# 第 3 章 JSP内置对象

## 学习目的与要求

本章主要介绍 request、response、out、session、application、pageContext、page、config 和 exception 等内置对象。通过本章的学习，要求读者理解 JSP 内置对象的含义，掌握 JSP 内置对象的使用方法。

## 本章主要内容

- request 对象
- response 对象
- out 对象
- session 对象
- application 对象
- pageContext 对象
- page 对象
- config 对象
- exception 对象

有些对象在 JSP 页面中不需要声明和实例化，可以直接在 Java 程序片和 Java 表达式部分使用，称这样的对象为 JSP 内置对象。JSP 内置对象由 Web 服务器负责实现和管理，JSP 自带了 9 个功能强大的内置对象，共分为 4 类。

（1）与 Input/Output 有关的内置对象：与 Input/Output 有关的内置对象包括 request、response 和 out，该类对象主要用来作为客户端和服务器之间通信的桥梁。request 对象表示客户端对服务器发送的请求；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。

（2）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会话的上下文环境；application 对象表示应用程序（Web 应用）的上下文环境；pageContext 对象表示当前 JSP 页面的上下文环境。

（3）与 Servlet 有关的内置对象：与 Servlet 有关的内置对象包括 page 和 config。page 对象表示 JSP 文件转换为 Java 文件后的 Servlet 对象；config 对象表示 JSP 文件转换为 Java 文件后的 Servlet 的 ServletConfig 对象。

（4）与 Error 有关的内置对象：与 Error 有关的内置对象只有一个 exception 对象。当 JSP 网页有错误时将产生异常，该对象用来处理这个异常。

按照 1.2.2 节的操作步骤创建一个 Web 项目 ch3，并为 ch3 添加 Tomcat 依赖。本章涉及的 JSP 页面保存在 ch3 项目的 web 目录中。

扫一扫

视频讲解

# 3.1 request 对象

request 对象的类型为 jakarta.servlet.http.HttpServletRequest。当客户端请求一个 JSP 页面时，JSP 页面所在的服务器将客户端发出的所有请求信息封装在内置对象 request 中，因此使用该对象就可以获取客户端提交的信息。

## 3.1.1 request 对象的常用方法

request 对象的常用方法如表 3.1 所示。

表 3.1 request 对象的常用方法

| 序号 | 方 法 | 功 能 说 明 |
|---|---|---|
| 1 | Object getAttribute(String name) | 返回指定属性的属性值 |
| 2 | Enumeration getAttributeNames() | 返回所有可用属性名的枚举 |
| 3 | String getCharacterEncoding() | 返回字符编码方式 |
| 4 | int getContentLength() | 返回请求体的字节数 |
| 5 | String getContentType() | 返回请求体的 MIME 类型 |
| 6 | ServletInputStream getInputStream() | 返回请求体中一行的二进制流 |
| 7 | String getParameter(String name) | 返回 name 参数的值 |
| 8 | Enumeration getParameterNames() | 返回可用参数名的枚举 |
| 9 | String[] getParameterValues(String name) | 返回包含 name 参数的所有值的数组 |
| 10 | String getProtocol() | 返回请求用的协议类型及版本号 |
| 11 | String getServerName() | 返回接受请求的服务器的主机名 |
| 12 | int getServerPort() | 返回服务器接受此请求所用的端口号 |
| 13 | String getRemoteAddr() | 返回发送此请求的客户端的 IP 地址 |
| 14 | String getRemoteHost() | 返回发送此请求的客户端的主机名 |
| 15 | void setAttribute(String key,Object obj) | 设置属性的值 |
| 16 | String getRealPath(String path) | 返回一个虚拟路径的真实路径 |

### ❶ 用 request 对象获取客户端提交的信息

1) String getParameter(String name)

该方法以字符串的形式返回客户端传来的某个参数的值，该参数名由 name 指定。

【例 3-1】 调用 getParameter(String name)方法获取表单信息。

example3_1.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_1.jsp</title>
</head>
<body>
<form action="getValue.jsp">
        <input type="text" name="userName"/>
        <input type="submit" value="提交"/>
</form>
</body>
</html>
```

getValue.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>getValue.jsp</title>
</head>
<body>
 <%
      String name = request.getParameter("userName");
      //userName 为 example3_1.jsp 页面中的表单参数名
      out.println(name);
 %>
</body>
</html>
```

2) String[] getParameterValues(String name)

该方法以字符串数组的形式返回客户端向服务器传递的指定参数名的所有值。

【例 3-2】 调用 getParameterValues(String name)方法获取表单信息。

example3_2.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_2.jsp</title>
</head>
<body>
 <form action="getValues.jsp">
      选择你去过的城市:<br/>
      <input type="checkbox" name="cities" value="北京"/>北京
      <input type="checkbox" name="cities" value="上海"/>上海
      <input type="checkbox" name="cities" value="西安"/>西安
      <input type="submit" value="提交"/>
 </form>
</body>
</html>
```

getValues.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>getValues.jsp</title>
</head>
<body>
 您去过的城市:<br>
 <%
      String yourCities[] = request.getParameterValues("cities");
      for(int i = 0; i < yourCities.length; i ++){
            out.println(yourCities[i] + "<br>");
      }
 %>
</body>
</html>
```

❷ NullPointerException 异常

如果不选择 example3_2.jsp 页面中的城市，直接单击“提交”按钮，那么 getValues.jsp 页面就会提示出现 NullPointerException 异常。为了避免在运行时出现 NullPointerException 异常，在 getValues.jsp 页面中使用以下代码：

```
if(yourCities != null){
      for(int i = 0;i < yourCities.length; i ++){
            out.print(yourCities[i] + "<br>");
      }
}
```

## 3.1.2 用 request 对象存取数据

request 对象可以通过 void setAttribute(String key, Object obj)方法将参数 obj 指定的对象保存到 request 对象中，key 为所保存的对象指定一个关键字。若保存的两个对象关键字相同，则先保存的对象被清除。

request 对象可以通过 Object getAttribute(String key)方法获取请求域(例如 forward 转发)中的关键字为 key 的对象(属性值)。

在实际应用中，request 对象经常用于存储、传递本次请求的处理结果。

【例 3-3】 编写两个 JSP 页面 example3_3.jsp 和 example3_3_1.jsp，在 example3_3.jsp 页面中输入一个整数提交给 example3_3_1.jsp 页面求平方。当输入值为非整数时，在 example3_3_1.jsp 页面中使用 request 对象的 setAttribute(String key, Object obj)方法将错误消息存储到 request 对象中，同时使用 forward 动作标记转发到 example3_3_1.jsp 页面并显示错误消息。

example3_3.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_3.jsp</title>
</head>
<body>
    <%
        //从请求域中获取 errorMsg 属性的错误消息
        String error = (String)request.getAttribute("errorMsg");
        if(error != null){
            out.print("<font color='red'>" + error + "</font>");
        }
     %>
    <form action="example3_3_1.jsp">
        输入一个整数求平方:
        <input type="text" name="mynumber">
        <br>
        <input type="submit" value="提交">
    </form>
</body>
</html>
```

example3_3_1.jsp 的代码如下：

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

```
<html>
<head>
<meta charset="UTF-8">
<title>example3_3_1.jsp</title>
</head>
<body>
    <%
        String number = request.getParameter("mynumber");
        int toNumber = 0;
        try{
            toNumber = Integer.parseInt(number);
            out.print(toNumber * toNumber);
        }catch(Exception e){
            request.setAttribute("errorMsg", "请输入一个整数!");
     %>
            <jsp:forward page="example3_3.jsp"></jsp:forward>
    <%
        }
     %>
</body>
</html>
```

在例 3-3 中，错误消息被以请求域属性的形式保存到 request 对象中，并通过请求转发的方式将请求对象再转发给 example3_3.jsp，在 example3_3.jsp 页面中便可以从 request 对象中获取到属性值，从而实现了错误消息在一次 request 请求范围内传递。

### ▶ 3.1.3 中文乱码问题

Java 的内核和 class 文件是基于 unicode 的，这使 Java 程序具有良好的跨平台性，但也产生了一些中文乱码问题。

如果在例 3-1 的 example3_1.jsp 页面的文本框中输入中文姓名，那么 getValue.jsp 页面获得的姓名可能是乱码。如果出现中文乱码，可以使用 request 对象的 setCharacterEncoding(String code)方法设置统一字符编码，其中参数 code 以字符串形式传入要设置的编码格式，但这种方法仅对于提交方式是 post 的表单(表单默认的提交方式是 get)有效。例如，使用该方法解决例 3-1 中的 getValue.jsp 页面出现的中文乱码问题，需要完成以下两件事。

首先将 example3_1.jsp 中的表单提交方式改为“post”，具体代码如下：

```
<form action="getValue.jsp" method="post">
```

然后在 getValue.jsp 中获取表单信息之前设置统一编码，具体代码如下：

```
request.setCharacterEncoding("UTF-8");
```

在使用该方法解决中文乱码问题时，接受参数的每个页面都需要执行 request.setCharacterEncoding("UTF-8")。为了避免每个页面都编写 request.setCharacterEncoding("UTF-8")，可以使用过滤器对所有 JSP 页面进行编码处理。过滤器将在本书的第 6 章中讲解。

### ▶ 3.1.4 实践环节——获取客户端的基本信息

编写一个 JSP 页面 practice3_1.jsp，在该页面中使用 request 的方法获取客户端的 IP 地址、客户机名称、服务器名称以及服务器端口号。

扫一扫

视频讲解

## 3.2 response 对象

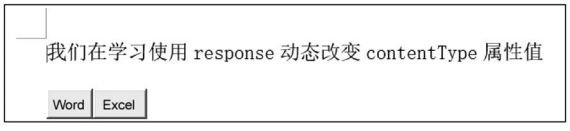
当客户端请求服务器的一个页面时会提交一个 HTTP 请求，服务器收到请求后返回 HTTP 响应。request 对象对请求信息进行封装，与 request 对象对应的对象是 response 对象。response 对象的类型为 jakarta.servlet.http.HttpServletResponse，对客户端的请求做出动态响应。动态响应通常有动态改变 contentType 属性值、设置响应表头和 response 重定向。

### 3.2.1 动态改变 contentType 属性值

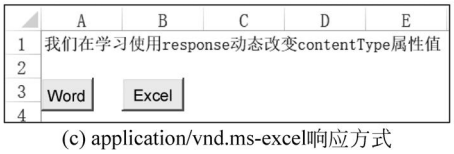
JSP 页面用 page 指令标记设置了页面的 contentType 属性值，response 对象按照此属性值的方式对客户端做出响应。在 page 指令标记中只能为 contentType 属性指定一个值。如果想动态改变 contentType 属性值，换一种方式来响应客户端，可以让 response 对象调用 setContentType(String s)方法来重新设置 contentType 的属性值。

【例 3-4】 编写一个 JSP 页面 example3_4.jsp，客户端通过单击页面上的不同按钮可以改变页面响应的 MIME 类型。当单击 Word 按钮时，JSP 页面动态改变 contentType 属性值为 application/msword，在内置的浏览器中启用本地的 Word 软件来显示当前页面内容；当单击 Excel 按钮时，JSP 页面动态改变 contentType 属性值为 application/vnd.ms-excel，浏览器启用本地的 Excel 软件来显示当前页面内容。页面效果如图 3.1 所示。


(a) text/html响应方式




(b) application/msword响应方式




(c) application/vnd.ms-excel响应方式

图 3.1 例 3-4 的效果图

example3_4.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_4.jsp</title>
</head>
```

```
<body>
  <form action="" method="post">
  <p>我们在学习使用 response 动态改变 contentType 属性值
  <p>
        <input type="submit" value="Word" name="submit">
        <input type="submit" value="Excel" name="submit">
        <%
            String str = request.getParameter("submit");
            if ("Word".equals(str)) {
                  //response 调用 setContentType()方法设置 MIME 类型为 application/msword
                  response.setContentType("application/msword");
            } else if ("Excel".equals(str)) {
                  //response 调用 setContentType()方法设置 MIME 类型为 application/vnd.ms-excel
                  response.setContentType("application/vnd.ms-excel");
            }
         %>
  </form>
</body>
</html>
```

### ▶ 3.2.2 设置响应表头

response 对象可以通过 setHeader(String name,String value)方法设置指定名字的 HTTP 文件头值,以此来操作 HTTP 文件头。如果希望某页面每 3 秒钟刷新一次,那么在该页面中添加如下代码:

```
response.setHeader("refresh","3");
```

大家有时候希望几秒钟后从当前页面自动跳转到另一个页面。比如打开 one.jsp 页面 3 秒钟后自动跳转到 another.jsp 页面(one.jsp 和 another.jsp 在同一个 Web 服务目录下)。这该如何实现呢?只需要为 one.jsp 设置一个响应头即可,也就是在 one.jsp 页面中添加如下代码:

```
response.setHeader("refresh","3;url=another.jsp");
```

【例 3-5】 编写一个 JSP 页面 example3_5.jsp,在该页面中使用 response 对象设置一个响应头“refresh”,其值是“3”。那么用户收到这个响应头后,该页面会每 3 秒钟刷新一次。

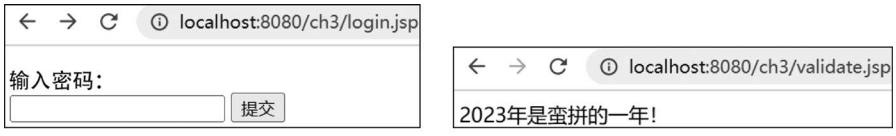
example3_5.jsp 的代码如下:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.util.*" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_5.jsp</title>
</head>
<body>
  <h2>该页面每 3 秒钟刷新一次</h2>
  <p>现在的秒钟时间是:
  <%
        Calendar c = Calendar.getInstance();
        out.print("" + c.get( Calendar.SECOND));
        response.setHeader("refresh","3");
   %>
</body>
</html>
```

### 3.2.3 response 重定向

当需要将客户端引导至另一个页面时，可以使用 response 对象的 sendRedirect(String url)方法实现客户端的重定向。例如客户端输入的表单信息不完整或有误，应该再次被重定向到输入页面。

【例 3-6】 编写两个 JSP 页面 login.jsp 和 validate.jsp，如果在 login.jsp 页面中输入正确的密码“nihao2023”，单击“提交”按钮后提交给 validate.jsp 页面，如果输入不正确，重新定向到 login.jsp 页面。先运行 login.jsp 页面，页面效果如图 3.2 所示。




(a) login.jsp页面　　(b) validate.jsp页面

图 3.2 例 3-6 的效果图

login.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>login.jsp</title>
</head>
<body>
<form action="validate.jsp" method="post" name=form>
  <p>
        输入密码：
  <br>
  <input type="password" name="pwd"/>
  <input type="submit" value="提交">
</form>
</body>
</html>
```

validate.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>validate.jsp</title>
</head>
<body>
<%
  String str = request.getParameter("pwd");
  if (!"nihao2023".equals(str)) {
        response.sendRedirect("login.jsp");
  } else {
        out.print("2023年是蛮拼的一年!");
  }
 %>
</body>
</html>
```

response 对象的 sendRedirect()方法在客户端的浏览器中工作，Web 服务器要求浏览器重新发送一个到被定向页面的请求。在浏览器的地址栏上将出现重定向页面的 URL，且为绝对路径。

forward 动作标记也可以实现页面的跳转，例如<jsp:forward page="info.jsp"/>。使用 forward 动作标记与使用 response 对象调用 sendRedirect()不同，两者的比较如下：

(1) forward 动作标记为服务器端跳转，浏览器的地址栏不变；sendRedirect()为客户端跳转，浏览器的地址栏改变为新页面的 URL。

(2) 执行到 forward 动作标记出现处停止当前 JSP 页面的继续执行，转向动作标记中 page 属性指定的页面；sendRedirect()在所有代码执行完毕之后再跳转。

(3) 使用 forward 动作标记，request 请求信息能够保留到下一个页面；使用 sendRedirect()不能保留 request 请求信息。

forward 动作标记传递参数的格式如下：

```
<jsp:forward page="info.jsp">
        <jsp:param name="no" value="001"/>
        <jsp:param name="age" value="18"/>
</jsp:forward>
```

response 对象的 sendRedirect()传递参数的格式如下：

```
response.sendRedirect("info.jsp?sno=001&sage=18");
```

### ▶ 3.2.4 实践环节—— 登录验证

编写 3 个 JSP 页面 login_1.jsp、server.jsp 和 loginSuccess.jsp。在 login_1.jsp 页面中输入用户名和密码，单击“提交”按钮将输入的信息提交给 server.jsp 页面。在 server.jsp 页面中进行登录验证：如果输入正确(用户名为“zhangsan”，密码为“123”)，提示“成功登录，3 秒钟后进入 loginSuccess.jsp 页面”，如果输入不正确，重新定向到 login_1.jsp 页面。先运行 login_1.jsp 页面，页面效果如图 3.3 所示。


(a) login_1.jsp页面


(b) server.jsp页面


(c) loginSuccess.jsp页面

图 3.3 3.2.4 实践环节的效果图

## 3.3 out 对象

out 对象的类型为 jakarta.servlet.jsp.JspWriter，它是一个输出流，用来向客户端的浏览器输出数据。out 对象的常用方法如表 3.2 所示。

表 3.2 out 对象的常用方法

| 序号 | 方 法 | 功 能 说 明 |
|---|---|---|
| 1 | void clear() | 清除缓冲区的内容 |
| 2 | void clearBuffer() | 清除缓冲区的当前内容 |
| 3 | void flush() | 清空流 |
| 4 | int getBufferSize() | 返回缓冲区的字节数，如果不设缓冲区则返回 0 |
| 5 | int getRemaining() | 返回缓冲区的剩余大小 |
| 6 | boolean isAutoFlush() | 返回缓冲区满时是自动清空还是抛出异常 |
| 7 | void close() | 关闭输出流 |
| 8 | void print() | 输出各种数据类型 |
| 9 | void newLine() | 输出一个换行符 |

【例 3-7】 编写一个页面 example3_7.jsp，在该页面中使用 out 对象输出信息。example3_7.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import="java.util.Date"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_7.jsp</title>
</head>
<body>
    <%
        int myNumber = 1000;
        Date myDate = new Date();
        out.print(myNumber);
        out.print("<br>");
        out.print(myDate);
     %>
</body>
</html>
```

扫一扫

视频讲解

## 3.4 session 对象

浏览器与 Web 服务器之间使用 HTTP 进行通信。HTTP 是一种无状态协议，客户端向服务器发出请求(request)，服务器返回响应(response)，连接就被关闭了，在服务器端不保留连接的相关信息，所以服务器必须采取某种手段来记录每个客户端的连接信息。Web 服务器可以使用内置对象 session 来存放有关连接的信息，session 对象的类型为 jakarta.servlet.http.HttpSession。session 对象指的是客户端与服务器端的一次会话，从客户端连到服务器端的一个 Web 应用程序开始，直到客户端与服务器端断开为止。

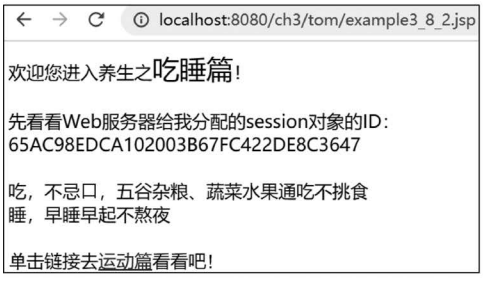
### ▶ 3.4.1 session 对象的 ID

Web 服务器会给每一个用户自动创建一个 session 对象，为每个 session 对象分配一个唯一标识的 String 类型的 session ID，这个 ID 用于区分其他用户。这样每个用户都对应着一个 session 对象，不同用户的 session 对象互不相同。session 对象调用 getId()方法可以获取当前 session 对象的 ID。

【例 3-8】 编写 3 个 JSP 页面 example3_8_1.jsp、example3_8_2.jsp 和 example3_8_3.jsp，其

中，example3_8_2.jsp 存放在目录 tom 中，example3_8_3.jsp 存放在目录 cat 中。用户首先访问 example3_8_1.jsp 页面，从该页面链接到 example3_8_2.jsp 页面，然后从 example3_8_2.jsp 页面链接到 example3_8_3.jsp，效果如图 3.4 所示。


(a) example3_8_1.jsp页面效果




(b) example3_8_2.jsp页面效果


(c) example3_8_3.jsp页面效果

图 3.4　获取 session 对象的 ID

example3_8_1.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_8_1.jsp</title>
</head>
<body>
  年轻人如何养生呢?<br><br>
  先看看 Web 服务器给我分配的 session 对象的 ID:
  <%
      String id = session.getId();
   %>
  <br>
  <%= id %>
  <br><br>
  单击链接去<a href="tom/example3_8_2.jsp">吃睡篇</a>看看吧!
</body>
</html>
```

example3_8_2.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_8_2.jsp</title>
</head>
<body>
  欢迎您进入养生之<font size=5>吃睡篇</font>!<br><br>
  先看看 Web 服务器给我分配的 session 对象的 ID:
```

```
    <%
        String id = session.getId();
    %>
    <br>
    <% = id %>
    <br><br>
    吃,不忌口,五谷杂粮、蔬菜水果通吃不挑食<br>
    睡,早睡早起不熬夜<br><br>
    单击链接去<a href = "../cat/example3_8_3.jsp">运动篇</a>看看吧!
</body>
</html>
```

example3_8_3.jsp 的代码如下：

```
<%@ page language = "java" contentType = "text/html; charset = UTF - 8" pageEncoding = "UTF - 8" %>
<!DOCTYPE html>
<html>
<head>
<meta charset = "UTF - 8">
<title> example3_8_3.jsp </title>
</head>
<body>
    欢迎您进入养生之<font size = 5 >运动篇</font>!<br><br>
    先看看 Web 服务器给我分配的 session 对象的 ID:
    <%
        String id = session.getId();
    %>
    <br>
    <% = id %>
    <br><br>
    动,坚持运动——这一点年轻人很多都做不好,<br>高兴起来就拼命打球,懒起来拼命睡觉,不好!
<br>
    总之,生活规律化,坚持长期运动<br><br>
    单击链接去<a href = "../example3_8_1.jsp">首页</a>看看吧!
</body>
</html>
```

从例 3-8 各个页面的运行结果来看，一个用户在同一个 Web 服务中只有一个 session 对象，当用户访问相同 Web 服务的其他页面时，Web 服务器不会再重新分配 session 对象，直到用户关闭浏览器或这个 session 对象结束了它的生命周期。当用户重新打开浏览器访问该 Web 服务时，Web 服务器为该用户再创建一个新的 session 对象。

需要注意的是，同一用户在多个不同的 Web 服务中所对应的 session 对象是不同的，一个 Web 服务对应一个 session 对象。

### ▶ 3.4.2 用 session 对象存取数据

使用 session 对象可以保存用户在访问某个 Web 服务期间的有关数据。有关数据处理的方法如下。

(1) public void setAttribute(String key, Object obj)：将参数 obj 指定的对象保存到 session 对象中，key 为所保存的对象指定一个关键字。若保存的两个对象的关键字相同，则先保存的对象被清除。

(2) public Object getAttribute(String key)：获取 session 中关键字是 key 的对象。

(3) public void removeAttribute(String key)：从 session 中删除关键字 key 所对应的对象。

(4) public Enumeration getAttributeNames()：产生一个枚举对象，该枚举对象可以使用

nextElements()方法遍历 session 中各个对象所对应的关键字。

【例 3-9】 使用 session 对象模拟在线考试系统。编写 3 个 JSP 页面 example3_9_1.jsp、example3_9_2.jsp 和 example3_9_3.jsp,在 example3_9_1.jsp 页面中考试,在 example3_9_2.jsp 页面中显示答题结果,在 example3_9_3.jsp 页面中计算并公布考试成绩。首先运行 example3_9_1.jsp 页面,效果如图 3.5 所示。


(a) 试卷页面


(b) 确认页面


(c) 成绩公布页面

图 3.5 使用 session 对象模拟在线考试系统

example3_9_1.jsp 的代码如下:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_9_1.jsp</title>
</head>
<body>
  <form action="example3_9_2.jsp" method="post">
            考号:
            <input type="text" name="id"/>
            <p>
                  一、单项选择题(每题 2 分)
                  <br/><br/>
                  1.下列哪个方法是获取 session 中关键字是 key 的对象(  )。
                  <br/>
                  <input type="radio" name="one" value="A"/>
                  A.public void setAttribute(String key, Object obj)<br/>
                  <input type="radio" name="one" value="B"/>
                  B.public void removeAttribute(String key)<br/>
                  <input type="radio" name="one" value="C"/>
                  C.public Enumeration getAttributeNames()<br/>
                  <input type="radio" name="one" value="D"/>
                  D.public Object getAttribute(String key)<br/>
            </p>
            <p>
                  二、判断题(每题 2 分)
                  <br/><br/>
                  1.同一用户在多个 Web 服务中,所对应的 session 对象是互不相同的。
                  <br/>
                  <input type="radio" name="two" value="True"/>
                  True
```

```
                    <input type="radio" name="two" value="False"/>
                    False
                </p><br/>
                <input type="submit" value="提交" name=submit>
                <input type="reset" value="重置" name=reset>
            </form>
</body>
</html>
```

example3_9_2.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_9_2.jsp</title>
</head>
<body>
  <form action="example3_9_3.jsp" method="post">
            <%
                    //考号
                    String id = request.getParameter("id");
                    //把考号 id 以"id"为关键字存储到 session 对象中
                    session.setAttribute("id", id);
                    //单项选择题的第 1 题
                    String first = request.getParameter("one");
                    //把答案 first 以"one"为关键字存储到 session 对象中
                    session.setAttribute("one", first);
                    //判断题的第 1 题
                    String second = request.getParameter("two");
                    //把答案 second 以"two"为关键字存储到 session 对象中
                    session.setAttribute("two", second);
             %>
            您的考号:<%=id%><br/>
            一、单项选择题(每题 2 分)
            <br/>
            1.<%=first%>
            <br/>
            二、判断题(每题 2 分)
            <br/>
            1.<%=second%><br/>
            <input type="submit" value="确认完毕"/>
            <a href="example4_9_1.jsp">重新答题</a>
        </form>
</body>
</html>
```

example3_9_3.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_9_3.jsp</title>
</head>
<body>
  <%
      //获取考号
      //获取 session 中关键字是 id 的对象(考号)
      String id = (String) session.getAttribute("id");
```

```
        //计算成绩
        int sum = 0;
        //如果单项选择题的第 1 题选中 D 选项,得 2 分
        //获取 session 中关键字是 one 的对象(选择答案)
        String first = (String) session.getAttribute("one");
        if ("D".equals(first)) {
            sum += 2;
        }
        //如果判断题的第 1 题选中 True,得 2 分
        //获取 session 中关键字是 two 的对象(判断答案)
        String second = (String) session.getAttribute("two");
        if ("True".equals(second)) {
            sum += 2;
        }
    %>
    您的成绩公布如下:
    <table border = "1">
        <tr>
            <th width = "50 %">
                考号
            </th>
            <th width = "50 %">
                成绩
            </th>
        </tr>
        <tr>
            <td><% = id %></td>
            <td align = "right"><% = sum %></td>
        </tr>
    </table>
</body>
</html>
```

### ▶ 3.4.3 session 对象的生命周期

在某个 Web 服务中 session 对象的生命周期依赖于以下几个因素：

(1) 用户是否关闭浏览器。

(2) session 对象是否调用 invalidate()方法。

(3) session 对象是否达到设置的最长“发呆”时间。

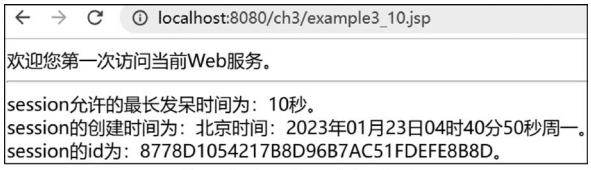
与 session 对象的生命周期相关的方法如表 3.3 所示。

表 3.3 与 session 对象的生命周期相关的方法

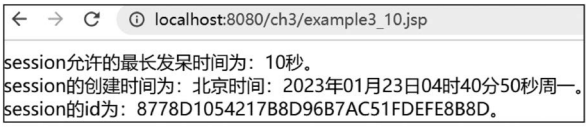
| 序号 | 方 法 | 功 能 说 明 |
|---|---|---|
| 1 | long getCreationTime() | 返回 session 的创建时间 |
| 2 | long getLastAccessedTime () | 返回此 session 中客户端最近一次请求的时间 |
| 3 | int getMaxInactiveInterval() | 返回两次请求的间隔时间(单位是秒) |
| 4 | void invalidate() | 使 session 失效 |
| 5 | boolean isNew() | 判断客户端是否已经加入服务器创建的 session |
| 6 | void setMaxInactiveInterval() | 设置两次请求的间隔时间(单位是秒) |

【例 3-10】 编写一个 JSP 页面 example3_10.jsp。如果用户是第一次访问该页面,会显示欢迎信息,并输出 session 对象允许的最长“发呆”时间、创建时间,以及 session 对象的 id。在 example3_10.jsp 页面中,session 对象使用 setMaxInactiveInterval(int maxValue)方法设置最长“发呆”时间为 10 秒。如果用户两次刷新的间隔时间超过 10 秒,用户先前的 session 会被

取消，用户将获得一个新的 session 对象。页面的运行效果如图 3.6 所示。




(a) 第一次或10秒后访问该页面




(b) 10秒之内访问该页面

**图 3.6 session 生命周期示例的运行效果**

example3_10.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%@ page import="java.util.*"%>
<%@ page import="java.text.*"%>
<html>
<head>
<meta charset="UTF-8">
<title>example3_10.jsp</title>
</head>
<body>
<%
  //session 调用 setMaxInactiveInterval(int n)方法设置最长"发呆"时间为 10 秒
  session.setMaxInactiveInterval(10);
  //session 调用 isNew()方法判断 session 是否为新创建的
        boolean flg = session.isNew();
        if (flg) {
              out.println("欢迎您第一次访问当前 Web 服务。");
              out.println("<hr/>");
        }
        out.println("session 允许的最长发呆时间为:" +
        session.getMaxInactiveInterval() + "秒。");
        //获取 session 对象被创建的时间
        long num = session.getCreationTime();
        //将整数转换为 Date 对象
        Date time = new Date(num);
        //用给定的模式和默认语言环境的日期格式符号构造 SimpleDateFormat 对象
        SimpleDateFormat matter = new SimpleDateFormat(
                    "北京时间:yyyy 年 MM 月 dd 日 HH 时 mm 分 ss 秒 E。");
        //得到格式化后的字符串
        String strTime = matter.format(time);
        out.println("<br/>session 的创建时间为:" + strTime);
        out.println("<br/>session 的 id 为:" + session.getId() + "。");
  %>
</body>
</html>
```

从例 3-10 中可以看出，如果用户长时间不关闭浏览器，session 对象也没有调用 invalidate()方法，那么用户的 session 也可能消失。例如该例中的 JSP 页面如果在 10 秒钟之内不被访问，它先前创建的 session 对象就消失了，服务器又重新创建一个 session 对象。这是因为 session 对象达到了它的最长“发呆”时间。所谓“发呆”时间，是指用户对某个 Web 服务发出的两次请

求之间的间隔时间。

用户对某个 Web 服务的 JSP 页面发出请求并得到响应，如果用户不再对该 Web 服务发出请求，比如不再操作浏览器，那么用户对该 Web 服务进入“发呆”状态，直到用户再次请求该 Web 服务时“发呆”状态结束。

Tomcat 服务器允许用户的最长“发呆”时间为 30 分钟，用户可以通过修改 Tomcat 安装目录中 conf 文件夹下的配置文件 web.xml，找到下面的片段，修改其中的默认值“30”，这样就可以重新设置各个 Web 服务目录下的 session 对象的最长“发呆”时间。这里的时间单位为分钟。

```
<session-config>
        <session-timeout>30</session-timeout>
</session-config>
```

另外，用户也可以通过 session 对象调用 setMaxInactiveInterval(int time)方法来设置最长“发呆”时间，参数的时间单位为秒。

### ▶ 3.4.4 实践环节——购物车

客户到便民超市采购商品，在购物前需要登录会员卡号，购物时先将选购的商品放入购物车，再到柜台清点商品。请借助 session 对象模拟购物车，并存储客户的会员卡号和购买的商品的名称。会员卡号在输入以后可以修改，购物车中的商品可以查看。编写程序模拟上述过程。loginID.jsp 实现会员卡号的输入，shop.jsp 实现商品导购，food.jsp 实现商品购物，count.jsp 实现清点商品。本节实践环节的 4 个 JSP 页面都保存在 practice4 目录中，先运行 loginID.jsp 页面，运行效果如图 3.7 所示。


(a) 会员卡输入页面　(b) 导购页面

(c) 购物页面　(d) 查看购物车商品页面

图 3.7 借助 session 对象模拟购物车

扫一扫

视频讲解

## 3.5 application 对象

### ▶ 3.5.1 什么是 application 对象

不同用户的 session 对象互不相同，但有时用户之间可能需要共享一个对象，在 Web 服务器启动后就产生了这样一个唯一的内置对象 application。application 对象实现了 jakarta.servlet.ServletContext 接口。任何用户在访问同一个 Web 服务的各个页面时共享一个 application 对象，直到服务器关闭这个 application 对象才被取消。

### 3.5.2 用 application 对象存取数据

application 对象和 session 对象一样也可以进行数据的存取，处理数据的方法如下。

（1）public void setAttribute(String key, Object obj)：将参数 obj 指定的对象保存到 application 对象中，key 为所保存的对象指定一个关键字。若保存的两个对象的关键字相同，则先保存的对象被清除。

（2）public Object getAttribute(String key)：获取 application 中关键字是 key 的对象。

（3）public void removeAttribute(String key)：从 application 中删除关键字 key 所对应的对象。

（4）public Enumeration getAttributeNames()：产生一个枚举对象，该枚举对象可以使用 nextElements()方法遍历 application 中各个对象所对应的关键字。

【例 3-11】 用 application 对象模拟“成语接龙”，用户通过 example3_11_1.jsp 向 example3_11_2.jsp 页面在提交四字成语 example3_11_2.jsp 页面在获取成语内容后用同步方法将该成语内容和以前的成语内容进行连接，然后将这些四字成语添加到 application 对象中。页面的运行效果如图 3.8 所示。


(a) 成语提交页面


(b) 接龙成功页面

图 3.8 成语接龙

example3_11_1.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_11_1.jsp</title>
</head>
<body>
<h2>四字成语接龙</h2>
<%
  //取出 application 中关键字是 message 的对象(成语内容)
  StringBuffer s = (StringBuffer)application.getAttribute("message");
  if(s != null){
        out.print(s.toString());
  }
  else{
        out.print("还没有词语,请您龙头开始!<br>");
  }
 %>
<form action="example3_11_2.jsp" method="post">
  四字成语输入:<input type="text" name="mes"/><br>
  <input type="submit" value="提交"/>
</form>
</body>
</html>
```

example3_11_2.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_11_2.jsp</title>
</head>
<body>
  <%!
        StringBuffer message = new StringBuffer("");
        ServletContext application;
        synchronized void sendMessage(String s){
              application = getServletContext();
              message.append(s + "->") ;
              //把成语内容 message 以"message"为关键字存储到 application 对象中
              application.setAttribute("message", message);
        }
   %>
   <%
        request.setCharacterEncoding("UTF-8");
        String content = request.getParameter("mes");
        sendMessage(content);
        out.print("您的四字成语已经提交!3 秒钟后回到成语页面,继续接龙!");
        response.setHeader("refresh", "3;url=example3_11_1.jsp");
   %>
</body>
</html>
```

例 3-11 中的成语接龙方法 sendMessage()为什么定义为同步方法？这是因为 application 对象对所有用户都是相同的，任何用户对该对象中存储的数据进行操作都会影响到其他用户。

如果客户端浏览不同的 Web 服务，将产生不同的 application 对象。同一个 Web 服务的所有 JSP 页面都共享一个 application 对象，即使浏览这些 JSP 页面的是不同的客户端也是如此。因此，保存在 application 对象中的数据不仅可以跨页面分享，还可以由所有用户共享。

有些 Web 服务器不能直接使用 application 对象，必须使用父类 ServletContext 声明这个对象，然后使用 getServletContext()方法为 application 对象进行实例化。例如例 3-11 中 example3_11_2.jsp 页面的代码。

### ▶ 3.5.3 实践环节——网站访客计数器

使用 application 对象实现网站访客计数器的功能。

## 3.6 pageContext 对象

pageContext 对象即页面上下文对象，表示当前页面运行环境，用于获取当前 JSP 页面的相关信息，它的作用范围是当前 JSP 页面。pageContext 对象的类型为 jakarta.servlet.jsp.PageContext。pageContext 对象可以访问当前 JSP 页面的所有内置对象，如表 3.4 所示。另外，pageContext 对象提供了存取属性的方法，如表 3.5 所示。

表 3.4 pageContext 对象获取内置对象的方法

| 序号 | 方 法 | 功 能 说 明 |
|---|---|---|
| 1 | ServletRequest getRequest() | 获取当前 JSP 页面的请求对象 |
| 2 | ServletResponse getResponse() | 获取当前 JSP 页面的响应对象 |
| 3 | HttpSession getSession() | 获取和当前 JSP 页面有关的会话对象 |
| 4 | ServletConfig getServletConfig() | 获取当前 JSP 页面的 ServletConfig 对象 |
| 5 | ServletContext getServletContext() | 获取当前 JSP 页面的运行环境的 application 对象 |
| 6 | Object getPage() | 获取当前 JSP 页面的 Servlet 实体的 page 对象 |
| 7 | Exception getException() | 获取当前 JSP 页面的异常对象 exception，这时此页面的 page 指令的 isErrorPage 属性要设置为 true |
| 8 | JspWriter getOut() | 获取当前 JSP 页面的输出流对象 out |

表 3.5 pageContext 对象提供的存取属性的方法

| 序号 | 方 法 | 功 能 说 明 |
|---|---|---|
| 1 | Object getAttribute(String key, int scope) | 获取范围为 scope、关键字为 key 的属性对象 |
| 2 | void setAttribute(String key,Object value,int scope) | 以“键-值”对的方式存储范围为 scope 的属性对象 |
| 3 | void removeAttribute(String key, int scope) | 从 scope 范围移除关键字为 key 的属性对象 |
| 4 | Enumeration getAttributeNamesInScope(int scope) | 从 scope 范围中获取所有属性对象对应的关键字 |

【例 3-12】 编写一个 JSP 页面 example3_12.jsp，在该页面中使用 pageContext 对象添加和获取请求域属性值。

example3_12.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <%
        //添加页面域属性值
        pageContext.setAttribute("pageKey", "页面域属性");
        //获取页面域属性值
        String pageValue = (String)pageContext.getAttribute("pageKey");
        //添加请求域属性值
        pageContext.getRequest().setAttribute("requestKey", "请求域属性");
        //获取请求域属性值
        String requestValue = (String)pageContext.getAttribute("requestKey", 2);
        out.print(pageValue + "<br>");
        out.print(requestValue);
    %>
</body>
</html>
```

## 3.7 page 对象

page 对象是一个与 Servlet 有关的内置对象，它表示 JSP 文件转译后的 Servlet 对象，代表 JSP 页面本身，即 this，因此它可以调用 Servlet 类所定义的方法。page 对象的类型为 jakarta.servlet.jsp.HttpJspPage，在实际应用中很少在 JSP 页面中使用 page 对象。

【例 3-13】 编写一个 JSP 页面 example3_13.jsp，在该页面中使用 page 指令的 info 属性设置页面的说明信息，并分别使用 this 和 page 对象获取页面的说明信息。

example3_13.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" info="page 内置对象测试" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_13.jsp</title>
</head>
<body>
    使用 this 获取本页面的说明信息:<%=this.getServletInfo() %><br>
    使用 page 对象获取本页面的说明信息:<%=((HttpJspPage)page).getServletInfo() %>
</body>
</html>
```

## 3.8 config 对象

config 对象即页面配置对象，表示当前 JSP 页面转译后的 Servlet 的 ServletConfig 对象，它存储着一些初始数据。config 对象实现了 jakarta.servlet.ServletConfig 接口。config 对象和 page 对象一样，也很少被用到。

【例 3-14】 编写一个 JSP 页面 example3_14.jsp，在该页面中使用 config 对象获取当前 JSP 页面转译后的 Servlet 对象名。

example3_14.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_14.jsp</title>
</head>
<body>
<!-- 获取当前 JSP 页面转译后的 Servlet 对象名 -->
<%=config.getServletName() %>
</body>
</html>
```

## 3.9 exception 对象

exception 对象是一个与 Error 有关的内置对象，表示 JSP 页面产生的异常。如果一个 JSP 页面需要使用此对象，必须将页面中 page 指令的 isErrorPage 属性设置为 true，否则无法编译。

【例 3-15】 编写两个 JSP 页面 example3_15.jsp 和 example3_15_1.jsp。在 example3_15.jsp 页面中使用语句“exception.printStackTrace(response.getWriter());”输出 JSP 页面产生的异常信息；在 example3_15_1.jsp 页面中产生数组越界异常，并设置该页面 page 指令的 errorPage 属性值为 example3_15.jsp。

example3_15.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" isErrorPage="true" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_15.jsp</title>
</head>
<body>
    <%
        exception.printStackTrace(response.getWriter());
     %>
</body>
</html>
```

example3_15_1.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" errorPage="example4_15.jsp" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_15_1.jsp</title>
</head>
<body>
<%
    int a[] = {1,2,3,4,5};
    for(int i = 0; i<= 5; i++){
          out.print(a[i]);
    }
 %>
</body>
</html>
```

运行 example3_15_1.jsp 页面，效果如图 3.9 所示。

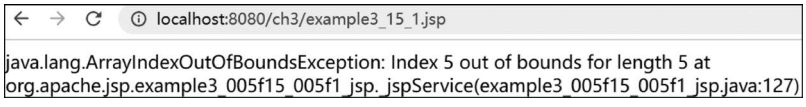



图 3.9 example3_15_1.jsp 页面的运行效果

## 3.10 JSP 的 4 种作用域

对象的作用域就是对象的生命周期和可访问性，在 JSP 中有 4 种作用域，即页面域、请求域、会话域和应用域。

❶ 页面域

页面域(page scope)的生命周期为页面执行期间。存储在页面域中的对象只能在它所在的页面被访问。

❷ 请求域

请求域(request scope)的生命周期为一次请求过程，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。

❸ 会话域

会话域(session scope)的生命周期为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结束会话。存储在会话域中的对象在整个会话期间(可以包含多次请求)都可以被访问。

❹ 应用域

应用域(application scope)的生命周期为从服务器开始执行服务到服务器关闭为止,是4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。

JSP的4种作用域分别对应pageContext、request、session和application 4个内置对象,这4个内置对象都可以通过setAttribute(String key, Object value)方法存储数据,通过getAttribute(String key)方法获取数据。

【例3-16】 编写一个JSP页面example3_16.jsp,在该页面中使用pageContext、session和application对象分别实现页面域、会话域和应用域的页面访问统计情况。

example3_16.jsp的代码如下:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>example3_16.jsp</title>
</head>
<body>
    <%
        int pageSum = 1;
        int sessionSum = 1;
        int applicationSum = 1;
        //页面域的计数
        if(pageContext.getAttribute("pageCount") != null){
              pageSum = Integer.parseInt(pageContext.getAttribute("pageCount").toString());
              pageSum++;
        }
        pageContext.setAttribute("pageCount", pageSum);
        //会话域的计数
        if(session.getAttribute("sessionCount") != null){
              sessionSum = Integer.parseInt(session.getAttribute("sessionCount").toString());
              sessionSum++;
        }
        session.setAttribute("sessionCount", sessionSum);
        //应用域的计数
        if(application.getAttribute("applicationCount") != null){
              applicationSum = Integer.parseInt(application.getAttribute("applicationCount").
toString());
              applicationSum++;
        }
        application.setAttribute("applicationCount", applicationSum);
    %>
        <p>页面域访问统计:<%=pageSum%></p>
        <p>会话域访问统计:<%=sessionSum%></p>
        <p>应用域访问统计:<%=applicationSum%></p>
</body>
</html>
```

第一次访问example3_16.jsp页面,运行效果如图3.10所示。

多次刷新浏览器窗口，运行效果如图 3.11 所示。

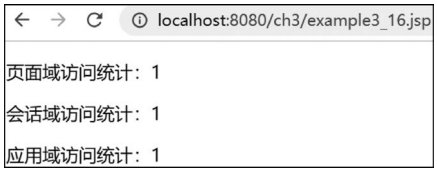



图 3.10 第一次访问 example3_16.jsp 页面

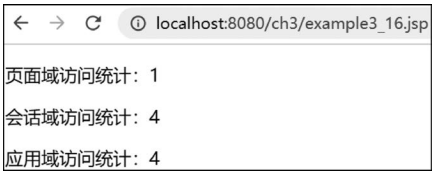



图 3.11 多次刷新 example3_16.jsp 页面

打开另一个浏览器窗口，再次访问 example3_16.jsp 页面，运行效果如图 3.12 所示。


图 3.12 打开新的浏览器窗口访问 example3_16.jsp 页面

从图 3.10～图 3.12 的运行效果可以看出，pageContext 域的访问范围为当前 JSP 页面，因此访问计数始终为 1；session 域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，但新打开浏览器窗口时会新建一个会话，计数又从 1 开始；application 域的访问范围为整个应用，因此只要服务器不停止运行，计数会不断累加。

## 3.11 本章小结

本章重点介绍了 request、session 和 application 对象，request、session 和 application 对象的范围是逐个增加的：request 只在一个请求范围内；session 在客户端与服务器会话范围内；application 则在整个服务器的运行过程中。

## 习 题 3

扫一扫

习题

扫一扫

自测题