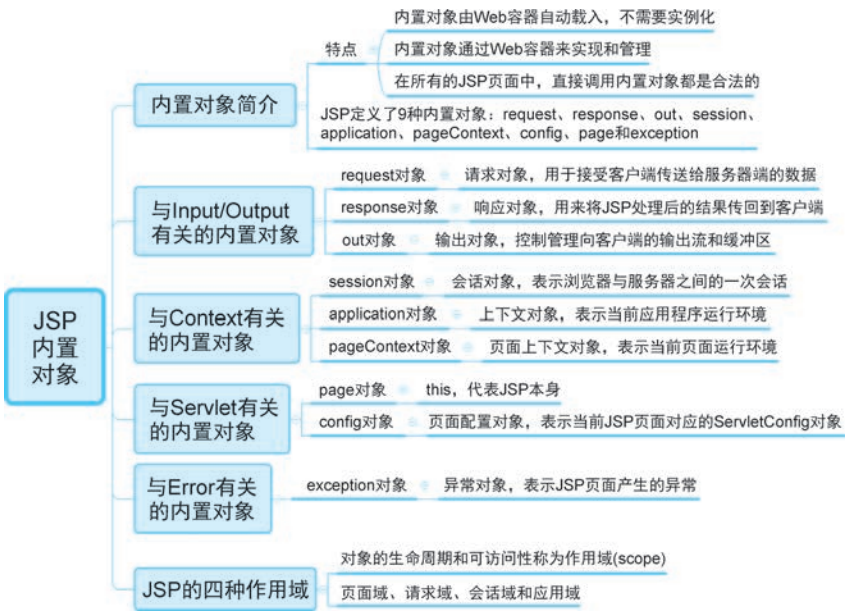
第5章 JSP内置对象

CHAPTER 5

本章思维导图



本章目标

- 熟悉 JSP 内置对象的分类及组成。
- 掌握 request、response 和 out 对象的特性及常用的使用方法。
- 掌握 session、application 对象的特性及常用的使用方法。
- 理解 pageContext、request、session、application 四种作用域的区别和联系。
- 了解 page、config 对象。
- 掌握 exception 对象的使用方式。

5.1 内置对象简介

JSP 内置对象是指在 JSP 页面中，不用声明就可以在脚本和表达式中直接使用的对象。JSP 内置对象也称隐含对象，它提供了 Web 开发常用的功能。为了提高开发效率，JSP 规

范预定义了一些内置对象。

JSP 内置对象有如下特点。

- 内置对象由 Web 容器自动载入,不需要实例化。
- 内置对象通过 Web 容器来实现和管理。
- 在所有的 JSP 页面中,直接调用内置对象都是合法的。

JSP 规范定义了 9 种内置对象,其名称、类型、功能如表 5-1 所示。

表 5-1 JSP 内置对象的名称、类型、功能

对象名称	类型	功能
request	javax. servlet. http. HttpServletRequest	请求对象,提供客户端 HTTP 请求数据的访问
response	javax. servlet. http. HttpServletResponse	响应对象,用来向客户端输出响应
out	javax. servlet. jsp. JspWriter	输出对象,提供对输出流的访问
session	javax. servlet. http. HttpSession	会话对象,用来保存服务器与每个客户端会话过程中的信息
application	javax. servlet. ServletContext	应用程序对象,用来保存整个应用环境的信息
pageContext	javax. servlet. jsp. PageContext	页面上下文对象,用于存储当前 JSP 页面的相关信息
config	javax. servlet. ServletConfig	页面配置对象,JSP 页面的配置信息对象
page	javax. servlet. jsp. HttpJspPage	当前 JSP 页面对象,即 this
exception	java. lang. Throwable	异常对象,用于处理 JSP 页面中的错误

5.2 与 Input/Output 有关的内置对象

与 Input/Output(输入/输出)有关的隐含对象包括 request 对象、response 对象和 out 对象,这类对象主要用来作为客户端和服务器间通信的桥梁。request 对象表示客户端对服务器端发送的请求;response 对象表示服务器对客户端的响应;而 out 对象负责把处理结果输出到客户端。


视频讲解

5.2.1 request 对象

request 对象即请求对象,表示客户端对服务器发送的请求,主要用于接受客户端通过 HTTP 传送给服务器端的数据。request 对象的类型为 javax. servlet. http. HttpServletRequest,与 Servlet 中的请求对象为同一对象。request 对象的作用域为一次 request 请求。

request 对象拥有 HttpServletRequest 接口的所有方法,其常用方法如下。

- void setCharacterEncoding(String charset):设置请求体参数的解码字符集。
- String getParameter(String name):根据参数名获取单一参数值。
- String[] getParameterValues(String name):根据参数名获取一组参数值。
- void setAttribute(String name,Object value):以名/值的方式存储请求域属性。
- Object getAttribute(String name):根据属性名获取存储的对象数据。

下述实例通过一个用户登录功能,演示 request 对象获取请求参数方法的使用。该实例需要两个 JSP 页面,分别是用户登录页面 login. jsp 和信息获取显示页面 loginParameter. jsp。首先创建用户登录表单页面 login. jsp,代码如下所示。

【案例 5-1】 login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>登录</title>
</head>
<body>
<form action="loginParameter.jsp" method="post">
<p>用户名:<input name="username" type="text"></p>
<p>密   码:<input name="password" type="password"></p>
<p><input name="submit" type="submit" value="登录"></p>
</form>
</body>
</html>
```

启动服务器,在浏览器中访问 http://localhost:8080/ch05/login.jsp,运行结果如图 5-1 所示。

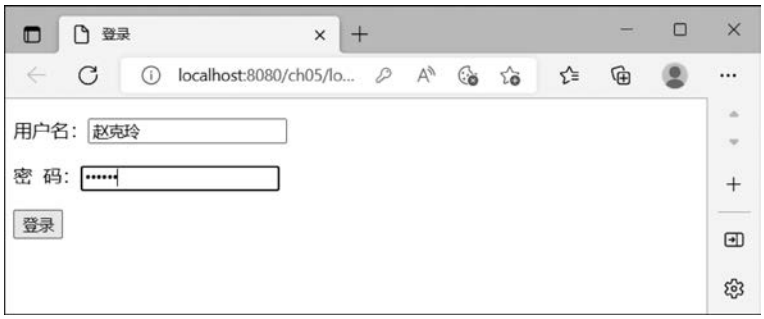


图 5-1 login.jsp 运行结果

单击“登录”按钮,form 表单将向 loginParameter.jsp 发送请求数据,loginParameter.jsp 从请求对象(request)中获取请求参数并输出显示,代码如下所示。

【案例 5-2】 loginParameter.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="java.util.Enumeration,java.util.Map"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>获取登录请求参数</title>
</head>
<body>
<%
//设置 POST 请求编码
```

```
request.setCharacterEncoding("UTF-8");
//获取请求参数的值
String username = request.getParameter("username");
String password = request.getParameter("password");

out.println("参数 username 的值:" + username + "<br>");
out.println("参数 password 的值:" + password + "<br>");
%>
</body>
</html>
```

提交表单后,loginParameter.jsp 的运行结果如图 5-2 所示。

图 5-2 loginParameter.jsp 运行结果

request 对象获取请求参数的方法既适用于 URL 查询字符串的 GET 请求,也适用于 Form 表单的 POST 请求。

request 对象可以通过 setAtrribute()和 getAttribute()方法存取请求域属性,在实际开发中,多用于存储、传递本次请求的处理结果。下述实例代码用来实现对案例 5-1 中 login.jsp 的登录信息进行验证,并将产生的验证结果回传到 login.jsp 页面中进行显示。其中,登录信息验证的代码如下所示。

【案例 5-3】 loginValidate.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>登录验证</title>
</head>
<body>
<%
//设置 POST 请求编码
request.setCharacterEncoding("UTF-8");
//获取请求参数
String username = request.getParameter("username");
String password = request.getParameter("password");
StringBuffer errorMsg = new StringBuffer();
//参数信息验证
if("".equals(username))
```

```
        errorMsg.append("用户名不能为空!<br>");
    if("".equals(password))
        errorMsg.append("密码不能为空!<br>");
    else
        if(password.length() < 6 || password.length() > 12)
            errorMsg.append("密码长度需为 6～12 位.<br>");

    //将错误信息保存在请求域属性 errorMsg 中
    request.setAttribute("errorMsg", errorMsg.toString());

    if(errorMsg.toString().equals(""))
        out.println(username + ",您的登录信息验证成功!");
    else{
    %>
    <jsp:forward page="login.jsp"></jsp:forward>
    <%
    }
    %>
    </body>
    </html>
```

在登录页面中加入验证信息的获取和显示的代码如下所示。

【案例 5-4】 login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>登录</title>
</head>
<body>
<%
//从请求域属性 errorMsg 中获取错误信息
String error = (String)request.getAttribute("errorMsg");
if(error != null)
    out.print("<font color='red'>" + error + "</font>");
%>
<form action="loginValidate.jsp" method="post">
<p>用户名:<input name="username" type="text"></p>
<p>密   码:<input name="password" type="password"></p>
<p><input name="submit" type="submit" value="登录"></p>
</form>
</body>
</html>
```

注意 在一开始请求 login.jsp 时,request 对象中还未设置 errorMsg 属性的情况,此时需要先进行属性是否存在的判断(即 getAttribute()方法返回值是否为 null),否则页面会显示“null”字样。

启动服务器,在浏览器中访问 http://localhost:8080/ch05/login.jsp,在用户名、密码

都不填写直接登录的情况下,运行结果如图 5-3 所示。

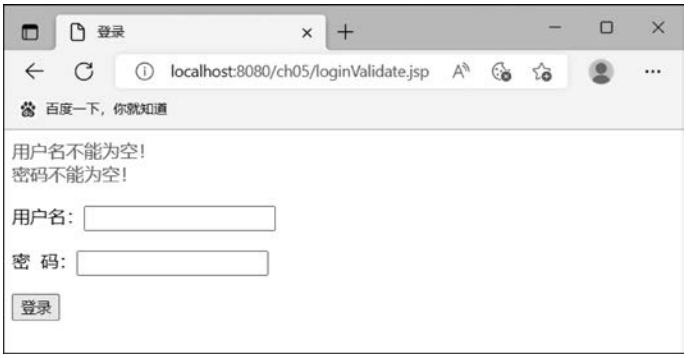


图 5-3 登录信息验证运行结果

上述代码中,验证错误信息以请求域属性的形式保存在 request 对象中,并通过请求转发的方式将请求对象再转发回 login.jsp,在 login.jsp 页面中便可从 request 对象中获取到属性值,从而实现验证信息在一次 request 请求范围内的传递。

5.2.2 response 对象

response 对象即响应对象,表示服务器对客户端的响应。主要用来将 JSP 处理后的结果传回到客户端。response 对象类型为 javax.servlet.http.HttpServletResponse,与 Servlet 中的响应对象为同一对象。

response 对象拥有 HttpServletResponse 接口的所有方法,其常用的方法如下。

- void setContentType(String name):设置响应内容的类型和字符编码。
- void sendRedirect(String url):重定向到指定的 URL 资源。

下述实例代码演示使用 sendRedirect()方法,在案例 5-3 中 loginValidate.jsp 登录信息验证成功时重定向到用户主页面 main.jsp。更改后的 loginValidate.jsp 如下所示。

【案例 5-5】 loginValidate.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>登录验证</title>
</head>
<body>
<%
//设置 POST 请求编码
request.setCharacterEncoding("UTF-8");
//获取请求参数
String username = request.getParameter("username");
String password = request.getParameter("password");
StringBuffer errorMsg = new StringBuffer();
//参数信息验证
if("".equals(username))
```

```
        errorMsg.append("用户名不能为空!<br>");
    if("".equals(password))
        errorMsg.append("密码不能为空!<br>");
    else
        if(password.length() < 6 || password.length() > 12)
            errorMsg.append("密码长度需为 6～12 位.<br>");
    //将错误信息保存在请求域属性 errorMsg 中
    request.setAttribute("errorMsg", errorMsg.toString());

    if(errorMsg.toString().equals("")){
        //验证成功,重定向到 main.jsp
        response.sendRedirect("main.jsp");
    }else{
    %>
    <jsp:forward page="login.jsp"></jsp:forward>
    <%
    }
    %>
    </body>
    </html>
```

用户主界面 main.jsp 的代码如下所示。

【案例 5-6】　main.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>用户主界面</title>
</head>
<body>
欢迎您!
</body>
</html>
```

启动服务器,在浏览器中访问 http://localhost:8080/ch05/main.jsp,在验证信息填写正确的情况下登录后,运行结果如图 5-4 所示。

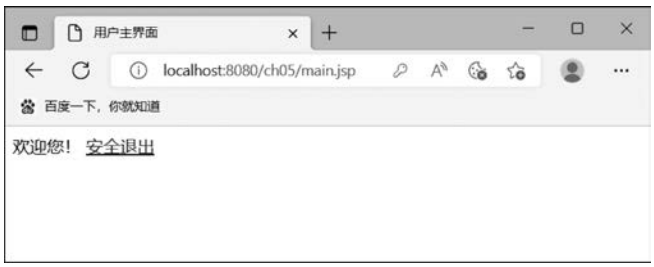


图 5-4　登录信息验证正确情况下的重定向结果

注意　因这里是使用重定向进行的页面跳转,故不能使用请求域属性进行用户名的传递。

5.2.3 out 对象

out 对象即输出对象，用来控制管理输出的缓冲区(buffer)和输出流(output stream)向客户端页面输出数据。out 对象类型为 javax.servlet.jsp.JspWriter，与 HttpServletResponse 接口的 getWriter()方法获得的 PrintWriter 对象功能相同，并都由 java.io.Writer 类继承而来。

out 对象的方法可以分为以下两类。

- 数据的输出。
- 缓冲区的处理。

其中数据输出的方法及描述如表 5-2 所示。

表 5-2 out 对象的数据输出方法及描述

方　法	描　述
print/println(基本数据类型)	输出一个基本数据类型的值
print/println(Object obj)	输出一个对象的引用地址
print/println(String str)	输出一个字符串的值
newLine()	输出一个换行符

【示例】 out 对象的数据输出方法

```
<%
int i = 0;
java.util.Date date = new java.util.Date();
out.print(i);
out.newLine();
out.println(date);
%>
```

注意 out 对象的 newLine()和 println()方法在页面显示上并不会有换行的效果，但在生成的 HTML 页面源代码中，这两个方法会在输出的数据后面进行换行。

out 对象缓冲区的处理方法及描述如表 5-3 所示。

表 5-3 out 对象缓冲区的处理方法及描述

方　法	描　述
void clear()	清除输出缓冲区的内容。若缓冲区为空，则产生 IOException 异常
void clearBuffer()	清除输出缓冲区的内容。若缓冲区为空，不会产生 IOException 异常
void flush()	直接将目前暂存于缓冲区的数据刷新输出
void close()	关闭输出流。一旦关闭，就不能再使用 out 对象做任何操作
int getBufferSize()	获取目前缓冲区的大小(KB)
int getRemaining()	获取目前使用后还剩下的缓冲区大小(KB)
boolean isAutoFlush()	返回 true 表示缓冲区满时会自动刷新输出；返回 false 表示缓冲区满时不会自动清除并产生异常处理

向 out 对象的输出流中写入数据时，数据会先被存储在缓冲区中，在 JSP 默认配置下，缓冲区满时会被自动刷新输出。相关的配置由 JSP 页面中 page 指令的 autoFlush 属性和 buffer 属性决定，autoFlush 属性表示是否自动刷新，默认值为 true；buffer 属性表示缓冲区大小，默认值为 8KB。在此配置下，out 对象在输出缓冲区内容每次达到 8KB 后，会自动刷

新输出而不会产生异常处理。

下述代码演示在取消自动刷新功能时，页面输出信息超过缓冲区指定大小的情况和使用 out.flush()刷新方法后的情况。

【案例 5-7】 outExample.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" autoFlush="false" buffer="1kb" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
for(int i=0;i<100;i++){
    out.println("********************");
    //out.flush();
}
%>
</body>
</html>
```

启动服务器，在浏览器中访问 http://localhost:8080/ch05/outExample.jsp，运行结果如图 5-5 所示。

图 5-5 outExample.jsp 运行结果

从运行结果可以看出，在取消了页面自动刷新功能(autoFlush="false")后，当输出流内容超过缓冲区大小(buffer="1KB")时，页面不能被正常执行。若在输出信息代码后面加上“out.flush()”刷新缓冲区的代码，在每次循环输出内容不超过 1KB 的情况下，内容被及时刷新输出，页面恢复正常运行，运行结果如图 5-6 所示。

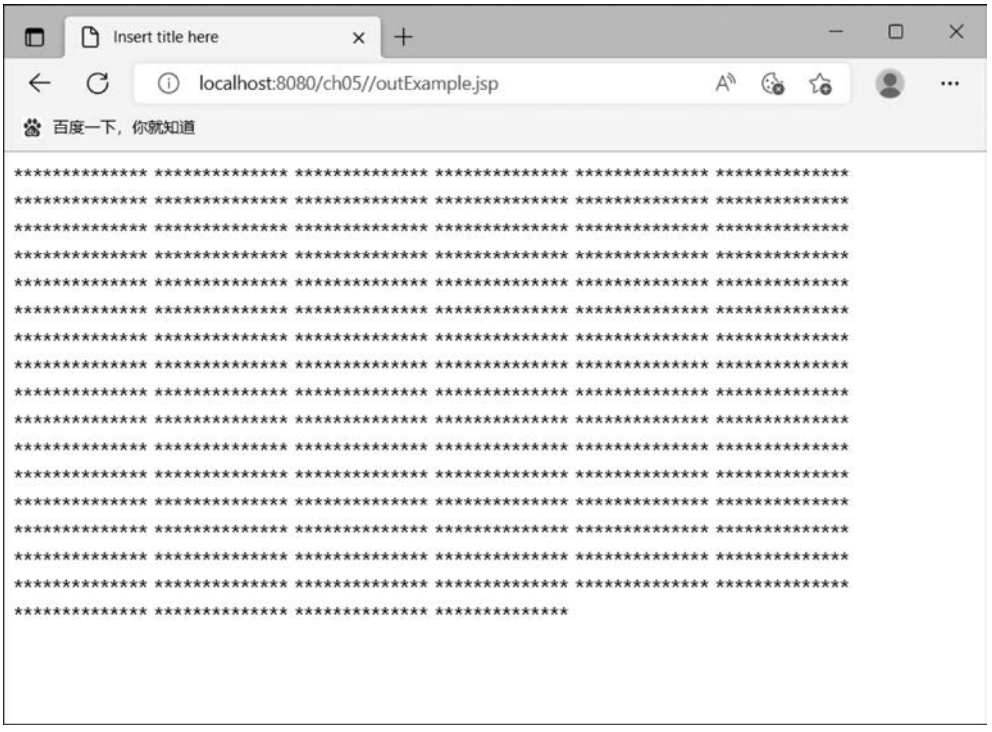


图 5-6　outExample.jsp 更改代码后的运行结果

5.3　与 Context 有关的内置对象

与 Context(上下文)有关的内置对象包括 session、application 和 pageContext。其中 session 对象表示浏览器与服务器的会话上下文环境；application 对象表示应用程序上下文环境；pageContext 对象表示当前 JSP 页面上下文环境。

5.3.1　session 对象

session 对象即会话对象，表示浏览器与服务器之间的一次会话。一次会话的含义是：从客户端浏览器连接服务器开始，到服务器端会话过期或用户主动退出后，会话结束。这个过程可以包含浏览器与服务器之间的多次请求与响应。

session 对象的类型为 javax.servlet.http.HttpSession，session 对象具有 HttpSession 接口的所有方法，其常用方法如下。

- void setAttribute(String name, Object value)：以名/值对的方式存储 session 域属性。
- Object getAttribute(String name)：根据属性名获取属性值。
- void invalidate()：使 session 对象失效，释放所有的属性空间。

下述代码演示使用 setAttribute()方法对用户登录验证成功后的用户名进行保存，在重定向的用户主界面中使用 getAttribute()方法获取用户名。改进后的 loginValidate.jsp 如下所示。

【案例 5-8】 loginValidate.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>登录验证</title>
</head>
<body>
<%
//设置 POST 请求编码
request.setCharacterEncoding("UTF-8");
//获取请求参数
String username = request.getParameter("username");
String password = request.getParameter("password");
StringBuffer errorMsg = new StringBuffer();
//参数信息验证
if("".equals(username))
    errorMsg.append("用户名不能为空!<br>");
if("".equals(password))
    errorMsg.append("密码不能为空!<br>");
else
    if(password.length() < 6 || password.length() > 12)
        errorMsg.append("密码长度需为 6～12 位.<br>");
//将错误信息保存在请求域属性 errorMsg 中
request.setAttribute("errorMsg", errorMsg.toString());

if(errorMsg.toString().equals("")){
    //将用户名存储在 session 域属性 username 中
    session.setAttribute("username", username);
    //验证成功,重定向到 main.jsp
    response.sendRedirect("main.jsp");
}else{
%>
<jsp:forward page="login.jsp"></jsp:forward>
<%
}
%>
</body>
</html>
```

重定向的 main.jsp 中获取用户名的改进代码如下所示。

【案例 5-9】 main.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>用户主界面</title>
</head>
<body>
欢迎您!
<%
String username = (String)session.getAttribute("username");
if(username != null)
    out.print(username);
%>
</body>
</html>
```

启动服务器,在浏览器中访问 http://localhost:8080/ch05/login.jsp,在登录页面中输入格式正确的用户名和密码登录后,运行结果如图 5-7 所示。

图 5-7 session 范围中用户名的运行结果

从运行结果可以看出,存储在 session 范围中的属性即使经过重定向的多次请求仍然有效。在浏览器未关闭的情况下,访问 main.jsp 一直可以获取到用户名,若要让其失效,可以使用 invalidate()方法。下述代码演示在 main.jsp 中增加“安全退出”功能,退出后重新返回登录页面。main.jsp 的改进代码如下所示。

【案例 5-10】 main.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>用户主界面</title>
</head>
<body>
欢迎您!
<%
String username = (String)session.getAttribute("username");
if(username != null)
    out.print(username);
%>
<a href="logout.jsp">安全退出</a>
</body>
</html>
```

实现退出功能的 logout.jsp 的代码如下所示。

【案例 5-11】 logout.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
session.invalidate();
response.sendRedirect("login.jsp");
%>
```

启动服务器，在浏览器中访问 http://localhost:8080/ch05/main.jsp，如图 5-8 所示单击“安全退出”。

图 5-8 单击“安全退出”

此时若再访问 http://localhost:8080/ch05/main.jsp，会发现用户名不再显示，表示上次会话已经失效，新的会话已经开始。

注意 考虑 session 本身的目的，通常应只把与用户会话状态相关的信息放入 session 范围内，不要仅为了两个页面之间传递信息就将信息放入 session 范围，这样会加大服务器端的开销。如果仅为了两个页面交换信息，应将该信息放入 request 范围内，然后通过请求转发即可。

5.3.2 application 对象


视频讲解

application 对象即应用程序上下文对象，表示当前应用程序运行环境，用以获取应用程序上下文环境中的信息。application 对象在容器启动时实例化，在容器关闭时销毁。作用域为整个 Web 容器的生命周期。

application 对象实现了 javax.servlet.ServletContext 接口，具有 ServletContext 接口的所有功能，application 对象常用方法如下。

- void setAttribute(String name,Object value)：以名/值对的方式存储 application 域属性。
- Object getAttribute(String name)：根据属性名获取属性值。
- void removeAttribute(String name)：根据属性名从 application 域中移除属性。

下述实例演示使用 application 对象实现一个页面留言板，代码如下所示。

【案例 5-12】 guestBook.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" import="java.util.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>用户留言板</title>
<script type="text/javascript">
function validate(){
    var uname = document.getElementById("username");
    var message = document.getElementById("message");
    if(uname.value == ""){
        alert("请填写您的名字!");
        uname.focus();
        return false;
    }else if(message.value == ""){
        alert("请填写留言");
        message.focus();
        return false;
    }
    return true;
}
</script>
</head>
<body>
<p>请留言</p>
<form action="guestBook.jsp" method="post" onsubmit="return validate();">
<p>输入您的名字:<input name="username" id="username" type="text"></p>
<p>输入您的留言:<textarea name="message" id="message" cols="50" rows="3"></textarea>
</p>
<p><input type="submit" value="提交留言"></p>
</form>
<hr>
<p>留言内容</p>
<%
//获取留言信息
request.setCharacterEncoding("UTF-8");
String username=request.getParameter("username");
String message=request.getParameter("message");
//从 application 域属性 messageBook 中获取留言本
Vector<String> book = (Vector<String>)application.getAttribute("messageBook");
if(book == null)//若留言本不存在则新创建一个
    book = new Vector<String>();
//判断用户是否提交了留言,若已提交,则将提交信息加入留言本,存入 application 域属性中
if(username!=null && message!=null){
    String info=username+"#"+message;
    book.add(info);
    application.setAttribute("messageBook", book);
}
//遍历显示出所有的用户留言
if(book.size()>0){
    for(String mess:book){
        String[] arr = mess.split("#");
        out.print("<p>姓名:"+arr[0]+"<br>留言:"+arr[1]+"</p>");
    }
}else{
    out.print("还没有留言!");
}
```

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

上述代码中，使用 Vector 集合类存放用户的每次留言，并将其作为 application 域属性 messageBook 的值，这样 Vector 对象在整个服务器生命周期内就可以不断添加各客户端提交的留言信息。启动服务器，在浏览器中访问 http://localhost:8080/ch05/guestBook.jsp，运行结果如图 5-9 所示。

图 5-9　guestBook.jsp 运行结果

5.3.3　pageContext 对象

pageContext 即页面上下文对象，表示当前页面运行环境，用于获取当前 JSP 页面的相关信息。pageContext 对象作用范围为当前 JSP 页面。

pageContext 对象类型为 javax.servlet.jsp.PageContext，pageContext 对象可以访问当前 JSP 页面所有的内置对象，如表 5-4 所示。另外 pageContext 对象还提供存取页面域属性的方法，如表 5-5 所示。

表 5-4　pageContext 对象获取内置对象的方法及描述

方　法	描　述
ServletRequest getRequest()	获取当前 JSP 页面的请求对象
ServletResponse getResponse()	获取当前 JSP 页面的响应对象
HttpSession getSession()	获取和当前 JSP 页面有联系的会话对象
ServletConfig getServletConfig()	获取当前 JSP 页面的 ServletConfig 对象
ServletContext getServletContext()	获取当前 JSP 页面的运行环境 application 对象
Object getPage()	获取当前 JSP 页面的 Servlet 实体 page 对象
Exception getException()	获取当前 JSP 页面的异常 exception 对象，不过此页面的 page 指令的 isErrorPage 属性要设为 true
JspWriter getOut()	获取当前 JSP 页面的输出流 out 对象

表 5-5　pageContext 对象存取页面域属性的方法及描述

方　法	描　述
Object getAttribute(String name, int scope)	获取范围为 scope，名为 name 的属性对象
void setAttribute(String name, Object value, int scope)	以名/值对的方式存储 scope 范围域属性

续表

方　法	描　述
void removeAttribute(String name, int scope)	从 scope 范围移除名为 name 的属性
Enumeration getAttributeNamesInScope(int scope)	从 scope 范围中获取所有属性的名称

在表 5-5 存取域属性的方法中 scope 参数被定义为四个常量，分别代表四种作用域范围：PAGE_SCOPE＝1 代表页面域，REQUEST_SCOPE＝2 代表请求域，SESSION_SCOPE＝3 代表会话域，APPLICATION_SCOPE＝4 代表应用域。

【示例】 添加和获取会话域属性

```
<%
pageContext.getSession().setAttribute("sessionKey","QST");
Object object = pageContext.getAttribute("sessionKey",pageContext.SESSION_SCOPE);
%>
<% = object %>
```

5.4 与 Servlet 有关的内置对象

与 Servlet 有关的内置对象包括 page 对象和 config 对象。page 对象表示 JSP 翻译后的 Servlet 对象，config 对象表示 JSP 翻译后的 Servlet 的 ServletConfig 对象。

5.4.1 page 对象

page 对象即 this，代表 JSP 本身，更准确地说它代表 JSP 被翻译后的 Servlet，因此它可以调用 Servlet 类所定义的方法。page 对象的类型为 javax.servlet.jsp.HttpJspPage，在实际应用中，page 对象很少在 JSP 中使用。

下述代码演示 page 对象获取页面 page 指令的 info 属性指定的页面说明信息。

【案例 5-13】 pageExample.jsp

```
<%@ page language = "java" contentType = "text/html; charset = UTF - 8"
    pageEncoding = "UTF - 8" info = "page 内置对象的使用" %>
<!DOCTYPE html PUBLIC " - //W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http - equiv = "Content - Type" content = "text/html; charset = UTF - 8">
<title> Insert title here </title>
</head>
<body>
<p>使用"this"获取的页面说明信息:<% = this.getServletInfo() %></p>
<p>使用"page"获取的页面说明信息:<% = ((HttpJspPage)page).getServletInfo() %></p>
</body>
</html>
```

启动服务器，在浏览器中访问 http://localhost:8080/ch05/pageExample.jsp，运行结果如图 5-10 所示。

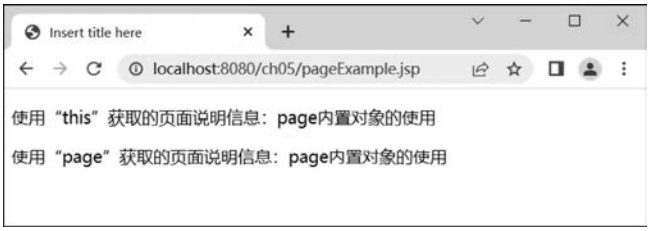


图 5-10 pageExample.jsp 运行结果

5.4.2 config 对象

config 对象即页面配置对象，表示当前 JSP 页面翻译后的 Servlet 的 ServletConfig 对象，存放着一些初始的数据结构。config 对象实现于 java.servlet.ServletConfig 接口。config 对象和 page 对象一样都很少被用到。

下述实例演示 JSP 通过 config 对象获取初始化参数。

【案例 5-14】 configExample.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
String initParam = config.getInitParameter("init");
out.println(initParam);
%>
</body>
</html>
```

初始化参数在 web.xml 文件中的配置如下所示。

【案例 5-15】 web.xml

```
<servlet>
    <servlet-name>configExample</servlet-name>
    <jsp-file>/configExample.jsp</jsp-file>
    <init-param>
        <param-name>init</param-name>
        <param-value>JSP 初始化参数值</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>configExample</servlet-name>
    <url-pattern>/configExample.jsp</url-pattern>
</servlet-mapping>
```

启动服务器，在浏览器中访问 http://localhost:8080/ch05/configExample.jsp，运行结果如图 5-11 所示。

图 5-11　configExample.jsp 的运行结果

5.5　与 Error 有关的内置对象

与 Error 有关的内置对象只有一个成员：exception 对象。当 JSP 网页有错误时会产生异常，exception 对象就用来处理这个异常。

exception 对象即异常对象，表示 JSP 页面产生的异常。需要注意的是，如果一个 JSP 页面要应用此对象，必须将此页面中 page 指令的 isErrorPage 属性值设为 true，否则无法编译。exception 对象是 java.lang.Throwable 的对象。

下述代码描述 exception 对象对页面异常的处理。

【案例 5-16】　error.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" isErrorPage="true" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<% exception.printStackTrace(response.getWriter()); %>
</body>
</html>
```

下述代码描述产生异常的页面，需要注意页面中 page 指令的 errorPage 属性要指向上面定义的异常处理页面“error.jsp”。

【案例 5-17】　calculate.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" errorPage="error.jsp" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
    <title>计算</title>
    </head>
    <body>
        <%
            int a, b;
            a = 10;
            b = 0;
            int c = a / b;
        %>
    </body>
    </html>
```

启动服务器，在浏览器中访问 http://localhost:8080/ch05/calculate.jsp，运行结果如图 5-12 所示。

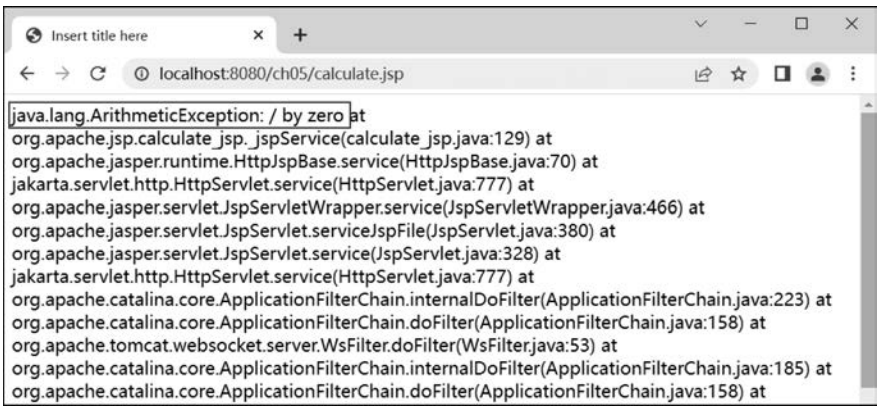


图 5-12　calculate.jsp 的运行结果

5.6　JSP 的四种作用域


视频讲解

对象的生命周期和可访问性称为作用域(scope)，在 JSP 中有四种作用域：页面域、请求域、会话域和应用域。四种作用域的生命周期和可访问性介绍如下。

- 页面域(page scope)，页面域的生命周期是指页面执行期间。存储在页面域的对象只对于它所在页面是可访问的。
- 请求域(request scope)，请求域的生命周期是指一次请求过程，包括请求被转发(forward)或者被包含(include)的情况。存储在请求域中的对象只有在此次请求过程中才可以被访问。
- 会话域(session scope)，会话域的生命周期是指某个客户端与服务器所连接的时间；客户端在第一次访问服务器时创建会话，在会话过期或用户主动退出后，会话结束。存储在会话域中的对象在整个会话期间(可能包含多次请求)都可以被访问。
- 应用域(application scope)，应用域的生命周期是指从服务器开始执行服务到服务器关闭为止，是四个作用域中时间最长的。存储在应用域中的对象在整个应用程序运行期间可以被所有 JSP 和 Servlet 共享访问，在使用时要特别注意存储数据的大小和安全性，否则可能会造成服务器负载过重和线程安全性问题。

JSP 的四种作用域分别对应 pageContext、request、session 和 application 四个内置对象，四个内置对象都通过 setAttribute(String name,Object value)方法来存储属性，通过 getAttribute(String name)来获取属性，从而实现属性对象在不同作用域的数据分享。

下述代码演示使用 pageContext、session、application 对象分别实现页面域、会话域、应用域的页面访问统计效果。

【案例 5-18】 visitCount.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>访问统计</title>
</head>
<body>
    <%
        int pageCount = 1;
        int sessionCount = 1;
        int applicationCount = 1;
        //页面域计数
        if (pageContext.getAttribute("pageCount") != null) {
            pageCount = Integer.parseInt(pageContext.getAttribute(
                    "pageCount").toString());
            pageCount++;
        }
        pageContext.setAttribute("pageCount", pageCount);
        //会话域计数
        if (session.getAttribute("sessionCount") != null) {
            sessionCount = Integer.parseInt(session.getAttribute(
                    "sessionCount").toString());
            sessionCount++;
        }
        session.setAttribute("sessionCount", sessionCount);
        //应用域计数
        if (application.getAttribute("applicationCount") != null) {
            applicationCount = Integer.parseInt(application.getAttribute(
                    "applicationCount").toString());
            applicationCount++;
        }
        application.setAttribute("applicationCount", applicationCount);
    %>
    <p>
        页面域计数:<%=pageCount%></p>
    <p>
        会话域计数:<%=sessionCount%></p>
    <p>
        应用域计数:<%=applicationCount%></p>
</body>
</html>
```

启动服务器，在 Chrome 浏览器中访问 http://localhost:8080/ch05/visitCount.jsp，第一次访问该页面，运行结果如图 5-13 所示。

图 5-13　第一次访问 visitCount.jsp

多次刷新 Chrome 浏览器窗口后，运行结果如图 5-14 所示。

图 5-14　visitCount.jsp 多次刷新后

另外打开一个 Firefox 浏览器窗口（此处使用不同的浏览器软件，同种浏览器的不同窗口 session 可能依然累加），再次访问此页面后，运行结果如图 5-15 所示。

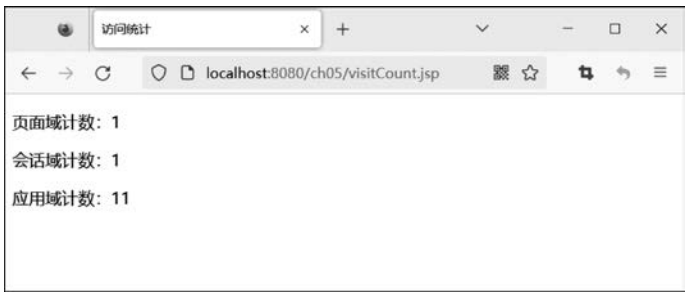


图 5-15　新开浏览器窗口访问 visitCount.jsp

通过上述运行结果可以看出，pageContext 域属性的访问范围为当前 JSP 页面，因此访问计数始终为 1；session 域属性的访问范围为当前浏览器与服务器的会话，因此刷新页面访问计数会累加，但新开启浏览器窗口时，会新建一个会话，计数又会从 1 开始；application 域属性的访问范围为整个应用，所以只要应用程序不停止运行，计数会不断累加。

在 Web 应用开发时需要仔细考虑这些对象的作用域，按照对象的需要赋予适合的作用域，不要过大也不要过小。为一个只在页面内使用的对象赋予了应用域显然毫无意义；但如果访问对象有太多的限制，那么也会使应用变得更加复杂。因此需要仔细权衡每个对象及其用途，从而准确推断其作用域。

本章总结

- JSP 内置对象是指不用声明就可以在 JSP 页面的脚本和表达式中直接使用的对象。
- request 对象即请求对象，表示客户端向服务器端发送的请求。request 对象的类型为 javax. servlet. http. HttpServletRequest。
- response 对象即响应对象，表示服务器对客户端的响应。response 对象类型为 javax. servlet. http. HttpServletResponse。
- out 对象即输出对象，用来控制管理输出的缓冲区(buffer)和输出流(output stream)向客户端页面输出数据。out 对象类型为 javax. servlet. jsp. JspWriter。
- session 对象即会话对象，表示浏览器与服务器之间的一次会话。session 对象的类型为 javax. servlet. http. HttpSession。
- application 对象即应用程序上下文对象，表示当前应用程序运行环境，用以获取应用程序上下文环境中的信息。application 对象类型为 javax. servlet. ServletContext。
- pageContext 即页面上下文对象，表示当前页面运行环境，用以获取当前 JSP 页面的相关信息。pageContext 对象类型为 javax. servlet. jsp. PageContext。
- page 对象即 this，代表 JSP 本身，更准确地说它代表 JSP 被翻译后的 Servlet，因此它可以调用 Servlet 类所定义的方法。page 对象的类型为 javax. servlet. jsp. HttpJspPage。
- config 对象即页面配置对象，表示当前 JSP 页面翻译后的 Servlet 的 ServletConfig 对象，存放着一些初始的数据结构。config 对象类型为 java. servlet. ServletConfig。
- exception 对象即异常对象，表示 JSP 页面产生的异常。exception 对象是 java. lang. Throwable 的对象。
- JSP 中有四种作用域：页面域、请求域、会话域和应用域。
- JSP 的四种作用域分别对应 pageContext、request、session 和 application 四个内置对象。四个内置对象都通过 setAttribute(String name，Object value)方法来存储属性，通过 getAttribute(String name)来获取属性，从而实现属性对象在不同作用域的数据分享。

本章习题

1. 下面_____不是 JSP 的内置对象。

   A. session　　B. request　　C. cookie　　D. out

2. response 对象的 setHeader(String name，String value)方法的作用是_____。

   A. 添加 HTTP 文件头

   B. 设定指定名字的 HTTP 文件头的值

   C. 判断指定名字的 HTTP 文件头是否存在

   D. 向客户端发送错误信息

3. 要设置某个 JSP 页面为错误处理页面，以下 page 指令正确的是_____。

   A. <%@ page errorPage="true"%>

B. <%@ page isErrorPage="true"%>

C. <%@ page extends="javax.servlet.jsp.JspErrorPage"%>

D. <%@ page info="error"%>

4. 下面关于JSP作用域对象的说法错误的是_____。

A. request对象可以得到请求中的参数

B. session对象可以保存用户信息

C. application对象可以被多个应用共享

D. 作用域范围从小到大是page、request、session、application

5. 在JSP中，request对象的_____方法可以获取页面请求中一个表单组件对应多个值时的用户的请求数据。

A. String getParameter(String name)

B. String[] getParameter(String name)

C. String getParameterValuses(String name)

D. String[] getParameterValues(String name)

6. 如果选择一种对象保存聊天室信息，则选择_____。

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

7. JSP中获取输入参数信息，使用_____对象的getParameter()方法。

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

8. JSP中保存用户会话信息使用_____对象。

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

9. 以下对象中作用域最大的是_____。

A. applicant　　B. request　　C. page　　D. session

10. 创建a.jsp页面，将一个字符串存入请求域属性temp中，转发请求到b.jsp；在b.jsp中获取并显示temp的值；将请求转发到b.jsp改为重定向到b.jsp，观察是否能够获取temp的值。

11. 充分利用session和application的特点，实现一个禁止用户使用同一用户名同时在不同客户端登录的功能程序。

12. 创建exceptionTest.jsp页面，模拟一个空指针异常，指定异常处理页面为error.jsp；使用exception内置对象在异常处理页面error.jsp中输出异常信息。