第3章 JSP隐含对象

前一章主要介绍了JSP页面的组成元素及其使用。本章主要介绍如何简化页面开发的复杂性,JSP提供了一些由容器实现和管理的隐含对象,也就是说,用户不需要实例化就可以直接使用的对象,通过存取这些隐含对象实现与JSP页面的Servlet环境的相互访问,因此要求用户必须熟悉这些对象,并掌握其主要方法的使用。

- 对象的4种有效范围
- JSP有9个隐含对象

JSP容器提供了以下几个隐含对象,它们是:request、response、out、session、application、config、page、pageContext和exception。由于JSP是构建在Servlet上的,从本质上来讲,JSP的每个隐含对象都与Java Servlet API包中的类相对应,在服务器运行时自动生成。本章对它们进行详细的介绍。

3.1 对象的属性与有效范围

1. 对象的属性

JSP技术提供给开发人员一种传递数据的机制,就是利用setAttribute()和getAttribute()方法。

1) setAttribute()方法

setAtrribute是应用服务器把对象放在该页面对应的一块内存中,当页面服务器重定向到另一个页面中时,应用服务器会把这块内存拷贝到另一个页面所对应的内存中。这种方式可以传对象,对于不同的隐含对象,这个传递的对象在内存中的生命周期不同。

方法的声明格式:

```
void setAttribute(String name,Object object)
```

说明:

将对象object绑定到Servlet上下文中提供的属性名称name中,如果指定的名称已经

使用,则方法 setAttribute()将该属性值替换为新的属性值。

2) getAttribute()方法

getAttribute 用于获取容器中的数据。但它只能接收 setAttribute 传过来的数据值。方法的声明格式:

```
Object getAttribute(String name)
```

说明:

返回指定名称的属性值,如果属性不存在,则返回 null。与 setAttribute 方法配合使用可实现两个 JSP 文件之间的参数传递。

例如,定义类 User(带参数的构造方法及 getName()方法)的对象 curruser:

```
User curruser = new User("susan", 20, "女");
```

则 request. setAttribute("curruser", curruser)方法就是将 curruser 这个对象保存在 request 作用域中,然后在转发进入的页面就可以获取值,可以在 JSP 页面编写 Java 小脚本来获取:<% User myuser = (User)request. getAttribute("curruser")%>,在 JSP 页面显示值:<%=myuser. getName()%>。

2. 对象的有效范围

有时会将 pageContext、request、session 和 application 归为一类,原因在于它们都借助以上 setAttribute()和 getAttribute()方法来设定和取得其属性。而这 4 个隐含对象最大的区别在于其范围不同。

1) 页内有效

具有页内有效范围的对象被绑定到 javax. servlet. jsp. PageContext 对象中。在这个范围内的对象,只能在创建对象的页面中访问。可以调用 pageContext 这个隐含对象的 getAttribute()方法来访问具有这种范围类型的对象(pageContext 对象还提供了访问其他范围对象的 getAttribute 方法),pageContext 对象本身也属于 page 范围。page 范围内的对象,在客户端每次请求 JSP 页面时创建,在页面向客户端发送回响应或请求被转发(forward)到其他的资源后被删除。

例 3.1 在同一 JSP 页面 pageSetGet. jsp 中利用 pageContext 对象、利用 setAttribute()和 getAttribute()两个方法实现对参数的设值和取值。代码如下:

```
<! -- pageSetGet. jsp -->
<% @ page contentType = "text/html;charset = GB2312" %>
<html>
<head>
   <title>页内有效范围</title>
</head>
<body>
页内有效 - 使用页面上下文对象 pageContext. setAttribute()和 pageContext. getAttribute()
<p/>
<%
    pageContext. setAttribute("歌曲名称","爸爸去哪儿");
%>
<em>
```

```
<%
    String Name = (String)pageContext.getAttribute("歌曲名称");
    out.print(" 歌曲名称 = " + Name);
 %>
</em>
</body>
</html>
```

将该文件放在 FirstJSP 项目下，运行界面如图 3.1 所示。

图 3.1　同一页面的 pageContext 对象的执行结果

在同一个页面内，参数的设值和取值均能实现。

如果现在将设值和取值分别放在两个不同的页面，结果会是如何呢？

例 3.2　修改例 3.1，参数的设置放在 pageSet.jsp 中，代码如下：

```
<!-- pageSet.jsp -->
<%@ page contentType = "text/html;charset = GB2312" %>
<html>
<head>
   <title>页内有效范围</title>
</head>
<body>
<
页内有效 - 使用页面上下文对象 pageContext.setAttribute()
<%
    pageContext.setAttribute("歌曲名称","爸爸去哪儿");
 %>
<jsp:forward page = "pageGet.jsp"/>
</body>
</html>
```

参数的取值放在 pageGet.jsp 中，代码如下：

```
<!-- pageGet.jsp -->
<%@ page contentType = "text/html;charset = GB2312" %>
<html>
<head>
 <title>页内有效范围</title>
</head>
```

```
<body>
<h3>页内有效 - 使用页面上下文对象 pageContext.getAttribute()</h3>
<em>
<%
    String Name = (String) pageContext.getAttribute("歌曲名称");
    out.print(" 歌曲名称 = " + Name);
%>
</em>
</body>
</html>
```

将两个文件放在 FirstJSP 项目下，执行 http://localhost：8080/FirstJSP/pageSet.jsp 后运行界面如图 3.2 所示。

图 3.2　不同页面的 pageContext 对象的执行结果

2）请求有效

请求有效的对象在同一请求不同 JSP 页面内都可以访问。如果请求转向到同一运行时(Runtime)的其他资源，那么这些对象依然有效。请求有效的对象在请求处理结束时就会失效。所有的请求有效的对象都存储在 JSP 页面的 request 对象中。

request 对象在服务器启动时自动创建，是 javax.servlet.HttpServletRequest 接口的一个实例。request 对象的作用是与客户端交互，收集客户端的 Form、Cookies、超链接或收集服务器端的环境变量。

例 3.3　修改例 3.2 中的 pageSet.jsp 和 pageGet.jsp，将 pageContext 对象修改为 request 对象。

requestSet.jsp 代码如下：

```
<!-- requestSet.jsp -->
<%@ page contentType = "text/html;charset = GB2312" %>
<html>
<head>
   <title>请求有效范围</title>
</head>
<body>
<
请求有效 - 使用 request 对象 request.setAttribute()
<%
```

```
    request.setAttribute("歌曲名称","爸爸去哪儿");
%>
<jsp:forward page = "requestGet.jsp"/>
</body>
</html>
```

requestGet.jsp 代码如下：

```
<!-- requestGet.jsp -->
<%@ page contentType = "text/html;charset = GB2312" %>
<html>
<head>
   <title>请求有效范围</title>
</head>
<body>
<h3>请求有效 - 使用 request 对象 request.getAttribute()</h3>
<em>
<%
    String Name = (String)request.getAttribute("歌曲名称");
    out.print(" 歌曲名称 = " + Name);
%>
</em>
</body>
</html>
```

运行结果如图 3.3 所示。

图 3.3　请求有效

从两个运行结果可以看出，request 对象在不同的页面进行设值和取值显示正常，也就是说，两个页面之间的请求有效。如果在第三个页面中再获取参数将会失败。除了利用转向(forward)的方法可以存取 request 对象的数据外，还能使用包含(include)的方法，观察两者的异同。

3）会话有效

会话是指客户端和服务器之间持续连接的一段时间。在这段时间内，当需要多次和服务器交互信息时，可以将有关信息存入 session 对象中，这些信息是会话有效的。在与服务器断线后，就会失效。可以说，session 的作用范围就是一段用户持续和服务器连接的时间。

会话有效的所有对象都存储在 JSP 页面的 session 对象中。

例 3.4 修改例 3.3，修改 requestSet.jsp 为 seesionSet.jsp，requestGet.jsp 修改为 sessionGet1.jsp，并在该代码中添加<jsp：include >，包含 sessionGet2.jsp 页面，重新获取参数并显示。

seesionSet.jsp 代码如下：

```
<!-- sessionSet.jsp -->
      <%@ page contentType = "text/html;charset = GB2312" %>
      <html>
      <head>
         <title>会话有效范围</title>
      </head>
      <body>
      会话有效 - 使用 session 对象 session.setAttribute()
      <%
      session.setAttribute("歌曲名称","爸爸去哪儿");
      %>
      <jsp:forward page = "sessionGet1.jsp"/>
    </body>
    </html>
```

sessionGet1.jsp 代码如下：

```
<!-- sessionGet1.jsp -->
<%@ page contentType = "text/html;charset = GB2312" %>
<html>
<head>
  <title>会话有效范围</title>
</head>
<body>
<h3>会话有效 - 使用 session 对象 session.getAttribute()</h3>
<em>
<%
    String Name = (String)session.getAttribute("歌曲名称");
    out.print(" 歌曲名称 = " + Name);
%>
<jsp:include page = "sessionGet2.jsp"/>
</em>
</body>
</html>
```

sessionGet2.jsp 代码如下：

```
<!-- sessionGet2.jsp -->
<%@ page contentType = "text/html;charset = GB2312" %>
<html>
<head>
  <title>会话有效范围</title>
</head>
<body>
<h3>会话有效 2 - 使用 session 对象 session.getAttribute()</h3>
<em>
```

```
<%
    String Name = (String)session.getAttribute("歌曲名称");
    out.print(" 歌曲名称重新显示 = " + Name);
 %>
</em>
</body>
</html>
```

运行效果如图 3.4 所示。

图 3.4　session 对象

由该例可以看出，只要在一个会话中，多个页面都可以获取参数。但是如果是在多个会话之间，则不能获取参数。

4）应用有效

应用有效的作用范围是从 Web 应用服务器一开始执行服务直到结束服务为止。应用有效范围最大、影响最长。应用有效的对象都存储在 JSP 页面的 application 对象中，其实就是服务端 Servlet 上下文信息对象（ServletContext）。在实际使用时注意不要使用过多，以免造成服务器负载过大。

例 3.5　修改例 2.4 中的登录页面 content.jsp，在代码最后添加如下脚本：

```
<%
    application.setAttribute("manager", request.getParameter("manager"));
    application.setAttribute("pwd", request.getParameter("pwd"));
 %>
```

另创建一个 TestApplication.jsp 页面，与 content.jsp 表面上没有关系，代码如下：

```
<!-- TestApplication.jsp -->
<%@ page language = "java" contentType = "text/html; charset = GBK" %>
<% request.setCharacterEncoding("GBK"); %>
<html>
    <body>
        用户名:<% = application.getAttribute("manager") %><br>
        密  码:<% = application.getAttribute("pwd") %><br>
    </body>
</html>
```

运行 content.jsp，如图 3.5 所示，并在“管理员”文本框中输入 application，在“密码”文本框中输入 test，然后单击“确定”按钮。

图 3.5 有关 application 的登录界面

再运行 TestApplication.jsp 页面，显示效果如图 3.6 所示。

图 3.6 利用 application 对象获取信息

可以看出，访问同一个网站的客户都共享一个 application 对象，因此，application 对象可以实现多客户间的数据共享。

3.2 JSP 的隐含对象

JSP 2.0 中有 9 个隐含对象，其名称、实现类及说明如表 3.1 所示。

表 3.1　9 个隐含对象

JSP 隐含对象	实现类	对 象 说 明
out	JspWriter	HTML 标准输出
request	HttpServletRequest	请求信息
response	HttpServletResponse	响应信息
application	ServletContext	服务端 Servlet 上下文信息
session	HttpSession	HTTP 联机会话信息
config	ServletConfig	JSP 页面的 Servlet 配置信息，由 Web 应用配置描述文件(web.xml)指定
exception	Throwable	异常处理信息
page	Object	如同 Java 中的 this
pageContext	PageContext	当前 JSP 页面的上下文信息

3.2.1　案例 1：使用 out 隐含对象

第 2 章介绍过表达式可以完成输出，但是表达式在求值之后的结果转换成了 String 对象，该对象被发送到 out 对象中，也就是说，客户端浏览器中显示的信息，就是服务器端通过 out 对象实现的。

out 对象的常用方法如表 3.2 所示。

表 3.2　out 常用方法

方 法 名 称	方 法 说 明
public abstract void clear()	清除缓冲区中的内容，不将数据发送至客户端
public abstract void clearBuffer()	将数据发送至客户端后，清除缓冲区中的内容
public abstarct void close()	关闭输出流
public abstract void flush()	输出缓冲区中的数据
public int getBufferSize()	获取缓冲区的大小。缓冲区的大小可用<%@ page buffer="size" %>设置
public abstract int getRemainning()	获取缓冲区剩余空间的大小
public boolean isAutoFlush()	获取用<%@ page is AutoFlush="true/false"%>设置的 AutoFlush 值
public abstract void newLine()	输出一个换行字符，换一行
public abstract void print()	显示各种数据类型的内容
public abstract void println()	分行显示各种数据类型的内容

在 JSP 中，out 对象主要用来管理响应缓冲和向客户端输出内容。

例 3.6　利用 out.print 和表达式两种方式进行输出。

out1.jsp 采用 out.print 方式输出，代码如下：

```
<!-- out1.jsp -->
<%@ page language="java" buffer="1kb" autoFlush="true" contentType="text/html;charset
 =GB2312"%>
<html>
    <head>
```

```
        <title>out 对象</title>
    </head>
    <body>
    <%
    out.println("<h2>Hello!</h2>");
    out.println("BufferSize of the Out Object is:" + out.getBufferSize() + "<br>");
    out.println("Remain of BufferSize is:" + out.getRemaining() + "<br>");
    %>
    <%
    for(int i = 0;i < 5;i++)
    out.println("<h3>Test</h3>") ;
    %>
    <% out.println("autoFlush:" + out.isAutoFlush()); %>
   </body>
</html>
```

out2.jsp 采用表达式方式输出，代码如下：

```
<!-- out2.jsp -->
<%@ page language = "java" buffer = "1kb" autoFlush = "true" contentType = "text/html;charset =
GB2312" %>
<html>
    <head>
       <title>out 对象</title>
    </head>
    <body>
     <h2>Hello!</h2>
     BufferSize of the Out Object is:<% = out.getBufferSize() %><br>
     Remain of BufferSize is:<% = out.getRemaining() %><br>
     <%
     for(int i = 0;i < 5;i++)
     out.println("<h3>Test</h3>") ;
     %>
     autoFlush:<% = out.isAutoFlush() %>
    </body>
</html>
```

两个页面的运行结果都如图 3.7 所示。

值得注意的是，程序实际在处理时是先将数据放在缓冲区，等 JSP 容器解析完整个程序后才把缓冲区的数据输出到客户端浏览器上。在 page 指令中 buffer="1kb"代表缓冲区为 1kb，autoFlush 属性为 false 时，当数据超过 1kb 时，则出现错误，所以这里设 autoFlush="true"。

3.2.2 案例 2：使用 request 隐含对象

request 对象是从客户端向服务器发出请求，包括用户提交的信息以及客户端的一些信息。客户端可通过 HTML 表单或在网页地址后面提供参数的方法提交数据，然后通过 request 对象的相关方法来获取这些数据。request 的各种方法主要用来处理客户端浏览器提交的请求中的各项参数和选项。常见的方法如表 3.3 所示。

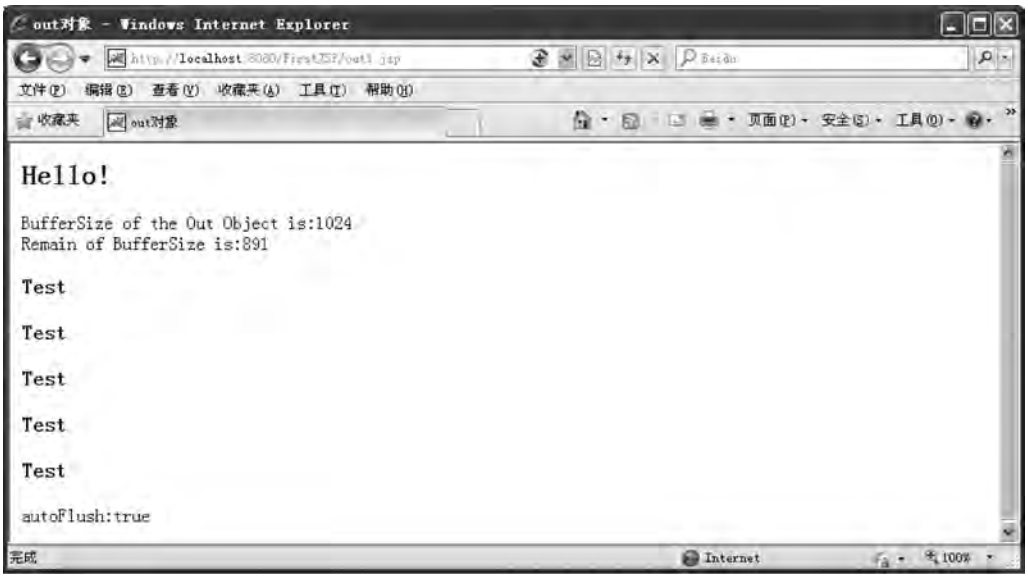

图 3.7　out 对象

表 3.3　request 常用方法

方 法 名 称	方 法 说 明
public java. lang. Object getAttribute(java. lang. String name)	返回以 name 为名字的属性的值。如果该属性不存在,这个方法将返回 null
public java. util. Enumeration getAttributeNames()	返回请求中所有可用的属性的名字。如果在请求中没有属性,这个方法将返回一个空的枚举集合
public void removeAttribute (java. lang. String name)	移除请求中名字为 name 的属性
public void setAttribute(java. lang. String name, java. lang. Object o)	在请求中保存名字为 name 的属性。如果第二个参数 o 为 null,那么相当于调用 removeAttribute(name)
public java. lang. String getCharacterEncoding()	返回请求正文使用的字符编码的名字。如果请求没有指定字符编码,这个方法将返回 null
public int getContentLength()	以字节为单位,返回请求正文的长度。如果长度不可知,这个方法将返回 1
public java. lang. String getContentType()	返回请求正文的 MIME 类型。如果类型不可知,这个方法将返回 null
public ServletInputStream getInputStream()	返回一个输入流,使用该输入流以二进制方式读取请求正文的内容。javax. servlet. ServletInputStream 是一个抽象类,继承自 java. io. InputStream
public java. lang. String getLocalAddr()	返回接收到请求的网络接口的 IP 地址,这个方法是在 Servlet 2.4 规范中新增的
public java. lang. String getLocalName()	返回接收到请求的 IP 接口的主机名,这个方法是在 Servlet 2.4 规范中新增的
public int getLocalPort()	返回接收到请求的网络接口的 IP 端口号,这个方法是在 Servlet 2.4 规范中新增的

续表

方 法 名 称	方 法 说 明
public java. lang. String getParameter(java. lang. String name)	返回请求中 name 参数的值。如果 name 参数有多个值,那么这个方法将返回值列表中的第一个值。如果在请求中没有找到这个参数,这个方法将返回 null
public java. util. Enumeration getParameterNames()	返回请求中包含的所有参数的名字。如果请求中没有参数,这个方法将返回一个空的枚举集合
public java. lang. String[] getParameterValues (java. lang. String name)	返回请求中 name 参数所有的值。如果这个参数在请求中并不存在,这个方法将返回 null
public java. lang. String getProtocol()	返回请求使用的协议的名字和版本,例如: HTTP/1.1
public java. io. BufferedReader getReader() throws java. io. IOException	返回 BufferedReader 对象,以字符数据方式读取请求正文
public java. lang. String getRemoteAddr()	返回发送请求的客户端或者最后一个代理服务器的 IP 地址
public java. lang. String getRemoteHost()	返回发送请求的客户端或者最后一个代理服务器的完整限定名
public int getRemotePort()	返回发送请求的客户端或者最后一个代理服务器的 IP 源端口,这个方法是在 Servlet 2.4 规范中新增的
public RequestDispatcher getRequestDispatcher (java. lang. String path)	返回 RequestDispatcher 对象,作为 path 所定位的资源的封装
public java. lang. String getServerName()	返回请求发送到的服务器的主机名
public int getServerPort()	返回请求发送到的服务器的端口号
public void setCharacterEncoding(java. lang. String env)throws java. io. Unsupported EncodingException	覆盖在请求正文中所使用的字符编码的名字
public HttpSession getSession()	取得与当前请求绑定的 session,如果当前 session 不存在,则为这个请求创建一个新的 session

例 3.7 利用 request 常见方式,显示客户端发送的 HTTP 请求包的信息、获取到的客户端和服务器端的信息。

```
<!-- request.jsp -->
<%@ page contentType = "text/html;charset = GBk" import = "java.util.*" %>
<html>
<head>
<title>request 的使用</title>
</head>
<body>
<p>您的客户端发送的 HTTP 请求头包含如下信息:</p>
<%
Enumeration enum1 = request.getHeaderNames();
while (enum1.hasMoreElements()) {
    String headerName = (String) enum1.nextElement();
    String headerValue = request.getHeader(headerName);
 %>
    <b><% = headerName %></b>:<% = headerValue %><br>
<% } %>
```

```
<p>使用 request 对象的方法获取如下信息:</p>
<%
//服务器
String localName = request.getLocalName();
String serverName = request.getServerName();
String localAddr = request.getLocalAddr();
int localPort = request.getLocalPort();
int serverPort = request.getServerPort(); %>
<b>服务器</b>:<% = localName %><br/>
<b>服务器端 IP</b>:<% = localAddr %><br/>
<b>服务器端口</b>:<% = localPort %><p/>
<%
//客户端信息
String remoteHost = request.getRemoteHost();
String remoteAddr = request.getRemoteAddr();
int remotePort = request.getRemotePort(); %>
<b>浏览器端</b>:<% = remoteHost %><br/>
<b>浏览器端 IP 是</b>:<% = remoteAddr %><br/>
<b>浏览器端口</b>:<% = remotePort %><p/>
<%
//协议相关
String pro = request.getProtocol();
String pro1 = request.getScheme();
int len = request.getContentLength();
String type = request.getContentType();
String charEncode = request.getCharacterEncoding();
 %>
<b>协议版本</b>:<% = pro %><br/>
<b>协议</b>:<% = pro1 %><br/>
<b>数据内容长度</b>:<% = len %><br/>
</body>
</html>
```

运行界面如图 3.8 所示。

在 Web 动态网站技术中,重要的一个环节就是获取从客户端发送的请求信息,如客户端表单登录信息、客户查询信息等,并根据提交信息做进一步操作。在 JSP 程序中,完成从客户端获取数据的方法可以是 getParameter()、getParameterName()和 getParameterValues(),其中比较常用的为 getPatameter()方法。getPatameter()方法与 getAttribute()方法不同,二者的区别主要是:

- getParameter()通过容器的实现来取得类似 post、get 等方式传入的数据;setAttribute()和 getAttribute()只是在 Web 容器内部流转,仅仅是请求处理阶段。
- getParameter()方法传递的数据,会从 Web 客户端传到 Web 服务器端,代表 HTTP 请求数据。getParameter()方法返回 String 类型的数据。setAttribute()和 getAttribute()方法传递的数据只会存在于 Web 容器内部。
- HttpServletRequest 类有 setAttribute()方法,而没有 setParameter()方法。

通常情况下,当一个浏览器向 Web 站点提出页面请求时,首先向服务器发送连接请求,请求的内容包括服务器地址、请求的页面路径等。服务器会将它们组合成确定所请求的页

图 3.8　request 对象方法的使用

面，返回到客户端。客户端向服务器发送数据时，通常采用 get 方法或 post 方法。二者的区别如下：

- get 是从服务器上获取数据；post 是向服务器传送数据。
- get 是把参数数据队列加到提交表单的 action 属性所指的 URL 中，值和表单内各个字段要一一对应，在 URL 中可以看到；post 是通过 HTTP post 机制，将表单内各个字段与其内容放置在 HTML HEADER 内一起传送到 action 属性所指的 URL 地址，用户看不到这个过程。
- 对于 get 方式，服务器端用 request.QueryString 获取变量的值；对于 post 方式，服务器端用 request.Form 获取提交的数据。
- get 传送的数据量较小，不能大于 2KB；post 传送的数据量较大，一般被默认为不受限制。但理论上，IIS4 中最大量为 80KB，IIS5 中为 100KB。
- get 安全性非常低，但是执行效率却比 post 方法好；post 安全性较高。get 方式的安全性较 post 方式要差些，包含机密信息的话，建议用 post 数据提交方式；数据查询时，建议用 get 方式；而数据添加、修改或删除时，建议用 post 方式。

3.2.3　案例 3：使用 response 隐含对象

response 对象是 javax.servlet.ServletResponse 接口中的一个针对 HTTP 协议和实现的子类的实例。Response 对象是表示服务器对请求的响应的 HttpServletResponse 对象，用于动态响应客户端请示，控制发送给用户的信息，并将动态生成响应。

response 对象和 request 对象功能恰好相反，request 对象封装的是客户端提交的信息，而对象封装的是返回客户端的信息。response 对象和 request 对象的作用域相同。response 对象也由容器生成，作为 jspService()方法的参数被传入 JSP。

response 方法可以分为三类：设定表头方法、设定响应状态码和重定向方法。具体如表 3.4 所示。

表 3.4　response 方法

方法名称	方法说明
String getCharacterEncoding()	返回在响应中发送的正文所使用的字符编码(MIME 字符集)
ServletOutputStream getOutputStream()	返回 ServletOutputStream 对象，用于在响应中写入二进制数据
public java. lang. String getContentType()	返回在响应中发送的正文所使用的 MIME 类型
PrintWriter getWriter()	返回 PrintWriter 对象，用于发送字符文本到客户端
void setContentLength(int len)	对于 HTTP Servlet，在响应中，设置内容正文的长度
void setBufferSize(int size)	设置响应正文的缓存大小
void setDateHeader(String name，long date)	指定 long 类型的值到 name 标头
void setHeader(String name，String value)	指定 String 类型的值到 name 标头
void setIntHeader(String name，int value)	指定 int 类型的值到 name 标头
void sendError(int sc，String msg)	传送状态码和错误信息
void setStatus(int sc)	设定状态码
String encodeRedirectURL(String url)	对使用 sendRedirect( )方法的 URL 予以编码
abstract void sendRedirect(String url)	将客户端的响应重定向到指定的 URL(JSP 页面、HTML 页面或 Servlet 等)上

在动态网站的操作中，经常需要从一个页面转向到另一个页面，如登录成功与否，可能需要转向不同的页面。要达到页面重定向的效果，可以采用第 2 章的动作指令 jsp：forward，也可以采用 response 对象的 sendRedirect 方法。该方法的具体格式为：

```
public abstract void sendRedirect(String url)
```

例 3.8　将前面请求有效的例子进行修改，将 requestSet. jsp 中转向页面的< jsp：forward >语句修改为：<%response. sendRedirect("requestGet. jsp")；%>，requestGet. jsp 保持不变，则运行界面如图 3.9 所示。

从执行效果可以看出，动作指令 jsp：forward 与 sendRedirect(String url)方法的不同在于以下几点：

1. 地址栏显示不同

forward 是服务器请求资源，服务器直接访问目标地址的 URL，把那个 URL 的响应内容读取过来，然后把这些内容再发给浏览器。浏览器根本不知道服务器发送的内容从哪里来的，所以它的地址栏还是原来的地址，即仍为 http://localhost：8080/FirstJSP/requestSet. jsp，但显示的内容却是 requestGet. jsp 的结果。

redirect 是服务端根据逻辑，发送一个状态码，告诉浏览器重新去请求那个地址，所以地址栏显示的是新的 URL，即为 http://localhost：8080/FirstJSP/requestGet. jsp。redirect 等于客户端向服务器端发出两次 request，同时也接受两次 response。

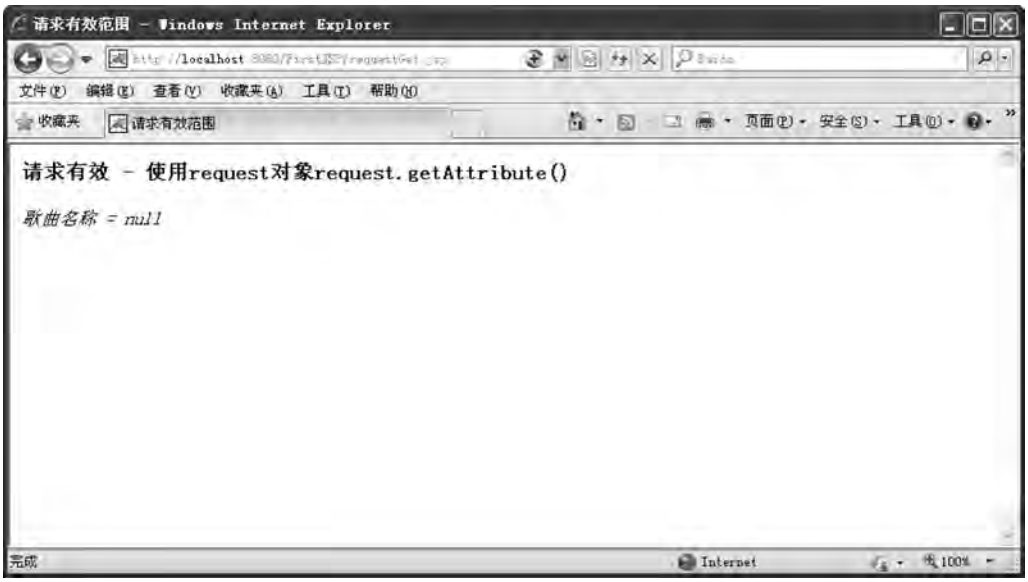

图 3.9　response 对象的应用

2. 数据共享不同

forward 转发页面和转发到的页面可以共享 request 中的数据，所以提取出来的歌曲名能正确显示，而 redirect 不能共享数据，提取出来的数据为 null。

3. 重定向范围不同

redirect 不仅可以重定向到当前应用程序的其他资源，还可以重定向到同一个站点上的其他应用程序中的资源，甚至是使用绝对 URL 重定向到其他站点的资源。forward 只能在同一个 Web 应用程序内的资源之间转发请求。

3.2.4　案例 4：使用 application 隐含对象

一个 Web 服务器通常有多个 Web 服务目录(网站)，当 Web 服务器启动时，它自动为每个 Web 服务目录都创建一个 application 对象，这些 application 对象各自独立，而且和 Web 服务目录一一对应。访问同一个网站的客户都共享一个 application 对象，因此，application 对象可以实现多客户间的数据共享。

一个 Web 应用程序启动后，将会自动创建一个 application 对象，而且在整个应用程序的运行过程中只有一个 application 对象，就是说所有访问该网站的客户都共享一个 application 对象。不管哪个客户来访问网站 A，也不管客户访问网站 A 下的哪个页面文件，都可以对网站 A 的 application 对象进行操作，因为所有访问网站 A 的客户都共用一个 application 对象。访问不同网站的客户，对应的 application 对象不同。

application 对象的基类是 javax.servlet.ServletContext 类，有些 Web 服务器不直接支持使用 application 对象，必须用 ServletContext 类(用于表示应用程序的上下文)来声明 application 对象，再调用 getServletContext()方法来获取当前页面的 application 对象。application 常用方法见表 3.5。

表 3.5　application 常用方法

方法名称	方法说明
String getAttribute(String name)	根据属性名称获取属性值
Enumeration getAttributeNames()	获取所有的属性名称
void setAttribute(String name, Object object)	设置属性,指定属性名称和属性值
void removeAttribute(String name)	根据属性名称删除对应的属性
ServletContext getContext(String uripath)	获取指定 URL 的 ServletContext 对象
String getContextPath()	获取当前 Web 应用程序的根目录
String getInitParameter(String name)	根据初始化参数名称,获取初始化参数值
int getMajorVersion()	获取 Servlet API 的主版本号
int getMinorVersion()	获取 Servlet API 的次版本号
String getMimeType(String file)	获取指定文件的 MIME 类型
String getServletInfo()	获取当前 Web 服务器的版本信息
String getServletContextName()	获取当前 Web 应用程序的名称
void log(String message)	将信息写入日志文件中

例 3.9　利用 application 对象设计一个所有用户对某网页的访问次数,并显示当前服务器的版本号。counter.jsp 代码如下:

```
<!-- counter.jsp -->
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    <%
    //判断 application 对象中有没有保存名为 count 的参数
    //如果没有,在 application 对象中新增一个名为 count 的参数
    if(application.getAttribute("count")==null){
        application.setAttribute("count", new Integer(0));
    }
    Integer count = (Integer)application.getAttribute("count");
    //使用 application 对象读取 count 参数的值,再在原值基础上累加 1
    application.setAttribute("count",new Integer(count.intValue()+1));
%>
    <h2>
        <!-- 输出累加后的 count 参数对应的值 -->
         欢迎您访问,本页面已经被访问过 <font color="#ff0000"><%=application.getAttribute("count")%></font>次.
    </h2>
 当前服务器的版本为 <%=application.getServerInfo()%>
</body>
</html>
```

运行界面如图 3.10 所示。

图 3.10　统计网页的访问次数

值得注意的是，本例采用的是 application 对象，但在服务器重启后计数将重新开始。

3.2.5　案例 5：使用 session 隐含对象

session 对象是 java.servlet.http.HttpSession 子类的对象，表示当前的用户会话信息。在 session 中保存的对象在当前用户连接的所有页面中都是可以被访问到的，在 3.1 中已经指出了这一特点。

当用户登录网站时，系统会自动分配给用户一个 session ID，用来标识不同的访问客户。JSP 容器会将这个 ID 号发送到客户端，保存在客户的 Cookie 中，这样 session 对象和客户之间就建立起一一对应的关系，即每个客户对应一个 session 对象。

1. Cookie 对象

Cookie 对象是 javax.servlet.http.Cookie 的实例，是由 Web 服务器端产生后被保存到浏览器中的信息。Cookie 对象可以用来保存一些信息在浏览器中，当浏览器请求服务器的页面时会自动发送到服务器端。目前主流的浏览器 IE、Netscape Navigator 都支持 Cookie。大多数浏览器允许用户禁止 Cookie 的使用，因此，如果应用中必须使用 Cookie 对象，一定要提示用户。

Cookie 对象的属性如表 3.6 所示。

表 3.6　Cookie 的属性

属　性　名	说　　明
name	Cookie 名称
value	Cookie 值
domain	只有在该域中的服务器才会发送该 Cookie，例如，www.bjtu.edu.cn
maxAge	Cookie 持续存在的时间，－1 表示一直有效
path	只有 URL 中包含指定的路径的服务器才发送该 Cookie，例如，helloBeijing
secure	是否仅当使用 https 协议才发送 Cookie。https 是 http 的加密形式

Cookie 对象的常用方法如表 3.7 所示。

表 3.7　Cookie 的方法

方法名称	方法说明
String getComment()	返回 Cookie 中注释，如果没有注释的话将返回空值
String getDomain()	返回 Cookie 中 Cookie 适用的域名，使用 getDomain()方法可以指示浏览器把 Cookie 返回给同一域内的其他服务器，而通常 Cookie 只返回给与发送它的服务器名字完全相同的服务器。注意域名必须以点开始(例如，yesky.com)
int getMaxAge()	返回 Cookie 过期之前的最大时间，以秒计算
String getName()	返回 Cookie 的名字
String getPath()	返回 Cookie 适用的路径。如果不指定路径，Cookie 将返回给当前页面所在目录及其子目录下的所有页面
boolean getSecure()	如果浏览器通过安全协议发送 Cookie 将返回 true 值，如果浏览器使用标准协议则返回 false 值
String getValue()	返回 Cookie 的值
int getVersion()	返回 Cookie 所遵从的协议版本
void setComment(String purpose)	设置 Cookie 中注释
void setDomain(String pattern)	设置 Cookie 中 Cookie 适用的域名
void setMaxAge(int expiry)	以秒计算，设置 Cookie 过期时间
void setPath(String uri)	指定 Cookie 适用的路径
void setSecure(boolean flag)	指出浏览器使用的安全协议，例如 HTTPS 或 SSL
void setValue(String newValue)	Cookie 创建后设置一个新的值
void setVersion(int v)	设置 Cookie 所遵从的协议版本

例 3.10　简单地写入和读出 Cookie。

写入 Cookie 的 writeCookie.jsp 代码如下：

```
<!-- writeCookie.jsp -->
<%@ page language="java" import="java.util.*" pageEncoding="GB2312"%>
<%
String str1="hello";
Cookie c=new Cookie("str2",str1);
response.addCookie(c);
 %>
正在将<%=str1%>写入 Cookie...<br>
<jsp:include page="readCookie.jsp"/>
```

读出 cookie 的 readCookie.jsp 代码如下：

```
<!-- readCookie.jsp -->
<%@ page language="java" import="java.util.*" pageEncoding="GB2312"%>
<br>
```

读出 Cookie 的值：

```
<%
Cookie cookies[]=request.getCookies();
for(int i=0;i<cookies.length;i++){
  if(cookies[i].getName().equals("str2"))
```

```
        out.print(cookies[i].getValue());
        }
%>
```


运行 writeCookie.jsp 的结果如图 3.11 所示。

图 3.11　读写 Cookie 对象

2. Session 对象

Session 的常用方法如表 3.8 所示。

表 3.8　Session 的常用方法

方法名称	方法说明
void setAttribute(String attribute, Object value)	设置 Session 属性。value 参数可以为任何 Java Object。通常为 Java Bean。value 信息不宜过大
String getAttribute(String attribute)	返回 Session 属性
Enumeration getAttributeNames()	返回 Session 中存在的属性名
void removeAttribute(String attribute)	移除 Session 属性
String getId()	返回 Session 的 ID。该 ID 由服务器自动创建，不会重复
long getCreationTime()	返回 Session 的创建日期。返回类型为 long，常被转化为 Date 类型，例如，Date createTime = new Date(session.getCreationTime())
long getLastAccessedTime()	返回 Session 的最后活跃时间。返回类型为 long
int getMaxInactiveInterval()	返回 Session 的超时时间。单位为秒。超过该时间没有访问，服务器认为该 Session 失效
void setMaxInactiveInterval(int second)	设置 Session 的超时时间。单位为秒

例 3.11　该案例包含三个文件，一个登录界面 login.jsp，一个判断页面 select.jsp，如果输入用户名为“susan”，密码为“1234”时，跳转到 ok.jsp 页面，若成功跳转将会显示 session 的 ID 及用户名，如果用户名或密码错误直接给出提示。

login.jsp 的代码如下：

```
<!-- login.jsp -->
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
```

```
 <head>
    <title>在线音乐_用户登录</title>
</head>
 <body>
<jsp:include page="top.jsp"/>
<div id="navigation" style="font-size:12px; color:#000000">→ 欢迎您进入甜橙音乐网登录页面,请您填写正确的用户名与密码进行后台登录!    祝您天天有份好心情!</div>
<div id="main" style="padding-top:5px;margin-bottom:5px;" class="tableBorder_blue">
  <form name="form1" method="post" action="select.jsp" onSubmit="return check()">
    <table width="200" border="0" cellspacing="0" cellpadding="0">
      <tr>
        <td height="69" colspan="2" align="center"> </td>
      </tr>
      <tr>
        <td width="80" height="30" class="word_gray1">用户名:</td>
        <td width="120"><input name="userName" type="text" id="user" size="16"></td>
      </tr>
      <tr>
        <td height="30" class="word_gray1">密    码:</td>
        <td><input name="pwd" type="password" id="pwd" size="16"></td>
      </tr>
      <tr>
         <td height="30" colspan="2" align="center"><input name="Submit2" type="submit" class="btn_green" value="确定">
         
        <input name="Submit3" type="reset" class="btn_green" value="重置"></td>
      </tr>
    </table>
  </form>
</div>
</body>
</html>
```

select.jsp 的代码如下：

```
<!-- select.jsp -->
<%@ page contentType="text/html;charset=GBK" %>
<html>
<body>
<%
String name= new String(request.getParameter("userName").getBytes("ISO-8859-1"));
String pwd= new String(request.getParameter("pwd").getBytes("ISO-8859-1"));
if(("susan".equals(name))&&("1234".equals(pwd)))
{
session.setAttribute("userName", name);
%>
<jsp:forward page="ok.jsp"/>
<%
}
```

```
else
out.print("用户名或密码错误");
%>
</body>
</html>
```

ok.jsp 的代码如下：

```
<!-- ok.jsp -->
<%@ page contentType="text/html;charset=GBK" %>
<body>
<%
String name=(String)session.getAttribute("userName");
String ID=session.getId();
%>
<table>
<tr>
<td><h2>服务器为您分配的账号是:<%=ID%></h2></td>
</tr>
<tr>
<td><h2>欢迎您<%=name%></h2></td>
</tr>
</table>
```

运行结果如图 3.12～图 3.14 所示。

图 3.12　登录界面

图 3.13　用户名或密码错误界面

图 3.14　登录成功界面

3.2.6　案例 6：使用 config 隐含对象

config 对象实现于 javax. servlet. ServletConfig 接口，Web 容器在初始化时使用一个 ServletConfig(即 config)对象向 JSP 页面传递信息，此配置信息包括初始化参数(在当前 Web 应用的应用部署描述文件 web. xml 中定义)以及表示 Servlet 或 JSP 页面所属 Web 应用的 ServletContext 对象。

config 对象的方法如表 3.9 所示。

表 3.9　config 对象的方法

方法名称	方法说明
public String getInitParameter(name)	获取指定名字的初始参数
public java. util. Enumeration getInitParameterNames( )	获取所有初始参数
public ServletContext getServletContext( )	获取 ServletContext 对象(application)
public Sring getServletName( )	获取 Servlet 名字

如果在 web. xml 文件中，针对某个 servlet 文件或 JSP 文件设置了初始化参数，则可以通过 config 对象来获取这些初始化参数。

例 3.12 利用 config 对象获取 web.xml 的初始化参数。

```
<!-- config.jsp -->
<%@ page contentType="text/html;charset=GBK" import="java.util.*" %>
<html>
<head>
<title>config 对象</title>
</head>
<body>
<b>web 应用初始参数:</b>
<pre>
<%
Enumeration e = config.getInitParameterNames();
while (e.hasMoreElements())
          {
             Object nameOb = e.nextElement();
             String name = (String) nameOb;
             out.print(name + ": ");
             out.println(config.getInitParameter(name));
          }
%>
<b>web 应用名称:</b>
<%= config.getServletName() %><br/>
</pre>
</body>
</html>
```

没有对 web.xml 进行初始化设置的情况下,运行结果如图 3.15 所示。

图 3.15 获取 web.xml 初始化参数

3.2.7 案例 7:使用 exception 隐含对象

exception 对象的基类是 javax.servlet.jsp.JspException 类。当 JSP 页面在执行过程中发生异常或错误时,会自动产生一个 exception 对象。在 JSP 页面中,使用 page 指令,设置 isErrorPage 属性值为 true 后,就可以使用该 exception 对象,来查找页面出错信息。

Exception 对象的使用包括两部分：确定可能出现异常的页面和专门处理异常的页面。

1. 可能出现异常的页面

在有可能产生异常或错误的 JSP 页面中，使用 page 指令设置 errorPage 属性，属性值为能够进行异常处理的某个 JSP 页面。

2. 专门处理异常的页面

在专门负责处理异常的 JSP 页面中，使用 page 指令设置 isErrorPage 属性为 true，并使用 exception 对象来获取出错信息。

exception 对象的常用方法如表 3.10 所示。

表 3.10　exception 的常用方法

方法名称	方法说明
String getMessage()	返回简短的 JSP 页面的出错信息
String toString()	返回详细的 JSP 页面的出错信息
String getLocalizedMessage()	输出错误信息
void printStackTrace()	显示异常的栈跟踪轨迹
nativeThrowable fillInStackTrace()	重写异常错误的栈执行轨迹

例 3.13　创建一个出现异常的页面 exception.jsp 和一个专门处理异常的页面 error.jsp。代码如下：

```
<!-- exception.jsp -->
<%@ page isErrorPage = "true" errorPage = "error.jsp" contentType = "text/html;charset = GBK"
%>
<h2>发生异常</h2>
<%! boolean throwError = true; %>
<%
  if (throwError){
      throwError = false;
      throw new Exception("抛出异常");
      }
%>
<i>错误描述:<i><% = exception.getMessage() %><br/>
<% = exception.toString() %><p/>
<i>详细出错原因:</i><p/>
<pre>
<% exception.printStackTrace(new java.io.PrintWriter(out));
   throwError = true;
   %>
</pre>
<!-- error.jsp -->
<%@ page contentType = "text/html; charset = UTF-8" language = "java" import = "java.sql.*"
errorPage = "" %>
<html>
<head>
<title>错误提示</title>
<link href = "CSS/style.css" rel = "stylesheet">
</head>
```

```
<body>
<table width="100%" height="100%" border="0" cellpadding="0" cellspacing="0">
  <tr>
    <td align="center"><table width="419" height="226" border="0" cellpadding="0"
cellspacing="0">
      <tr>
        <td align="center">  错误提示信息 ${error}<br>
          <br>
          <input name="Submit" type="submit" class="btn_grey" value="返回" onClick=
"history.back(-1)"></td>
      </tr>
    </table></td>
  </tr>
</table>
<center>
</center>
</body>
</html>
```

运行 exception.jsp 的结果如图 3.16 所示。

图 3.16　exception 对象的应用

3.2.8　案例 8：使用 page 隐含对象

page 是 java.lang.Object 类的对象。page 对象在当前 JSP 页面中可以用 this 关键字替代。在 JSP 页面的 Java 程序脚本中和 JSP 表达式中都可以使用 page 对象。

page 对象常见的方法如表 3.11 所示。

表 3.11　page 的常用方法

方法名称	方法说明
class getClass()	返回当时 Object 的类
int hashCode()	返回此时 Object 的哈希代码
String toString()	将此时的 Object 类转换成字符串
boolean equals(Object ob)	比较此对象是否与指定的对象相等

续表

方法名称	方法说明
void copy(Object ob)	将此对象复制到指定的对象中
Object clone()	对此对象进行克隆
ServletConfig getServletConfig()	获取 ServletConfig 对象
ServletContext getServletContext()	获取 ServletContext 对象
String getServletInfo()	获取当前 JSP 页面的 Info 属性

例 3.14 使用 page 指令定义属性 info，然后使用 getServletInfo()方法取得属性 info 的值，page.jsp 代码如下：

```
<!-- page.jsp -->
<%@ page info="音乐之声 可以试听下载。" contentType="text/html;charset=GBK" %>
<html>
<head>
  <title>page 对象</title>
</head>
<body>
<b>页面指令中定义的 Info 属性 :</b>
<% =this.getServletInfo() %>
</body>
</html>
```

运行结果如图 3.17 所示。

图 3.17 page 对象的应用

3.2.9 案例 9：使用 pageContext 隐含对象

pageContext 对象是 javax.servlet.jsp.PageContext 类的实例对象，它是一个比较特殊的对象，它相当于页面中所有其他对象功能的最大集成者，即使用它可以访问到本页面中所有其他的对象，例如前面描述过的 request、response、out 和 page 对象等。由于在 JSP 中 request 和 response 等对象本来就可以通过直接调用方法使用，所以 pageContext 对象在实际 JSP 开发中很少使用到。

pageContext 对象在存取其他隐含对象属性的 setAttribute 方法中需要指定范围的参数，其语法格式为：

```
void setAttribute(String name, Object value, int scope)
```

语法说明：

范围参数 scope 有 4 个，分别代表 4 种范围：PAGE_SCOPE、REQUEST_SCOPE、SESSION_SCOPE、APPLICATION_SCOPE。

pageContext 对象的常用方法如表 3.12 所示。

表 3.12　pageContext 常用方法

方法名称	方法说明
Exception getException( )	回传网页的异常
JspWriter getOut( )	回传网页的输出流，例如，out
Object getPage( )	回传网页的 Servlet 实体(instance)，例如，page
ServletRequest getRequest( )	回传网页的请求，例如，request
ServletResponse getResponse( )	回传网页的响应，例如，response
ServletConfig getServletConfig( )	回传此网页的 ServletConfig 对象，例如，config
ServletContext getServletContext( )	回传此网页的执行环境(context)，例如，application
HttpSession getSession( )	回传和网页有联系的会话(session)，例如，session
Object getAttribute(String name, int scope)	回传 name 属性，范围为 scope 的属性对象，回传类型为 Object
Enumeration getAttributeNamesInScope(int scope)	回传所有属性范围为 scope 的属性名称，回传类型为 Enumeration
int getAttributesScope(String name)	回传属性名称为 name 的属性范围
void removeAttribute(String name)	移除属性名称为 name 的属性对象
void removeAttribute(String name, int scope)	移除属性名称为 name，范围为 scope 的属性对象
void setAttribute(String name, Object value, int scope)	指定属性对象的名称为 name、值为 value、范围为 scope
Object findAttribute(String name)	寻找在所有范围中属性名称为 name 的属性对象

例 3.15　利用 pageContext 对象设置 4 种范围的属性值并进行获取。利用 pageContext 对象的常用方法进行删除、查找属性等简单操作。pageContext.jsp 代码如下：

```
<!-- pageContext.jsp -->
<%@page contentType="text/html;charset=gb2312"%>
<html>
<head>
<title>pageContext 对象_例 1</title>
</head>
<body>
<h2>pageContext 对象</h2>
<%
pageContext.setAttribute("name","爸爸去哪儿");
request.setAttribute("name","下载歌曲");
session.setAttribute("name","甜橙音乐之声");
application.setAttribute("name","音乐之声");
%>
page 设定的值:<%=pageContext.getAttribute("name")%><br>
request 设定的值:<%=pageContext.getRequest().getAttribute("name")%><br>
session 设定的值:<%=pageContext.getSession().getAttribute("name")%><br>
```

```
application 设定的值:<% = pageContext.getServletContext().getAttribute("name") %><br>
<br>
页面有效的值:<% = pageContext.getAttribute("name",1) %><br>
请求有效的值:<% = pageContext.getAttribute("name",2) %><br>
会话有效的值:<% = pageContext.getAttribute("name",3) %><br>
应用有效的值:<% = pageContext.getAttribute("name",4) %><br>
<br>
<i>现在利用 removeAttribute 方法删除 session 设置。</i><br>
<% pageContext.removeAttribute("name",3); %>
<br>
pageContext 修改后的 session 设定的值:<% = session.getValue("name") %><br>
<br>
<i>现在修改 application 的值。</i><br>
<% pageContext.setAttribute("name","音乐网",4); %>
<br>
pageContext 修改后的 application 设定的值:<% = pageContext.getServletContext().getAttribute
("name") %><br>
<br>
默认属性的值:<% = pageContext.findAttribute("name") %><br>
默认属性的范围值:<% = pageContext.getAttributesScope("name") %><br>
</body>
</html>
```

运行界面如图 3.18 所示。

图 3.18　pageContext 对象的应用

3.3 小　　结

JSP 隐含对象包括 9 种，本章主要对这 9 种对象进行了详细的介绍。从使用和开发的角度介绍了 9 种隐含对象各自的由来、特点以及常用方法，对每个隐含对象都有案例进行相应的解释。

9 种隐含对象中涉及了 4 种有效范围，重点学习 4 种有效范围的特点。

学习本章时，应把注意力放在掌握各种隐含对象的使用上，为以后的编程打下坚实的基础。由于对象比较多，方法又很丰富，因此用了比较简单的例子进行说明。

3.4 习　　题

一、选择题

1. 下面不属于 JSP 内置对象的是(　　)。

A. out 对象　　B. respone 对象　　C. application 对象　　D. page 对象

2. 以下哪个对象提供了访问和放置页面中共享数据的方式？(　　)

A. pageContext　　B. response　　C. request　　D. session

3. 在 JSP 中为内建对象定义了 4 种作用范围，即 Application Scope、Session Scope、Page Scope 和(　　)4 个作用范围。

A. Request Scope　　B. Response Scope

C. Out Scope　　D. Writer Scope

4. Form 表单的 method 属性能取下列哪项的值？(　　)

A. submit　　B. puts　　C. post　　D. out

5. 可以利用 JSP 动态改变客户端的响应，使用的语法是(　　)。

A. response. setHeader()　　B. response. outHeader()

C. response. writeHeader()　　D response. handlerHeader()

6. JSP 页面中 request. getParamter(String)得到的数据，其类型是(　　)。

A. Double　　B. int　　C. String　　D. Integer

7. 当利用 request 的方法获取 Form 中元素时，默认情况下字符编码是(　　)。

A. ISO-8859-1　　B. GB2312　　C. GB3000　　D. ISO-82591

二、填空题

1. JSP 的(　　)对象用来保存单个用户访问时的一些信息。

2. response 对象的(　　)方法可以将当前客户端的请求转到其他页面去。

3. 当客户端请求一个 JSP 页面时，JSP 容器会将请求信息包装在(　　)对象中。

4. 表单标记中的(　　)属性用于指定处理表单数据程序 url 的地址。

5. < select >标记中的 size 属性默认值为(　　)。

三、简答题

1. 请说出 JSP 中常用的隐含对象。

2. 简述 request 对象和 response 对象的作用。

3. session 对象与 application 对象有何区别?
4. 网页中的表单如何定义? 通常表单中包含哪些元素?
5. JSP 隐含对象有哪 4 个作用范围? 什么情况下 session 会关闭?
6. response.sendRedirect(URL url)方法有何作用?
7. 是不是所有 Web 服务目录共用一个 application?
8. 怎样使用 request、session 和 application 对象进行参数存取?
9. 设计注册表单,利用 request 对象实现获取用户注册信息。
10. 利用 session 对象实现购物车。