第 3 章　内置对象技术

JSP 页面或 Servlet 页面的内置对象由容器(服务器)提供，可以使用标准的变量访问这些对象，并且不用编写任何额外的代码，在 JSP 页面或 Servlet 页面中使用。在 JSP 2.0 规范中定义了以下 9 个内置对象：request(请求对象)、response(响应对象)、session(会话对象)、application(应用程序对象)、out(输出对象)、page(页面对象)、config(配置对象)、exception(异常对象)、pageContext(页面上下文对象)。在本章中，将对它们进行介绍，并通过示例介绍它们的具体使用方法。

3.1　内置对象概述

Web 应用程序的特点是一个 JSP 文件(或者一个 Servlet)相当于一个独立的运行单元，类似于一个独立的应用程序，并由容器(Tomcat)统一管理。对于一个实际项目来说，不可能只有一张页面，且页面之间存在着各类内部数据的实时通信及共享问题，例如，把 A 页面登录数据传递到 B 页面进行验证，购物车的设计涉及若干页面共享数据问题，公告栏涉及不同用户的数据共享问题。而且，在实际项目中，存在着对各类请求/响应的一些特殊要求等。因此，容器根据规范要求，向用户提供了一些内置对象，用于解决上述问题，并负责对这些对象的管理，包括内置对象的生存期、作用域等。

在这些内置对象中，request、response 对象是在客户端请求 JSP 页面或 Servlet 页面时，由容器实时生成并作为服务参数传递给 JSP 文件(实际上是 Servlet)，在请求/响应过程结束时由容器回收；session 一般是在用户开始登录系统时生成的，在退出系统时由容器回收。

3.2 request 对象

request 对象最主要的作用在当次请求中进行数据传递，当请求发起方(JSP 页面或 Servlet 页面，甚至是 HTML 页面)向另一方(JSP 页面或 Servlet 页面)发起请求时，容器(服务器)会将客户端的请求信息包装在这个 request 对象中，请求信息的内容包括请求的头信息、请求的方式、请求的参数名称和参数值等信息。request 对象封装了用户提交的信息，通过调用该对象相应的方法可以获取来自客户端的请求信息，然后根据不同需求做出响应。它是 HttpServletRequest 类的实例。

3.2.1 主要方法

request 对象的主要方法如表 3-1 所示。

表 3-1 request 对象的主要方法

方 法 名	方 法 说 明
getAttribute(String name)	返回指定属性的值
getAttributeNames()	返回所有可用属性名
getCharacterEncoding()	返回字符编码方式
getContentLength()	返回请求体的长度(以字节为单位)
getContentType()	得到请求体的 MIME 类型
getInputStream()	得到请求体中的二进制流
getParameter(String name)	返回指定参数的值
getParameterNames()	返回所有可用参数名
getParameterValues(String name)	返回包含指定参数的所有值的数组
getProtocol()	返回请求方使用的协议类型及版本号
getServerName()	返回接收请求的服务器主机名
getServerPort()	返回服务器接收请求所用的端口号
getReader()	返回解码后的请求体
getRemoteAddr()	返回发送请求的客户端 IP 地址
getRemoteHost()	返回发送请求的客户端主机名
setAttribute(String key,Object obj)	设置指定属性的值
getRealPath(String path)	返回指定虚拟路径的真实路径

续表

方 法 名	方 法 说 明
getMethod()	返回客户端向服务器传输数据的方式
getRequestURL()	返回发出请求字符串的客户端地址
getSession()	创建一个 session 对象

下面的程序给出了 request 对象的常用方法示例,通常使用 request 对象获得客户端传来的数据。

Example3_1.jsp 代码如下:

```
<%@ page contentType="text/html;charset=utf-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>requestHTML.html</title>
</head>
<body>
    <form action="RequestHTML" method="post">
        用户名:
        <input type="text" name="name"><br>
           密 码:
        <input type="text" name="pass"><br>
          
        <input type="submit" value="提交">
    </form>
    <br>
</body>
</html>
```

RequestHTML.java 代码如下:

```
public class RequestHTML extends HttpServlet {
    …
}
public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    //请求方式
    System.out.println(request.getMethod());
    //请求的资源
    System.out.println(request.getRequestURI());
    //请求用的协议
```

```
        System.out.println(request.getProtocol());
        //请求的文件名
        System.out.println(request.getServletPath());
        //服务器主机名
        System.out.println(request.getServerName());
        //服务器的端口号
        System.out.println(request.getServerPort());
        //客户端 IP 地址
        System.out.println(request.getRemoteAddr());
        //客户端主机名
        System.out.println(request.getRemoteHost());
        String user=request.getParameter("name");
        if(user==null)user="无输入";
        String pass=request.getParameter("pass");
        if(pass==null)user="无输入";
        System.out.println("user="+user+"pass="+pass);
    }
```

如果输入 123 和 123,输出如下:

```
POST
/JSP1501/RequestHTML
HTTP/1.1
/RequestHTML
localhost
8080
127.0.0.1
127.0.0.1
user=123 pass=123
```

3.2.2 常用技术

1. 用 getParameter()方法获取表单提交的信息

request 对象获取客户端提交的信息的常用方法是 getParameter(String key),其中 key 与 JSP(或 HTML)页面中表单的各输入域(如 text、checkbox 等)的 name 属性一致。在下面的示例中,form1.jsp 页面通过表单向 servlet(requestForm1)提交用户名和密码信息,requestForm1 通过 request 对象获取表单提交的信息。

form1.jsp 示例代码:

```
<%@ page contentType="text/html;charset=utf-8" %>
<html>
    <body>
        <form action="requestForm1" method="post">
```

```
        <P>姓名:<input type="text" size="20" name="UserID"></P>
        <P>密码:<input type="password" size="20" name="UserPWD"></P>
        <P><input type="submit" value="提 交"> </P>
        </form>
    </body>
</html>
```

表单提交的方法主要有两种：get 与 post，二者的主要区别是前一种方法会在提交过程中在地址栏中显示提交的信息。

requestForm1 核心代码：

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html");
    String name=request.getParameter("UserID");
    String pass=request.getParameter("UserPWD");
    System.out.println(name);
    System.out.println(pass);
}
```

中文显示问题包括两方面：其一是页面在浏览器中的中文显示问题，JSP 文件的默认编码为 ISO-8859-1，应改为 uft-8；其二是中文字符在不同环境（HTML、JSP、Servlet、数据库）中的传输问题，如从 JSP 传输到 Servlet，因为不同环境下默认编码不一样，会产生中文乱码问题。解决办法如下：

（1）服务器端重新编码技术：

```
String user=request.getParameter("UserID ");
if(user==null) user="无输入";
byte b[]=user.getBytes("ISO-8859-1");
user=new String(b);
```

（2）用过滤器技术。5.2 节中会有说明。

（3）随着集成开发环境版本的升级，中文问题可以通过平台的设置来解决。以 Eclipse 为例，可以通过 Window→Properties→General→Workspace→Text file encoding 设置项目空间的字符编码方式，一般设为 UTF-8，如图 3-1 所示。

2. 用 getParameterValues()方法获取表单成组信息

通过 request 对象的 getParameterValues()方法可以获得指定参数的成组信息，通常在表单的复选框中使用。该方法的原型如下：

```
public String[] getParameterValues(String str)
```

在下面的示例中，form2.jsp 表单中有 3 个复选框。选择复选框后，表单信息提交给 servlet(requestForm2.class)。在 Servlet 中使用 getParameterValues()方法获取复选框的

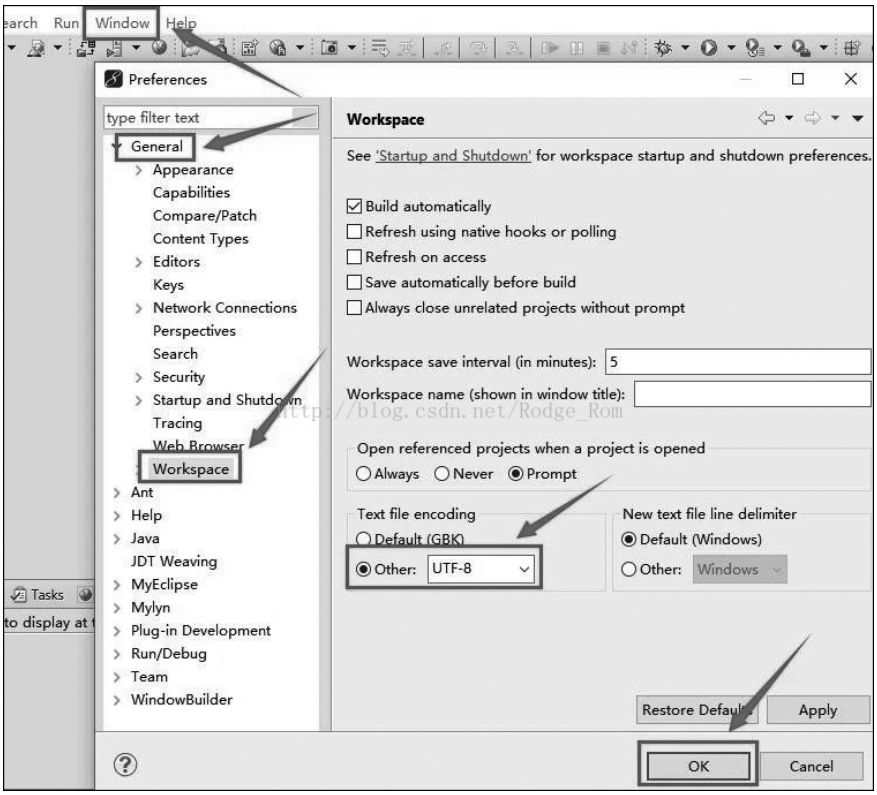

图 3-1　设置中文编码

成组信息并输出。

form2.jsp 代码如下：

```
<%@ page contentType="text/html;charset=utf-8" %>
<html>
<body>
<form id="form1" name="form1" method="post" action="requestForm2">
    请选择喜欢的水果:<p>
    <input type="checkbox" name="checkbox" value="apple" />apple
    <input type="checkbox" name="checkbox" value="banana" />banana
    <input type="checkbox" name="checkbox" value="peach" />peach<p>
    <input type="submit" name="Submit" value="提交" />
</form>
</body>
</html>
```

requestForm2.java 核心代码：

```
String[] temp=request.getParameterValues("checkbox");
System.out.println("你喜欢的水果是:");
```

```
for (int i=0; i<temp.length; i++){
    System .out.println(temp[i]+" ");
```

在实际的项目开发中,目前很少采用这种技术,一般用 JSON 数据方式。

3. getAttribute()及 request.setAttribute()方法的应用

如上所述,request 主要用于当次请求中的数据传递(两个 JSP 页面或 Servlet 页面之间)。getParameter()和 getParameterValues()方法用于后端(Servlet 或 JSP)获取前端的各类表单信息。如果后端向前端发回数据(也是一次请求过程),则需要用到 request.setAttribute()方法,前端接收数据则用 getAttribute()方法。

例如,传递过程为 login.jsp(前端)→LoginServlet(后端)→login.jsp(前端),则对于 LoginServlet(后端),可用以下代码实现数据回传。

(1) 在 doGet()方法中,代码如下:

```
RequestDispatcher dispatcher=null;
if(!LoginManagement.login(account, secret))              //登录失败
    request.setAttribute("log", "error");                //request 中写数据
dispatcher=getServletContext().getRequestDispatcher("/login.jsp");
dispatcher.forward(request, response);                   //向前端发数据
```

(2) 在前端 login.jsp 中,代码如下:

```
<% if(request.getAttribute("log")!=null){
    String str=(String)request.getAttribute("log");
    if(str.equals("error"))
        out.println("<br>用户名或者密码错误");
} %>
```

另外,利用 request 可以传递任意类型对象数据。

有时,项目要求传递其他类型值。例如,在一个 Servlet 中,通过数据库操作,得到一个学生记录集,以二维数组形式存放,具体可以用 ArrayList 实现。进一步把该记录集发回 JSP 页面,以显示该记录集,此时就可以利用 request 的 setAttribute(String key,Object obj)和 getAttribute(String key)来设值和取值了。代码如下:

(1) Servlet 中的代码如下:

```
ArrayList studentList;                                   //其具体值的获得代码略
request.setAttribute("student",studentList );
…                                                        //提交到 A.jsp
```

(2) A.jsp 中的代码如下:

```
ArrayList list=(ArrayList)request. getAttribute("student");
```

在实际的项目开发中,目前很少采用这种技术,一般用 JSON 数据方式。

4. 文件处理

文件上传和下载是 Web 项目的常用功能。默认情况下，前端通过网络传输的是符合 HTTP 要求的文本串（HTML 表单的所有内容，包括表单的数据），此时，enctype 的值为 application/x-www-form-urlencoded。服务器通过解析，让 request 的各种方法获取文本串数据。而文件上传则需要二进制流数据，这需要改变报文格式，enctype＝multipart/form-data，表单才会把文件的内容编码到 HTML 请求中；在服务器端，用 request. getInputStream()方法获得二进制流数据，具体实现会在以后章节说明。

3.2.3 作用域与生命周期

只要发出一个请求，服务器就会创建一个 request 对象，它的作用域是当前请求过程。一般情况下，一个请求过程包括两个对象（发起方与接受方），可以是两个 JSP 页面之间、两个 Servlet 页面之间，当然也可以是 JSP 页面与 Servlet 页面之间。其生命周期也是一个请求过程。

综上所述，request 就是两个服务器对象之间的数据传递工具。

3.3 response 对象

request 对象和 response 对象是相辅相成的，request 对象用于封装客户端的请求报文信息。response 对象用于处理服务器对客户端的一些响应。response 对象可以对客户端做出动态响应，主要是向客户端发送头部数据。它是 HttpServletResponse 类的实例。

3.3.1 主要方法

response 对象的主要方法如表 3-2 所示。

表 3-2 response 对象的主要方法

方 法 名	方 法 说 明
addCoolie(Cookie cookie)	向客户端写入一个 Cookie
addHeader(String name,String value)	添加 HTTP 头
containsHeader(String name)	判断指定的 HTTP 头是否存在
encodeURL(String url)	把 SessionID 作为 URL 参数返回客户端
getOutputStream()	获得到客户端的输出流对象
getWriter()	返回输出字符流

续表

方 法 名	方 法 说 明
sendError(int)	向客户端发送错误信息
sendRedirect(String url)	重定向请求
setContentType(String type)	设置响应的 MIME 类型
setHeader(String name, String value)	设置指定的元信息值。如果该值已经存在,则新值会覆盖旧值

3.3.2 常用技术

1. 使用 response 对象设置 HTTP 头信息

这里主要介绍两个方法:setContentType()和 setHeader()。

setContentType()方法可以动态改变 ContentType 的属性值,参数可设为 text/html、text/plain、application/x-msexcel、application/msworld 等。该方法的作用是:客户端浏览器区分不同种类的数据,并调用浏览器内不同的程序嵌入模块来处理相应的数据。例如,Web 浏览器就是通过 MIME 类型来判断文件是否为 GIF 图片的。通过 MIME 类型来处理 JSON 字符串。Tomcat 的安装目录\conf\web.xml 中定义了大量 MIME 类型,读者可以自行了解。

response.setHeader 用来设置返回页面的元(meta)信息。元信息用来在 HTML 文档中模拟 HTTP 的响应头报文。<meta>标签用于网页的<head>与</head>中,例如:

<meta name="Generator" contect="">用于说明页面生成工具。

<meta name="Keywords" contect="">用于说明页面的关键词。

<meta name="Description" contect="">用于说明网站的主要内容。

<meta name="Author" contect="你的姓名">用于说明网站的制作者。

例如,利用 response 对象将 contentType 属性值设置为 application/x-msexcel。

(1) A.txt 内容如下:

```
34   79   51   99<br>
40   69   92   22<br>
67   71   85   20<br>
72   30   78   38<br>
55   61   39   43<br>
43   81   10   55<br>
36   93   41   99<br>
```

(2) contenttype.html 代码如下:

```
<HTML>
```

```
<BODY bgcolor=cyan><Font size=5>
  <P>您想使用什么方式查看文本文件 A.txt?
    <FORM action="response1.jsp" method="post" name=form>
    <INPUT TYPE="submit" value="word" name="submit1">
    <INPUT TYPE="submit" value="excel" name="submit2">
    </FORM>
</FONT>
</BODY>
</HTML>
```

(3) response1.jsp 代码如下：

```
<%@ page contentType="text/html;charset=gb2312"%>
<HTML>
<BODY>
    <%
        String str1=request.getParameter("submit1");
        String str2=request.getParameter("submit2");
        if (str1==null) {
            str1="";
        }
        if (str2==null) {
            str2="";
        }
        if (str1.startsWith("word")) {
            response.setContentType("application/msword;charset=GB2312");
            out.print(str1);
        }
        if (str2.startsWith("excel")) {
            response.setContentType("application/x-msexcel;charset=GB2312");
        }
    %>
    <jsp:include page="A.txt"/>
</BODY>
</HTML>
```

2. 使用 response 实现重定向

对于 response 对象的 sendRedirect()方法，可以将当前客户端的请求转到其他页面，相应的代码格式为

```
response.sendRedirect("URL 地址");
```

下面的示例中，login.html 提交姓名到 response3.jsp 页面。如果提交的姓名为空，需要重定向到 login.html 页面；否则显示欢迎页面。

login.html 代码如下：

```
<HTML>
    <BODY>
        <FORM ACTION="response3.jsp" METHOD="POST">
        <P>姓名:<INPUT TYPE="TEXT" SIZE="20" NAME="UserID"></P>
        <P><INPUT TYPE="SUBMIT" VALUE="提 交"> </P>
        </FORM>
    </BODY>
</HTML>
```

response3.jsp 代码如下：

```
<%@ page contentType="text/html;charset=GB2312" %>
<HTML>
<BODY>
    <%
        String s=request.getParameter("UserID ");
        byte b[]=s.getBytes("ISO-8859-1");
        s=new String(b);
        if (s==null) {s="" ; response.sendRedirect("login.html");}
        else out.println("欢迎您来到本网页!"+s);
    %>
</BODY>
</HTML>
```

注意，用 dispatcher.forward(request, response)方法和 response 对象中的 sendRedirect()方法都可以实现页面的重定向，但二者是有区别的。前者只能在本网站内跳转，且跳转后，在地址栏中仍然显示以前页面的 URL，跳转前后的两个页面属同一个 request 对象，用户程序可以用 request 对象设置或传递用户程序数据。response.sendRedirect 则不一样，它相对前者是绝对跳转，在地址栏中显示的是跳转后页面的 URL，跳转前后的两个页面不属于同一个 request 对象。当然也可以用其他技术手段来保证 request 对象为同一个，但这不在本节的讨论范围内。对于后者来说，可以跳转到任何一个地址的页面。例如：

```
response.sendRedirect ("http: //www.baidu. com/")
```

3. 利用 response 的 body 数据区

在 Web 服务的异步处理过程中，服务器一般采用向 response 的 body 数据区写数据(一般是普通文本串或格式化文本串，如 JSON 文本串)的方式实现与客户端的异步通信。当然，写数据的工具是另一个内置对象 out。关于异步处理，会在以后章节介绍。

3.4 session 对象

3.4.1 基本概念和主要方法

session 是会话的意思，指的是客户端与服务器的一次会话过程，以便跟踪每个用户的操作状态。一般情况下，session 对象在第一个 JSP 页面或 Servlet 被装载时由服务器自动创建，并在用户退出应用程序时由服务器销毁，完成会话期管理，这也是 session 对象的生命周期。服务器为每个访问者都设立一个独立的 session 对象，用于存储 session 变量，并且各个访问者的 session 对象互不干扰。

session 对象是 HttpSession 类的实例。session 机制是一种服务器端的机制，服务器使用类似于散列表的结构(也可能就是散列表)来管理客户端信息，因此在实际项目中，应该注意慎用 session 对象，以免服务器内存溢出。服务器为每个客户新建一个 session 对象时产生一个唯一的 sessionID 与 session 相关联，这个 sessionID 的值是一个既不会重复又不容易找到规律以伪造的字符串，而且它保存在客户端的 Cookie 中。如果客户端不支持 Cookie，那么就不能使用 session。可以通过重写 URL 等技术来保证 sessionID 的唯一性。

在实际的项目中，session 对象往往作为一次会话期内共享数据的容器，用户程序可以把最能标识用户的信息(如用户名、密码及权限等)存放在 session 对象中，以便对用户进行管理。表 3-3 为 session 对象的主要方法。

表 3-3　session 对象的主要方法

方　法　名	方 法 说 明
getAttribute(String name)	获取与指定名称关联的 session 属性值
getAttributeNames()	取得 session 内所有属性的集合
getCreationTime()	返回 session 创建时间，单位为千分之一秒
getId()	返回 session 创建时 JSP 引擎为它设置的唯一的 sessionID
getLastAccessedTime()	返回当前 session 中客户端最后一次访问时间
getMaxInactiveInterval()	返回两次请求间隔时间，以秒为单位
getValueNames()	返回一个包含此 session 中所有可用属性的数组
invalidate()	取消 session，使 session 不可用
isNew()	判断是否为新创建的 session
removeValue(String name)	删除 session 中指定的属性
setAttribute(String name, Object value)	设置指定名称的 session 属性值
setMaxInactiveInterval()	设置两次请求间隔时间，以秒为单位

下面的程序用到了 session 的一些常用方法,代码如下:

```
<%@ page contentType="text/html;charset=gb2312"%>
<%@ page import="java.util.*;"%>
<html>
<head>
    <title>session 对象示例</title>
</head>
<body>
    <br>
    session 的创建时间:<%=session.getCreationTime()%>  
    <!--返回从 1970 年 1 月 1 日 0 时起到计算时的毫秒数-->
    <%=new java.sql.Time(session.getCreationTime())%>
    <br>
    session 的 Id 号:<%=session.getId()%><br>
    客户端最近一次请求时间:
    <%=session.getLastAccessedTime()%>  
    <%=new java.sql.Time(session.getLastAccessedTime())%><br>
    两次请求间隔多长时间此 session 被取消(ms):
    <%=session.getMaxInactiveInterval()%><br>
    是否是新创建的一个 session:<%=session.isNew() ?"是" : "否"%><br>
    <%
        session. setAttribute ("name", "练习 session");
        session. setAttribute ("name2", "10000");
        out.println("name"=+getAttribute("name"));
        out.println("name2"=+getAttribute("name2"));
    %></body></html>
```

以上程序显示了如何获知 session 的创建时间、sessionID 以及 session 的生命周期等。程序运行结果如图 3-2 所示。

图 3-2 程序运行结果

session 对象生命周期结束有几种情况：客户端关闭浏览器，session 过期，调用 invalidate 方法使 session 失效，等等。

为了保证系统安全，session 对象有默认的活动间隔时间，通常为 1800s，这个时间可以通过 setMaxInactiveInterval()方法设置，单位是 s(秒)。该方法的原型如下：

```
public void setMaxInactiveInterval(int n)
```

以下程序给出关于 session 生命周期的设置方法示例：

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.util.*"%>
<html>
<body>
<h2>JspSession Page</h2>
会话标识:<%=session.getId()%>
<p>创建时间:<%=new Date(session.getCreationTime())%>
<p>最后访问时间:<%=new Date(session.getLastAccessedTime())%>
<p>是否是一次新的对话???<%=session.isNew()%>
<p>原设置中的一次会话持续的时间:<%=session.getMaxInactiveInterval()%>
<%--重新设置会话的持续时间--%>
<%session.setMaxInactiveInterval(100);%>
<p>新设置中的一次会话持续的时间:<%=session.getMaxInactiveInterval()%>
<p>属性 UserName 的原值:<%=session.getAttribute("UserName")%>
<%--设置属性 UserName 的值--%>
<%session.setAttribute("UserName","The first user!");%>
<p>属性 UserName 的新值:<%=session.getAttribute("UserName")%>
</body>
</html>
```

程序运行结果如图 3-3 所示。

图 3-3　程序运行结果

3.4.2 常用技术

1. 多页面数据共享技术

对于多页面的 Web 应用系统,一个用户在一个会话期内可能出现以下两种多页面数据共享的情况:

- 登录后,把相关登录信息(如用户名、角色、权限等)保存在数据共享区内,相当于一个会话期内的全局变量,给其他页面或 Servlet 查询这些信息提供便利。
- 在特定情形下,多页面数据共享也是电子商务购物车技术实现的方案之一。多页面相当于多货架,购物车相当于多页面数据共享。

实现以上技术的原理简述如下:

(1) 数据录入:

```
session.setAttribute(String key, Object value)
```

其中,value 是任意类型的 Java 对象,当然也可以是 JSON 对象,可存放各类数据。需要注意的是,在 Servlet 中,需要通过以下方法获得 session 对象:

```
session=request. getSession()
```

(2) 数据查询:

```
session.getAttribute(String key)
```

2. 安全控制技术

主要安全控制技术如下:

- 防止非法用户绕过登录页面,直接利用 URL 进入需要登录才能进入的页面。具体解决办法有两种:其一是利用 session 中的信息,在页面中进行合法性验证;其二是利用过滤器技术(以后章节会详细介绍)。
- 当登录用户由于特殊原因暂时离开时,非法用户趁机进行非法操作,会带来意想不到的损失。对于这种情况,除了安全意识教育外,还可以利用 session 进行技术防范,其主要原理是:设置有效的 session 活动间隔时间,默认是 30min,可以人工设置 session 对象的生命周期。
- 实现安全退出机制。关闭浏览器,并不能马上触发后台结束 session,这会带来意想不到的安全隐患。可在系统中专门设立“安全退出”按钮,单击该按钮,后台实际上调用 session.invalidate()方法,服务器同时回收内存。

3.5 其他内置对象介绍

3.5.1 application 对象

application 对象实现了用户间数据的共享。与 session 对象只存放一个用户的共享数据不同，application 对象可存放所有用户的全局变量。application 对象开始于服务器的启动，随着服务器的关闭而消亡。在此期间，此对象将一直存在，这样，在用户的前后两次连接或不同用户之间的连接中，可以对此对象的同一属性进行操作；在任何地方对此对象属性的操作都将影响到其他用户对此对象的访问。服务器的启动和关闭决定了 application 对象的生命周期。它是 ServletContext 类的实例。表 3-4 是 application 对象的主要方法。

表 3-4 application 对象的主要方法

方 法 名	方 法 说 明
getAttribute(String name)	返回指定属性的值
getAttributeNames()	返回所有可用属性名
setAttribute(String name,Object object)	设置指定属性的值
removeAttribute(String name)	删除属性及其值
getServerInfo()	返回 JSP(或 Servlet)引擎名及版本号
getRealPath(String path)	返回指定虚拟路径的真实路径
getInitParameter(String name)	返回指定属性的初始值

从表 3-4 可见，application 对象的数据存取方式与 session 对象相似。在具体应用过程中，可以把 Web 应用的状态数据放入 application 对象中，如实时在线人数、公共留言等信息；也可使用 application 对象的 getlnitParameter(String paramName)方法获取 Web 应用的参数，这些参数在 web.xml 文件中使用 context-param 元素配置。

3.5.2 out 对象

out 对象代表向客户端发送数据，发送的内容是浏览器需要显示的内容。out 对象是 PrintWriter 类的实例，是向客户端输出内容时的常用对象。out 对象的主要方法如表 3-5 所示。

表 3-5　out 对象的主要方法

方　法　名	方 法 说 明
clear()	清除缓冲区的内容
clearBuffer()	清除缓冲区的当前内容
flush()	清空流
getBufferSize()	返回缓冲区的大小(以字节为单位),如不设缓冲区则为 0
getRemaining()	返回缓冲区剩余空间大小
isAutoFlush()	返回缓冲区满时是自动清空还是抛出异常
println()	向页面输出内容
close()	关闭输出流

在同步请求/响应过程中,可以用 request 对象传递数据。而在异步请求/响应过程中,则可以用 response 对象向请求端输出字符数据流(如 JSON 文本串),这是通过 out 内置对象完成的,具体如下:

(1) 前端(浏览器)的 JSP 页面或者 HTML 页面发起一个异常请求(以后章节会介绍)。

(2) 后端(如 Tomcat、Servlet)接收数据,并在处理后写回文本串数据。

```
PrintWriter out=response.getWriter();
out.write("文本串");
```

文本串可以是简单的字符串,如"ok",也可以是 JSON 文本串。具体应用在以后章节会介绍。

3.5.3　config 对象

config 对象用于在一个 Servlet 初始化时 JSP 引擎向它传递信息,此信息包括 Servlet 初始化时要用到的参数(由属性名和属性值构成)以及服务器的有关信息(通过传递一个 ServletContext 对象提供)。config 对象的主要方法如表 3-6 所示。

表 3-6　config 对象的主要方法

方　法　名	方 法 说 明
getServletContext()	返回含有服务器相关信息的 ServletContext 对象
getInitParameter(String name)	返回初始化参数的值
getInitParameterNames()	返回初始化所需的所有参数

config 对象提供了对每一个给定的服务器小程序或 JSP 页面的 javax.servlet.Servlet Config 对象的访问方法。它封装了初始化参数以及一些方法。其作用范围是当前页面,而

在别的页面无效。config 对象在 JSP 中作用不大，而在 Servlet 中作用比较大。

3.5.4 exception 对象

exception 对象用于异常处理。当一个页面在运行过程中发生了异常，就会产生此对象。如果一个 JSP 页面要应用此对象，就必须把 isErrorPage 设为 true，否则无法编译。此对象是 java.lang.Throwable 的实例。exception 对象的主要方法如表 3-7 所示。

表 3-7 exception 对象的主要方法

方 法 名	方 法 说 明
getMessage()	返回描述异常的消息
toString()	返回关于异常的简短描述消息
printStackTrace()	显示异常及其栈轨迹
FillInStackTrace()	重写异常的执行栈轨迹

下面用一个示例来说明 exception 对象的用法。首先在 errorthrow.jsp 中抛出一个异常，代码如下：

```
<%@ page language="java" import="java.util.*;" pageEncoding="ISO-8859-1"
    errorPage="error.jsp"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <body>
        <% int result=1 / 0; %>
    </body>
</html>
```

在上面的代码中，使用 page 指令设定当前页面发生异常时重定向到 error.jsp，error.jsp 的代码如下：

```
<%@ page language="java" import="java.util.*"
pageEncoding="ISO-8859-1" isErrorPage="true"%>
  <%
    String path=request.getContextPath();
    String basePath=request.getScheme() + "://" + request.getServerName() +
        ":"+request.getServerPort()+path+"/";
  %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head> <title>My JSP 'error.jsp' starting page</title></head>
<body>
error Message:getMessage() Method<br>
```

```
<% out.println(exception.getMessage());%>
<br><br>
Error String:toString() Method<br>
<% out.println(exception.toString());%>
</body>
</html>
```

程序运行结果如图 3-4 所示。

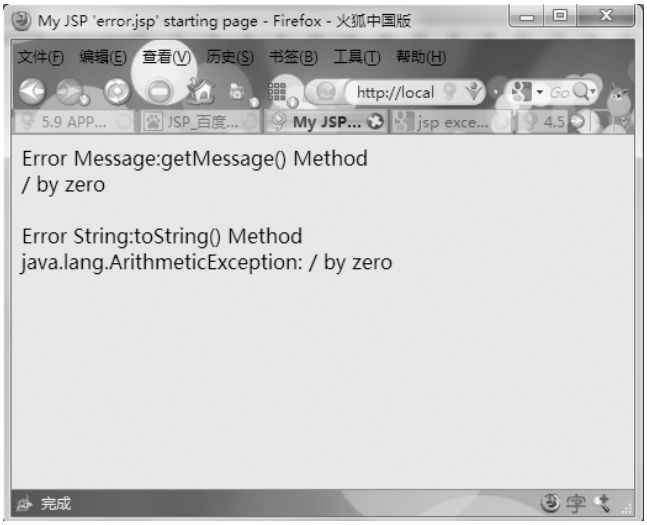

图 3-4　程序运行结果

注意,exception 对象不能在 JSP 文件中直接使用。如果要使用 exception 对象,要在 page 指令中设定<%@ isErrorPage="true"%>。

3.5.5　page 对象与 pageContext 对象

page 对象指向当前 JSP 页面本身,有点像类中的 this 指针。它是 java.lang.Object 类的实例,可以使用 Object 类的方法,例如 hashCode()、toString()等。page 对象在 JSP 程序中的应用不是很广,但是 java.lang.Object 类还是十分重要的,因为 JSP 内置对象的很多方法的返回类型是 Object,需要用到 Object 类的方法。读者可以参考相关的文档,这里就不详细介绍了。

pageContext 对象提供了对 JSP 页面内所有对象及名字空间的访问,也就是说它可以访问本页面所在的 session,也可以获取本页面所在的 application 对象的某一属性值。因此,相当于页面中所有功能的集大成者,它所属的类名也是 PageContext。pageContext 对象的主要方法如表 3-8 所示。

表 3-8　pageContext 对象的主要方法

方　法　名	方 法 说 明
getSession()	返回当前页中的 HttpSession 对象(session)
getRequest()	返回当前页的 ServletRequest 对象(request)
getResponse()	返回当前页的 ServletResponse 对象(response)
getException()	返回当前页的 Exception 对象(exception)
getServletConfig()	返回当前页的 ServletConfig 对象(config)
getServletContext()	返回当前页的 ServletContext 对象
setAttribute(String name,Object attribute)	设置属性的值
setAttribute(String name,Object obj,int scope)	在指定范围内设置属性的值
getAttribute(String name)	取属性的值
getAttribute(String name,int scope)	在指定范围内取属性的值
findAttribute(String name)	寻找指定属性,返回其属性值或 NULL
removeAttribute(String name)	删除指定属性
removeAttribute(String name,int scope)	在指定范围内删除属性
getAttributeScope(String name)	返回指定属性的作用范围
forward(String relativeUrlPath)	使当前页面导向另一页面

其中,scope 参数的值是 4 个常数,代表 4 种范围: PAGE_SCOPE 代表 page 范围, REQUEST _ SCOPE 代表 request 范围, SESSION _ SCOPE 代表 session 范围, APPLICATION_ SCOPE 代表 application 范围。

3.6 案例——主页面中的用户管理

3.6.1 需求分析

主页面如图 3-5 所示。

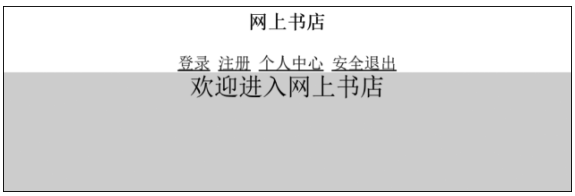

图 3-5　主页面