() //该方法返回建立会话的以毫秒计的时间

·long getLastAccessedTime() //该方法返回客户最后一次发送请求的以毫秒计的时间

·long getMaxInactiveInterval() //返回以秒计的最大时间间隔，如果客户请求之间的间隔不超过该值
//JSP 引擎将保持会话有效。负数表示会话永远不会超时

·long setMaxInactiveInterval(long millis) //设定以秒计的 session 不活动的最大时间间隔

·Object getValue(Object key) //查看 session 对象中键 key 所保存数据值

·void putValue(Object key, Object value) // 在 session 对象中保存数据，键为 key，所对应的值为 value.

·Object getAttribute(Object key) // 功能同 getValue()方法

·void setAttribute(Object key, Object value) //功能同 putValue()方法

·String[] getValueNames() //得到 session 对象中所有属性的名字

·Enumeration getAttributeNames() //得到 session 对象中所有属性的名字的枚举

关于 session 对象的方法有几点说明：

（1）在有关 session 的生命周期函数中，如：getCreationTime()、setMaxInactiveInterval()、getLastAccessedTime()等，时间的基准点都是从格林威治（GMT）时间 1970 年 1 月 1 日午夜 0 时 0 分 0 秒算起。要得到时间值，可以把该值传递给 Date 构造函数，或者 GregorianCalendar 的 setTimeInMillis 方法。

（2）会话对象提供一个内建的数据结构，在这个结构中可以保存任意数量的键—值对。在 2.1 或者更早版本的 Servlet API 中，使用 getValue()方法和 putValue()方法，而在 Servlet API 2.2 版推荐用 getAttribute()来代替 getValue()，用 setAttribute()方法代替 putValue()。这不仅是因为 getAttribute()和 setAttribute()的名字更加匹配，同时也因为允许使用一个附属的 HttpSessionBindingListener 来监视属性值的变化，而 getValue()和 putValue()则不能。

（3）getValue()和 getAttribute()的返回结果类型为 Object，因此你必须把它转换成更加具体的数据类型。如果参数中指定的键不存在，则返回 null。

（4）putValue()和 setAttribute()中的参数，即键和值，都是 Object 类型。如果你要使用的键和值不是对象类型，则必须包装成对象类型。如，要向 session 对象中存放属性，键为 1，值为 2，则必须写成 setAttribute(new Integer(1),new Integer(2))或 setAttribute(“1”,new Integer(2))，而写成 setAttribute(1,2)是错误的。

（5）大多数时候我们都是根据特定的名字寻找与它关联的值，但也可以调用 getValueNames()返回 session 对象中所有属性的名字的数组。GetValuesNames()返回的是一个 String 数组。Servelt API 2.2 版推荐使用 getAttributeNames()方法，这不仅是因为其名字更好，而且因为它返回的是一个 Enumeration，和其他方法（例如 HttpServletRequest 的 getHeaders 和 getParameterNa mes）更加一致。

3.4.1　session 对象的 ID 标识

客户首次访问服务器上的 jsp 页面时，jsp 引擎产生 session 对象，并分配给它一个 String 类型的 ID，并将其发送到客户端存入 Cookie 中，这样 session 对象和客户端之间建立了一一对应关系，即每个客户端对应一个 session。session 对象不同，它对应的 ID 也不同。客户在该服务器不同网页间转换或从其他服务器回到该服务器时，不分配给客户新的 session,直到客户关闭浏览器，分配给客户的 session 对象被取消。当客户再打开浏览器再次连接到该服务器时，服务器为该客户创建一个新的 session 对象，并分配 ID。

下例中，客户在 3 个页面 session.jsp、tom0.jsp 和 jerry.jsp 之间进行连接，只要不关闭浏览器，三个页面的 session 对象完全相同，这可以从 3 个页面中得到的 session ID 看到。运行结果如图 3-21 所示。

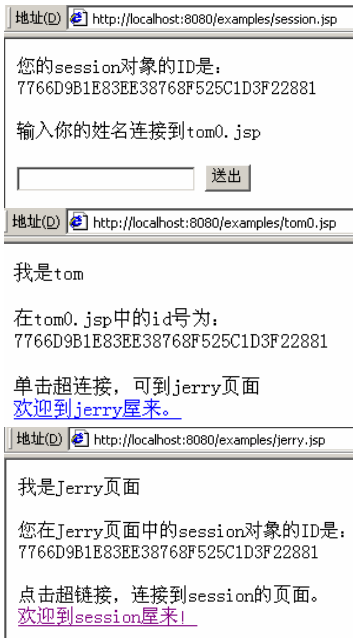


图 3-21　session.jsp 的运行结果

session.jsp 关键代码：

```
    <% 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 tom0.jsp
    <FORM action="tom0.jsp" method=post name=form>
        <INPUT type="text" name="boy">
        <INPUT TYPE="submit" value="送出" name=submit>
  </FORM>
```

tom0.jsp 关键代码：

```
<p>我是 tom
<% String sid=session.getId();
%>
<p>在 tom0.jsp 中的 id 号为：<%=sid%>
<br>
<p>单击超连接，可到 jerry 页面
<a href="jerry.jsp">
<br>欢迎到 jerry 屋来。
</a>
```

jerry.jsp 关键代码：

```
<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>
```

3.4.2 session 对象和 URL 重写

session 对象能和客户建立一一对应关系的基础是客户端支持 Cookie。如果客户端不支持 Cookie，也就不能建立 session 对象和客户端的一一对应关系。因为服务器无法将 session ID 写到客户端，当客户访问服务器的不同页面时，服务器得不到它曾经分配给该客户的 session ID，于是服务器认为这是一个新客户，并分配一个新的 session 对象。这样，客户端在不同网页中的 session 对象的 ID 就不同。可以将浏览器的 Cookie 设置为禁止状态（选择浏览器菜单→工具→Internet 选项→安全→Internet 和本地 Internet→自定义级别→Cookie,将全部选项置为禁止），再运行例 session.jsp，会得到不同的结果，如图 3-22 所示。可见，服务器已经无法识别同一个客户端。

解决该问题的有效方法是 URL 重写技术。采用 URL 重写后，当客户在不同页面之间连接时，把 session ID 添加在新页面的 url 地址后面一起传过去，这样保障客户在不同的网页之间共享同一个 session 对象。可以使用 response 对象的 encodeURL()或 encodeRedirectURL()方法

实现 URL 重写。比如，从 tom.jsp 连接到 jerry.jsp 时，首先实现 URL 重写，将 session ID 附加在 URL 地址后面形成新字符串，使用：

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

然后将连接目标写成<%=str%>。

下例 session1.jsp 是对例 session.jsp 的改进，保证无论客户端是否支持 Cookie，三个页面之间的 session ID 都是相同的，运行结果如图 3-22 所示，与图 3-21 相同。

图 3-22　session.jsp 再次运行结果

session1.jsp 关键代码：

```
<% String sid=session.getId();
    String str=response.encodeURL("tom1.jsp");
%>
<p>在 session.jsp 中的 id 号为：<%=sid%>
<br>
<p>您向 http://localhost:8080/tom1.jsp 发送的消息为<%=str%>
<br>
<form action="<%=str%>" method=post name=form>
<input type="text" name="boy">
<input type="submit" name="submit" value="送出">
</form>
```

tom1.jsp 关键代码：

```
<% String sid=session.getId();
    String str=response.encodeURL("jerry1.jsp");
%>
```

```
<p>在 tom1.jsp 中的 id 号为：<%=sid%>
<p>您向 http://localhost:8080/jerry1.jsp 发送的消息为<%=str%>
<p>单击超连接，可到 jerry1 页面
<a href="<%=str%>">
<br>欢迎到 jerry1 屋来。
</a>
```

jerry1.jsp 关键代码：

```
<% String sid=session.getId();
    String str=response.encodeURL("session1.jsp");
%>
<p>在 jerry1.jsp 中的 id 号为：<%=sid%>
<br>
<p>您向 http://localhost:8080/session1.jsp 发送的消息为<%=str%>
<br>
<p>单击超连接，可到 session1 页面
<a href="<%=str%>">
<br>欢迎到 session1 屋来。
</a>
```

3.4.3 在会话对象中保存数据

保存和获得 session 对象中的数据是 JSP 页面常做的工作，需要使用 getAttribute()和 setAttribute()方法。下面举一个简单的例子 simplesession.jsp 在 session 对象中保存数据 numVisits，这样服务器可以告诉客户他访问该页面一共多少次，只要客户不关闭浏览器。运行结果如图 3-23 所示。

图 3-23 Simplesession.jsp 的运行结果

simplesession.jsp 关键代码:

```
<% String count=(String)session.getAttribute("numVisits");
    count=increment(count);
    out.println("you have visit this page :"+count+"times.");
    session.setAttribute("numVisits",count);
%>
<%!
  String increment(String count)
  {
    if(count!=null)
    return Integer.toString(Integer.parseInt(count)+1);
    else
    return "1";
  }
%>
```

下面的例子简单模拟网上购物过程。客户首先进入银座商场的主页 example3_17.jsp,输入姓名后连接到 first.jsp，在其中输入要购买的货物并连接到结账处。结账处的页面是 count.jsp，此页面显示用户的姓名、购买的货物等信息。运行结果如图 3-24 所示。

图 3-24　example3_17.jsp 的运行结果

example3_17.jsp 关键代码：

```
<% session.setAttribute("custom","顾客");  //将顾客对象加入 session 中，关键字为 custom
%>
<p>输入您的名字，连接到银座商场。
<form action="first.jsp" method=post name=form>
<input type="text" name="boy">
<input type="submit" name="submit" value="送出">
</form>
```

first.jsp 关键代码：

```
<% String nm=request.getParameter("boy");
   session.setAttribute("name",nm);  //将 nm 对象加入 session 中，并指定关键字为 name
%>
<p>这里是银座商场,请输入您购买的商品，连接到结账处。
<form action="count.jsp" method=post name=form>
<input type="text" name="buy">
<input type="submit" name="submit" value="送出">
</form>
```

count.jsp 关键代码：

```
<%! 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;
       }
%>
<% String pa=request.getParameter("buy");
   session.setAttribute("goods",pa);   //将 nm 对象加入 session 中，并指定关键字为 name
%>
<p>这里是结账处。
<% String cus=(String)session.getAttribute("custom");
     String nam=(String)session.getAttribute("name");
     String goo=(String)session.getAttribute("goods");
     nam=getString(nam);
     goo=getString(goo);
%>
<br>
<p><%=cus%>的姓名是：<%=nam%>
<p>您选购的商品是：<%=goo%>
```

3.4.4 session 对象的生存时间

下面的例子是一个猜数字游戏，当客户访问 example3_18.jsp 时，页面产生一个 1 到 100 之间的随机数，并将其存入客户的 session 对象，然后让客户猜这个数。客户输入数后提交给 result.jsp，该页面将客户猜的数和产生的随机数做比较，若猜对了，就连接到 success.jsp。该页面显示客户猜数所用的时间等信息。如果猜的数大于随机数，连接到 large.jsp；否则连接到 small.jsp。这两个页面都要求客户重新猜数。运行结果如图 3-25 所示。

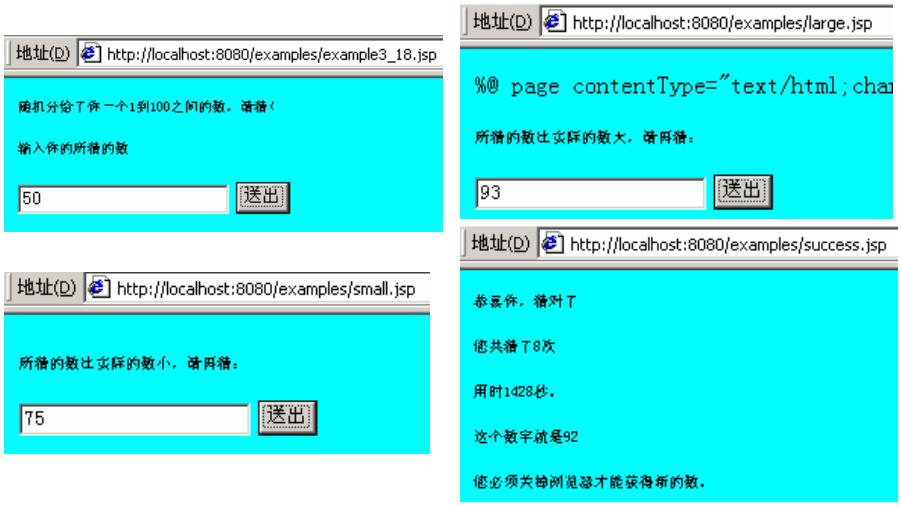


图 3-25　example3_18.jsp 的运行结果

example3_18.jsp 关键代码：

```
<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>
```

result.jsp 关键代码：

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

large.jsp 关键代码：

```
<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>
```

small.jsp 关键代码：

```
<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>
```

success.jsp 关键代码：

```
<%   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
<BR>
<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。
```

3.4.5 session 对象的新与旧

前面已经讲到，会话是有生存期的。当客户首次访问服务器上的 JSP 页面时，JSP 引擎产生 session 对象，直到客户关闭浏览器。在此期间，客户在该服务器不同网页间转换或从其他服务器回到该服务器时，始终使用的是同一个 session 对象。一般来说，下列情况会使会话结束：由于网络故障套接字删除、关闭浏览器、服务器关闭重起、会话超时和主动撤销会话。

在第 2 章已经实现了计数器功能，显示客户是访问本网站的第多少位客户。但那时有个问题没有解决，就是当客户不断刷新页面时，计数器的数目在增加，这是不正确的。下面的例子解决了这个问题。example3_19.jsp 对第 2 章的例子进行了改进，检查客户的 session 对象是否是新的，若是就让计数器数增 1；否则计数器的数目不变。

example3_19.jsp 关键代码：

```
<%! int number=0;
 synchronized void countPeople()
     {
       if(number==0)
           {
             try{File f=new File("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("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("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())   //检查 session
         {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。
```

3.5　application 对象

application 对象负责提供应用程序在服务器中运行时的一些全局信息。服务器启动时就创建一个 application 对象，当一个客户访问服务器上的一个 JSP 页面时，JSP 引擎就为该客户分配这个 application 对象，客户在该服务器的各个页面之间连接时，application 对象都是同一个。而且所有访问该服务器的客户共享同一个 application 对象，直到服务器关闭。application 对象是 javax.servlet.ServletContext 类的一个实例，其主要方法是：

```
String GetServerInfo() //获得服务器信息
int getMajorVersion() //返回 Servlet api 的主要版本
int getMiniVersion()//返回 Servlet api 的次要版本
String getRealPath (path)//将 url 转成文件系统路径名
String getMimeType() //获得 MIME 类型
String getResource(path)//返回指定路径的 url
InputStream getResourceAsStream(path) //获得指定文件的输入流
ServletContext getContext()//得到 ServletContext
RequestDispatcher getRequestDispatcher(path) //获得请求分发器
void log(message)   //日志中写消息
void log(message,exception)//日志中写消息和异常的堆栈跟踪
```

Object getAttribute(Object key) //获得 application 对象中指定属性的值
void setAttribute(Object key,Object value) //设定 application 对象中指定属性的值
Enumeration getAttributeNames() //得到 application 对象中所有属性的名字的枚举

下面举例说明 application 对象的使用。

3.5.1 application 对象的常用方法

下例中，application.jsp 对 application 对象的几个常用方法作了演示，运行结果如图 3-26 所示。

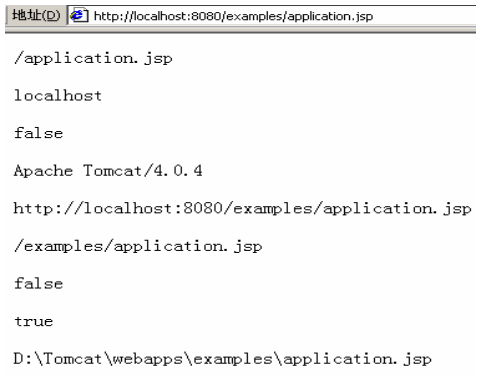


图 3-26 application.jsp 的运行结果

application.jsp 关键代码：

```
<%=request.getServletPath() %><p>
<%=request.getServerName()%><p>
<%=request.isSecure()%><p>
<%--这是 HttpServletRequest 中的方法--%>
<%=application.getServerInfo() %><p>
<%--返回 Servlet 所在的 server 版本号，形式为 servername/versionnumber--%>
<%=request.getRequestURL() %><p>
<%=request.getRequestURI() %><p>
<%=request.isRequestedSessionIdFromURL()%><p>
<%=request.isRequestedSessionIdFromCookie()%><p>
<%--这是 HttpServletRequest 中的方法--%>
<%=application.getRealPath(request.getServletPath()) %>
```

下例中，setAttribute.jsp 页面中分别在 session 对象和 application 对象中设定属性。在 getAttribute.jsp 页面中分别取出属性，修改后再回写进去。运行该例子时，打开两次浏览器，会发现 application 对象中修改的属性值在两个浏览器打开的页面中始终保持，而 session 对象中的属性值在两个浏览器打开的页面中没能保持。运行结果如图 3-27 所示。

setAttribute.jsp 关键代码：

```
<%application.setAttribute("appAttributeName","appAttributeValue"); %>
<%session.setAttribute("sessAttributeName","sessAttributeValue"); %>
<%application.setAttribute("appName","1"); %>
<%session.setAttribute("sessName","1"); %>
```

图 3-27　application 对象与 session 对象比较

getAttribute.jsp 关键代码：

```
<b>the value of appAttributeName is:<%=application.getAttribute("appAttributeName") %><p></b>
<b>the value of sessAttributeName is:<%=session.getAttribute("sessAttributeName") %><p></b>
<b>the value of appName is:<%=application.getAttribute("appName") %><p></b>
<b>the value of sessName is:<%=session.getAttribute("sessName") %><p></b>
<%String app=(String)application.getAttribute("appName");
application.setAttribute("appName",String.valueOf(Integer.parseInt(app)+1)); %>
<%String sss=(String)session.getAttribute("sessName");
session.setAttribute("sessName",String.valueOf(Integer.parseInt(sss)+1)); %>
```

下例 example3_20.jsp 中，将计数器放在 application 对象中，即用 application 对象实现网站计数器功能。运行结果与第 2 章中的 example2_3.jsp 相同。

example3_20.jsp 关键代码：

```
<%!
 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。
```

这个例子有待改进，因为没有将计数器存在文件中，因此如果服务器关闭，将造成计数器的丢失。如何改进留给读者自己完成。

3.5.2 用 application 对象制作留言板

在下例中，客户在 submit.jsp 中输入姓名、留言标题和留言内容。这些信息提交给 messagePane.jsp。该页面将客户信息存入向量，并将向量存放在 application 对象中。当查看留言板时，showMessage.jsp 负责从 application 对象中取出向量，并遍历向量显示其中的所有留言。

向量是 java.util.Vector 类，它相当于一个长度可变的一维数组。向量的最基本方法是 add(int index, Object o)方法和 elementAt(int index)方法。add()方法把一个对象插入到 index 指定的位置，elementAt()方法取出向量中 index 位置的对象。应注意的是，向量中的元素可以是不同数据类型。本例的运行结果如图 3-28 所示。

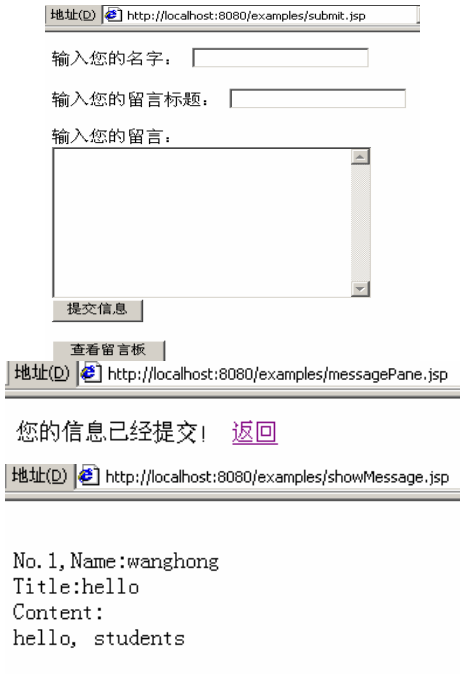


图 3-28 留言板的运行结果

submit.jsp 关键代码：

```
<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>
```

messagePane.jsp 关键代码：

```
    <%! 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");
         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
```

showMessage.jsp 关键代码：

```
    <% 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);
                  }
          }
     %>
```

3.6 page 对象

它是 this 的同义词，表示 JSP 页面本身，是 Servlet 类的一个实例，就是转换后的 Servlet 类，可以调用 Servlet 中的任何方法。它实现了 javax.servlet.jsp.HttpJspPage 接口。当前 page 对象用处不大，它是为了 Java 不再是惟一的 JSP 编程语言而准备的占位符。

3.7 config 对象

config 对象是 javax.servlet.servletConfig 接口的一个实例，存储 Servlet 配置对象。其重要方法有：

Enumeration getInitParameterNames() //获得 servletConfig 中的初始化参数名
String getInitParameter(name) //获得 servletConfig 中指定初始化参数的值
ServletContext getServletContext() //获得 Servlet 上下文
config 对象在 JSP 中很少用到。

3.8 exception 对象

exception 对象代表了 JSP 文件运行时所产生的例外对象，目的是在 JSP 内处理错误。此对象不能在一般 JSP 文件中直接使用，而只能在使用了<%@ page isErrorPage="true "%>的 JSP 文件中使用。它是 java.lang.Throwable 类的一个实例，其主要方法有：

getMessage() //获得异常信息
toString() //获得异常信息和相关描述
printStackTrace() //获得产生异常时的调用栈情况
下例中，dividedexample.jsp 出现被 0 除异常，转到出错页 error.jsp，在该页面获得异常信息。运行结果与第 2 章 compute.html 的相同。

dividedexample.jsp 关键代码：

```
<%@page contentType="text/html;charset=GBK" errorPage="error.jsp"%>
<%int a=19;
int b=0;
int c=a/b;
%>
```

error.jsp：

```
<%@page  isErrorPage="true"%>
<b><i>Something bad just happened:<%=exception.getMessage()%></i></b>
```

3.9 pageContext 对象

pageContext 对象主要用来管理页面的属性，描述 JSP 文档的运行环境。它是页面中对象功能的最大集成者，提供对所有其他隐式对象及其属性的访问。PageContext 对象是 javax.servlet.jsp.PageContext 类的一个实例，其主要方法有：

HttpSession getSession () //获得 session 对象
Objetc getPage () //获得 page 对象
ServletRequest getRequest() //获得 request 对象
ServletResponse getResponse() //获得 response 对象
JspWriter getOut()　//获得 out 对象
ServletConfig getConfig() //获得 config 对象
ServletContext getContext() //获得 application 对象
Exception getException()　//获得 exception 对象
void forward(path)　//请求转向指定页面
void include(path)　//动态包含指定文件
void setAttribute(key,value,scope) //在某隐含对象范围内设定属性值
Objetc getAttribute(key,scope)　//获取某隐含对象范围内设定属性值
void removeAttribute(key,scope) //取消某隐含对象范围内设定属性值

其中，scope 取以下静态常量：PAGE_SCOPE、REQUEST_SCOPE、SESSION_SCOPE、APPLICATION_SCOPE。

下例中，scopeexample1.jsp 在 pageContext 对象中存放一些属性，在 scopeexample2.jsp 中获取这些属性。运行结果如图 3-29 所示。

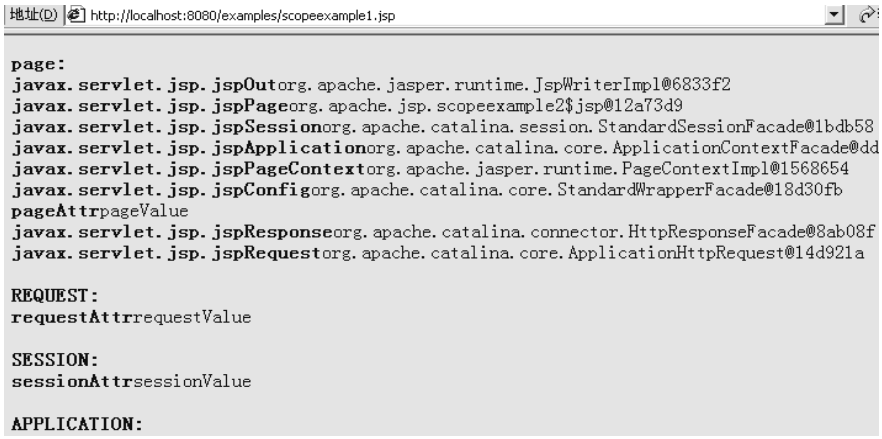


图 3-29　scopeexample1.jsp 运行结果

scopeexample1.jsp 关键代码：

```
Value being seting...
<% pageContext.setAttribute("pageAttr","pageValue",PageContext.PAGE_SCOPE);
    pageContext.setAttribute("requestAttr","requestValue",PageContext.REQUEST_SCOPE);
    session.setAttribute("sessionAttr","sessionValue");
    application.setAttribute("appAttr","appValue");
    pageContext.forward("scopeexample2.jsp");
    %>
```

scopeexample2.jsp 关键代码：

```
<% pageContext.setAttribute("pageAttr","pageValue",PageContext.PAGE_SCOPE);
%>
<% java.util.Enumeration attribNames;
  int
```

```
scopes[]={PageContext.PAGE_SCOPE,PageContext.REQUEST_SCOPE,PageContext.SESSION_SCOPE,Pag
eContext.APPLICATION_SCOPE};
 String attribName;
 for(int i=0;i<4;i++)
 {
       switch(scopes[i])
       {
            case PageContext.PAGE_SCOPE:out.println("<b>page:</b><br>");
            break;
           case PageContext.REQUEST_SCOPE:out.println("<b>REQUEST:</b><br>");
            break;
           case PageContext.SESSION_SCOPE:out.println("<b>SESSION:</b><br>");
            break;
            case PageContext.APPLICATION_SCOPE:out.println("<b>APPLICATION:</b><br>");
            break;
       }
       attribNames=pageContext.getAttributeNamesInScope(scopes[i]);
       while(attribNames.hasMoreElements())
       {
            out.print("<b>"+(attribName=(String)attribNames.nextElement())+"</b>");
            out.print(pageContext.getAttribute(attribName,scopes[i]));
            out.println("<br>");
       }
       out.println("<p>");
 }
 %>
```

在结果中看到很多其他信息，这些信息有些是以前设定的，有些是系统工作时设定的。

本章小结

在 JSP 代码片段中，可以利用隐含对象与 JSP 页面的代码片段执行环境产生互动。本章介绍了 JSP 页面中使用的主要隐含对象，它们是：

- request：客户端请求，包括从 GET/POST 请求传递过来的参数。
- response：网页传回客户端的响应。
- pageContext：在此管理网页属性。
- session：与请求关联的会话。
- application：代码片段的运行环境。
- out：传送响应的输出流。
- config：代码片段配置对象。
- page：JSP 网页本身。
- exception：有错的网页中未被捕获的例外。

正是由于这些隐含对象，你可以进入执行 JSP 代码的代码片段，用不着深入了解太多的 Servlet API 细节。例如，你可以不用表达式，直接进入“out”隐含对象，将某些内容输出到

响应中：

```
<% out.println("Hello"); %>
```

再如，用不着把一个参数直接送到 JavaBean，你可以从请求对象获取参数值：

```
<% String name=request.getParameter("name"); out.println(name); %>
```

从本质上讲，JSP 的这些内置对象其实都是由特定的 Java 类所产生的，在服务器端运行时根据情况自动生成，所以如果你有较好的 Java 基础，可以参考相应的类说明，表 3-3 给出了它们的对应关系。

表 3-3　JSP 内置对象映射表

对象名	类型	作用域
request	javax.servlet.ServletRequest 的子类	Request
response	javax.servlet.ServletResponse 的子类	Page
pageContext	javax.servlet.jsp.PageContext	Page
session	javax.servlet.http.Httpsession	Session
application	javax.servlet.ServletContext	Application
Out	javax.servlet.jsp.JspWriter	Page
config	javax.servlet.ServletConfig	Page
Page	java.lang.Object	Page
exception	java.lang.Throwable	Page

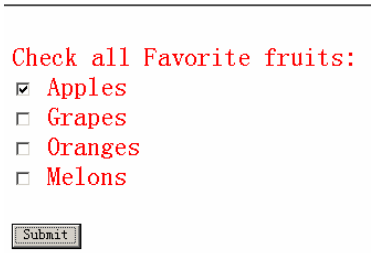
习　　题

1．编写 JSP 页面，完成下图所示的功能：

Request Information

JSP Request Method: GET
Request URI: /examples/jsp/snp/snoop.jsp
Request Protocol: HTTP/1.1
Servlet path: /jsp/snp/snoop.jsp
Path info: null
Path translated: null
Query string: null
Content length: -1
Content type: null
Server name: localhost
Server port: 8080
Remote user: null
Remote address: 127.0.0.1
Remote host: shdz243
Authorization scheme: null
Locale: zh_CN

2．编写 JSP 页面，完成下图所示的功能，提交请求后显示相应的选中项标签：



3．选择正确答案。

在 aa.jsp 中有行代码：

```
<% request.setAttribute("Co.","jb-aptech"); %>
```

在 bb.jsp 中有行代码：

```
<% out.println((String)request.getAttribute("Co.")); %>
```

为了使在 bb.jsp 中的如上代码可以显示"jb-aptech"，可以使用（　　）发送。

A）在 aa.jsp 中使用<form method=post action="bb.jsp">把请求提交到 bb.jsp

B）在 aa.jsp 中使用<jsp:forward file="bb.jsp"/>把页面重定向到 bb.jsp

C）在 aa.jsp 中使用<% response.sendRedirect("bb.jsp");%>把页面重定向到 bb.jsp

D）在 aa.jsp 中使用<%@ include file="bb.jsp" %>包含页面 bb.jsp

E）在 aa.jsp 中使用

<%

config.getServletContext().getRequestDisqatcher("/bb.jsp").forward(request.response);

%>把页面重定向到 bb.jsp

4．选择正确答案。

如果在 JSP 脚本中有如下代码：

```
int I=10;                          //1
String str=”jb-aptech”;            //2
Vector v=new Vector();             //3
v.add("jb");                       //4
v.add("aptech");                   //5
session.setAttribute("I",I);       //6
session.setAttribute("str".str);   //7
session.setAttribute("v",v);       //8
```

以下正确的选项是（　　）

A）第 6、7、8 行代码是错误的

B）修正第 1 到第 8 行中错误的代码后，使用

String str=(String)session.getAttribute("str");可以取出属性 str 的值

C）修正第 1 到第 8 行中错误的代码以后，使用 int x=(int)session.getAttribute("I");此时 I 的值为 10

D）修正第 1 到第 8 行中错误的代码以后，可以使用 Object v=session.getAttribute("v");取得属性 v 的值

实验二　JSP 隐含对象的使用

一、实验目的

练习在 JSP 中使用隐含对象

二、实验工具

（1）JDK1.4 或 JDK1.5，可以从 SUN 公司的网站免费下载
（2）Tomcat 服务器，也可以从网上免费下载

三、实验步骤

1．response 对象的使用

第 1 步：打开记事本，键入以下文档，并保存为 login.html。

```
<html>
<h1>Login Page<h1>
<body>
<form action="authenticate.jsp" method="post">
      <b>UseerName:</b><input type="text" name="username"><br>
      <b>PassWord:</b><input type="password" name="password"><br>
      <input type="submit" value="Submit">
</form>
</body>
</html>
```

第 2 步：打开记事本，键入以下文档，并保存为 autho.jsp。

```
<html>
<body bgcolor="#ffcccc">
<%
if(request.getParameter("password").equals("wangjianme"))
{
      out.println("Welcome :"+request.getParameter("username"));
}
else
{
      response.sendRedirect("login.jsp");
}
%>
</body>
</html>
```

第 3 步：将 login.html 和 autho.jsp 拷贝到 Tomcat 安装目录下的 webapps/examples/。

第 4 步：在 IE 地址栏中键入 http://localhost:8080/examples/login.html。

2．session 对象的使用

第 1 步：打开记事本，键入以下文档，并保存为 visit.jsp。

```
<html>
<body>
<%
String count=(String)session.getAttribute("numVisits");
count=increment(count);
out.println("You have visited this page "+count+" times.");
session.setAttribute("numVisits",count);
%>
<%!
String increment(String count)
{
    if(count!=null)
    {
        return Integer.toString(Integer.parseInt(count)+1);
    }
    else
    {
        return "1";
    }
}
%>
</body>
</html>
```

第 2 步：将 visit.jsp 拷贝到 Tomcat 安装目录下的 webapps/examples/。

第 3 步：在 IE 地址栏中键入 http://localhost:8080/examples/visit.jsp。

3．pageContext 对象的使用

第 1 步：打开记事本，键入以下文档，并保存为 scope1.jsp。

```
<html>
<body bgcolor="#ffccac">
values being set.....
<% pageContext.setAttribute("pageAtri","pageVlaue",PageContext.PAGE_SCOPE);
pageContext.setAttribute("requestAttr","requestValue",PageContext.REQUEST_SCOPE);
    session.setAttribute("sessionSttr","sessionValue");
    application.setAttribute("appAttr","appValue");
    pageContext.forward("scopeExample2.jsp");
 %>
</body>
</html>
```

第 2 步：打开记事本，键入以下文档，并保存为 scope2.jsp。

```
<%@ page import="java.util.*" %>
<html>
<body>
<%
pageContext.setAttribute("pageAttr","pageValue",PageContext.PAGE_SCOPE);
Enumeration names;
```

```
int scopes[]={PageContext.PAGE_SCOPE,
                    PageContext.REQUEST_SCOPE,
                    PageContext.SESSION_SCOPE,
                    PageContext.APPLICATION_SCOPE};
String attriName;
for(int i=0;i<4;i++)
{
    switch(scopes[i])
    {
        case PageContext.PAGE_SCOPE:
            out.println("<b>Page:</b><br>");
            break;
        case PageContext.REQUEST_SCOPE:
            out.println("<B>Request:</b><br>");
            break;
        case PageContext.SESSION_SCOPE:
            out.println("<b>Session:</b><br>");
            break;
        case PageContext.APPLICATION_SCOPE:
            out.println("<b>Application:</b><br>");
            break;
    }
    names=pageContext.getAttributeNamesInScope(scopes[i]);
    while(names.hasMoreElements())
    {
        attriName=(String)names.nextElement();
        out.println("<b>"+attriName+":</b>");
        out.println(pageContext.getAttribute(attriName,scopes[i]));
        out.println("<br>");
    }
    out.println("<br>");
}
%>
</body>
</html>
```

第 3 步：将 scope1.jsp 和 scope2.jsp 拷贝到 Tomcat 安装目录下的 webapps/examples/。

第 4 步：在 IE 地址栏中键入 http://localhost:8080/examples/scope1.jsp。

四、实验练习

练习 1：编写两个页面，第一个页面在 application 对象中设置属性，第二个页面取出该属性并显示输出。

练习 2：编写程序显示客户的国家和语言等信息。

练习 3：编写程序故意抛出异常，异常由该页面的错误页捕捉。