

Java web简介

2018年4月8日 13:55

11.修改Tomcat服务器默认端口号

JavaWeb简介

修改Tomcat服务器默认端口

修改conf/server.xml文件

```
<Connector port="8080"
    protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443 "
/>>
```

JavaWeb简介

修改Tomcat服务器默认端口

修改conf/server.xml文件

```
<Connector port="8080" → 8888
    protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443 "
/>>
```

单选练习题

修改Tomcat服务器默认端口号需要修改的配置文件是()

A server.xml

B web.xml

C content.xml

D Tomcat-users.xml

Jsp基础语法

- 1.Jsp简介
- 2.常用动态网站开发技术
- 3.Jsp页面元素构成
- 4.Jsp的生命周期
- 5.阶段项目

1.

Jsp基础语法

JSP简介

JSP全名为Java Server Pages，其根本是一个简化的Servlet设计，他实现了在Java当中使用HTML标签。Jsp是一种动态网页技术标准也是JAVAEE的标准。JSP与Servlet一样，是在服务器端执行的。



- 2.常见动态网站开发技术对比
- 三种最常见的动态网站开发技术的对比

Jsp基础语法

常见动态网站开发技术对比

Jsp: Java平台，安全性高，适合开发大型的，企业级的Web应用程序。

Asp.net: .Net平台，简单易学。但是安全性以及跨平台性差。

Php: 简单，高效，成本低开发周期短，特别适合中小型企业的Web应用开发。（LAMP：Linux+Apache+MySQL+PHP）

JAVA EE 的例子

Jsp基础语法



网上证券交易所

电子银行

4.JSP页面元素简介及page指令

Jsp基础语法

Jsp页面元素构成



Jsp基础语法

Jsp指令

page指令

通常位于jsp页面的顶端，同一个页面可以有多个page指令。

include指令

将一个外部文件嵌入到当前 JSP 文件中，同时解析这个页面中的 JSP 语句。

taglib指令

使用标签库定义新的自定义标签，在JSP页面中启用定制行为

Jsp基础语法

Jsp指令

page指令语法：

```
<%@ page 属性1="属性值" 属性2="属性值1,属性值2"...
    属性n="属性值n"%>
```

属性	描述	默认值
language	指定JSP页面使用的脚本语言	java
import	通过该属性来引用脚本语言中使用到的类文件	无
contentType	用来指定JSP页面所采用的编码方式	text/html, ISO-8859-1

contentType	用来指定JSP页面所采用的编码方式	text/html, ISO-8859-1
--------------------	-------------------	--------------------------

```

1  %@ page language="java" contentType="text/html; charset=ISO-8859-1"
2      pageEncoding="ISO-8859-1"%>
3  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www
4@<html>
5@<head>
6  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7  <title>Insert title here</title>
8  </head>
9@<body>
10 </body>
11 </html>
```

页面中不能出现中文，除非使用国际化的utf-8

Jsp页面元素

Jsp注释

在JSP页面的注释。

HTML的注释：

<!-- html注释--> //客户端可见

JSP的注释：

<%-- html注释--%> //客户端不可见

JSP脚本注释：

//单行注释

/*多行注释*/

```

, ...
<body>
    <h1>关于JSP的注释</h1>
    <hr>
    <!-- HTML注释 -->
    <%--HTML注释客户端不可见 --%>
    <%
        //单行注释
        /*多行注释*/
    %>>
```

```

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.
<html>
<head>
<meta http-equiv="Content-Type" content="t
<title>我的个人主页</title>
</head>
<body>
    <h1>关于JSP的注释</h1>
    <hr>
    <!-- HTML注释 -->
</body>
</html>
```

7.jsp脚本

Jsp基础语法

Jsp脚本

在JSP页面中执行的java代码。

语法：

```
<% Java代码%>
```

```
<body>
<h1>我的第一个JSP web程序</h1>
<hr>
<!-- HTML注释 -->
<%--HTML注释客户端不可见 --%>
<%
    out.println("欢迎大家学习JAVAEE开发。");
%
</body>
```

写在那对橙色的代码框里面的就是Jsp脚本

我的第一个JSP web程序

欢迎大家学习JAVAEE开发。

8.Jsp的声明元素

Jsp基础语法

Jsp声明

在JSP页面中定义变量或者方法。

语法：

```
<%! Java代码%>
```

9.JSP表达式

Jsp基础语法

Jsp表达式

在JSP页面中执行的表达式。

语法：

```
<% =表达式 %> //注意： 表达式不以分号结束
```

```

<%!
String s="张三"; //声明了一个字符串变量
int add(int x,int y){ //声明一个返回整型的函数，实现两个整数的求和。
    return x+y;
}
%>
<br>
你好，<%=s %><br>
x+y=<%=add(10,5) %><br>

```

我的第一个JSP web程序

欢迎大家学习JAVAEE开发。

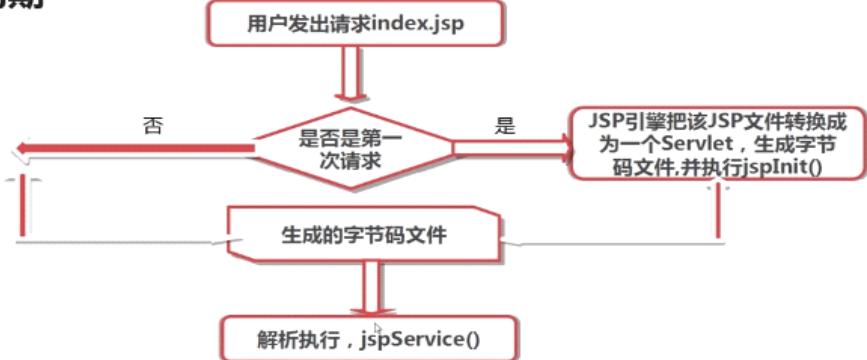
你好，张三

x+y=15

10 JSP页面生命周期

Jsp基础语法

JSP页面生命周期



Jsp基础语法

Jsp生命周期

jspService()方法被调用来处理客户端的请求。对每一个请求，JSP引擎创建一个新的线程来处理该请求。如果有多个客户端同时请求该JSP文件，则JSP引擎会创建多个线程。每个客户端请求对应一个线程。以多线程方式执行可以大大降低对系统的资源需求，提高系统的并发量及响应时间。但也要注意多线程的编程带来的同步问题，由于该Servlet始终驻于内存，所以响应是非常快的。

单选练习题

当用户第一次请求一个jsp页面时，首先被执行的方法是（ ）

① 你的答案是B，不正确！

解析

第一次请求一个jsp页面时，首先被执行的方法是构造方法

[看正确答案](#)

Jsp基础语法

阶段项目

分别使用表达式和脚本实现打印九九乘法表。

九九乘法表

1*1=1									
2*1=2	2*2=4								
3*1=3	3*2=6	3*3=9							
4*1=4	4*2=8	4*3=12	4*4=16						
5*1=5	5*2=10	5*3=15	5*4=20	5*5=25					
6*1=6	6*2=12	6*3=18	6*4=24	6*5=30	6*6=36				
7*1=7	7*2=14	7*3=21	7*4=28	7*5=35	7*6=42	7*7=49			
8*1=8	8*2=16	8*3=24	8*4=32	8*5=40	8*6=48	8*7=56	8*8=64		
9*1=9	9*2=18	9*3=27	9*4=36	9*5=45	9*6=54	9*7=63	9*8=72	9*9=81	

na1//EN">

pageEncoding 是 jsp 文件本身的编码

contentType 的 charset 是指服务器发给客户端时的内容编码

contentType 更常用

">

```
</head>
<body>
<%!
//返回九九乘法表对应的HTML代码，通过表达式调用，在页面上显示
String printMultiTable1(){
    String s="";
    for(int i=1;i<=9;i++){
        for(int j=1;j<=i;j++){
            s+=i+"*"+j+"="+ (i*j)+" &nbsp;&nbsp";
        }
        s+="<br>";//追加换行标签
    }
    return s;
}

void printMultipleTable2(JspWriter out) throws Exception{
    for(int i=1;i<=9;i++){
        for(int j=1;j<=i;j++){
            out.println(i+"*"+j+"="+ (i*j)+" &nbsp;&nbsp");
        }
        out.println("<br>");//追加换行标签
    }
}
%>
<h1>九九乘法表</h1>
<hr>
<%=printMultiTable1() %>
<br>
<%printMultipleTable2(out); %>
</body>
</html>
```

九九乘法表

1*1=1
2*1=2 2*2=4
3*1=3 3*2=6 3*3=9
4*1=4 4*2=8 4*3=12 4*4=16
5*1=5 5*2=10 5*3=15 5*4=20 5*5=25
6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81

1*1=1
2*1=2 2*2=4
3*1=3 3*2=6 3*3=9
4*1=4 4*2=8 4*3=12 4*4=16
5*1=5 5*2=10 5*3=15 5*4=20 5*5=25
6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81

第三章 JSP内置对象（上）

2018年4月8日 16:52

1.web应用程序都是基于请求和响应模式的

JSP内置对象

Web程序的请求响应模式

用户发送请求(request)

服务器给用户响应(response)



JSP内置对象

Web程序的请求响应模式



点击登录向服务器发送请求，在
请求对象中封装了用户名和密码



服务器端接收用户的请求，判断用
户名和密码后，给客户端发送响应
页面

JSP内置对象

1. 内置对象简介
2. 四种作用域范围
3. out
4. request/response
5. session
6. application
7. 其它内置对象
8. 项目案例

2.JSP内置对象简介

JSP内置对象

内置对象简介

JSP内置对象是 Web 容器创建的一组对象，不使用new关键字就可以使用的内置对象。

```
<%
int[ ] value = { 60, 70, 80 };
for (int i : value) {
    out.println(i);
}
%>
```

```
<%
int[ ] value = { 60, 70, 80 };
for (int i : value) {
    out.println(i);
}
%>
```

JSP内置对象

内置对象简介



其中out、request、response、session、application是比较常用的

- 什么是缓冲区

缓冲区：Buffer, 所谓缓冲

区就是内存的一块区域用来
保存临时数据。

• 什么是缓冲区

缓冲区：Buffer, 所谓缓冲区就是内存的一块区域用来保存临时数据。



刚煮好的一锅米饭
一粒一粒吃不知道
要吃到猴年马月。。。

IO输出最原始的的就是一个字节一个字节输出，
就像一粒一粒吃一样，但效率太差。

JSP内置对象

• 什么是缓冲区

缓冲区：Buffer, 所谓缓冲区就是内存的一块区域用来保存临时数据。

当然，也可以把勺子当作容器，将米在不同的容器中传递，最后吃掉。

勺子也可看作缓冲区，多个字节在不同的缓冲区传递。



刚煮好的一锅米饭
一粒一粒吃不知道
要吃到猴年马月。。。



把米饭放到碗里，
一碗一碗吃岂不痛快！

缓冲区

3.JSP的四种属性范围

在JSP中定义了四种属性的保存范围。所谓属性的保存范围，指的是一个内置对象可以在多少个页面中保存并继续使用，4种属性范围分别是

(1) page (用pageContext表示)：只在一个页面中保存属性，跳转之后无效 (2) request：只在一次请求中保存属性，服务器跳转之后无效 (3) session：在一次对话范围内保存，无论何种跳转都可以使用，但是新开的浏览器中不能使用 (4) application：在整个服务器上保存，所有用户都可以使用

4.OUT内置对象

JSP内置对象

out对象：

out对象是JspWriter类的实例,是向客户端输出内容常用的对象。

常用方法如下：

- 1、void println() 向客户端打印字符串
- 2、void clear() 清除缓冲区的内容，如果在flush之后调用会抛出异常
- 3、void clearBuffer();清除缓冲区的内容，如果在flush之后调用不会抛出异常
- 4、void flush() 将缓冲区内容输出到客户端
- 5、int getBufferSize() 返回缓冲区以字节数的大小，如不设缓冲区则为0
- 6、int getRemaining() 返回缓冲区还剩余多少可用
- 7、boolean isAutoFlush() 返回缓冲区满时，是自动清空还是抛出异常
- 8、void close() 关闭输出流

实战

```
<body>
    <h1>out内置对象</h1>
    <%
        out.println("<h2>静夜思</h2>");
        out.println("床前明月光<br>");
        out.println("疑是地上霜<br>");
        out.println("举头望明月<br>");
        out.println("低头思故乡<br>");
    %>
    缓冲区大小: <%=out.getBufferSize() %>byte<br>
    缓冲区剩余:<%=out.getRemaining() %>byte<br>
    缓冲区是否已满:<%=out.isAutoFlush() %><br>
</body>
</html>
```

out内置对象

静夜思

床前明月光
疑是地上霜
举头望明月
低头思故乡
缓冲区大小: 8192byte
缓冲区剩余:7859byte
缓冲区是否已满:true

验证flush()：输出不会有任何不同，但是会占用缓冲区

```
<body>
    <h1>out内置对象</h1>
    <%
        out.println("<h2>静夜思</h2>");
        out.println("床前明月光<br>");
        out.println("疑是地上霜<br>");
        out.flush();
        out.println("举头望明月<br>");
        out.println("低头思故乡<br>");
    %>
    缓冲区大小: <%=out.getBufferSize() %>byte<br>
    缓冲区剩余:<%=out.getRemaining() %>byte<br>
    缓冲区是否已满:<%=out.isAutoFlush() %><br>
</body>
```

out内置对象

静夜思

床前明月光
疑是地上霜
举头望明月
低头思故乡
缓冲区大小: 8192byte
缓冲区剩余:8140byte
缓冲区是否已满:true

验证clear()方法如果用在flush()方法后面会抛出异常，但是clearBuffer()方法用在其后面就不会抛出异常

```
</head>
<body>
    <h1>out内置对象</h1>
    <%
        out.println("<h2>静夜思</h2>");
        out.println("床前明月光<br>");
        out.println("疑是地上霜<br>");
        out.flush();
        out.clear();
        out.println("举头望明月<br>");
        out.println("低头思故乡<br>");
    %>
    缓冲区大小: <%=out.getBufferSize() %>byte<br>
    缓冲区剩余:<%=out.getRemaining() %>byte<br>
    缓冲区是否已满:<%=out.isAutoFlush() %><br>
</body>
</html>
```

out内置对象

静夜思

床前明月光
疑是地上霜

```
at java.lang.Thread.run()
Caused by: java.io.IOException: Error! Attempt to clear a buffer that's already
at org.apache.jasper.runtime.JspWriterImpl.clear(JspWriterImpl.java:152)
at org.apache.jsp.out_jsp._jspService(out_jsp.java:97)
at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:731)
at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapp...
... 23 more
```

```
<body>
<h1>out内置对象</h1>
<%
out.println("<h2>静夜思</h2>");
out.println("床前明月光<br>");
out.println("疑是地上霜<br>");
out.flush();
//out.clear();
out.clearBuffer();
out.println("举头望明月<br>");
out.println("低头思故乡<br>");
%>
缓冲区大小: <%=out.getBufferSize() %>byte<br>
缓冲区剩余:<%=out.getRemaining() %>byte<br>
缓冲区是否已满:<%=out.isAutoFlush() %><br>
</body>
</html>
```

out内置对象

静夜思

床前明月光
疑是地上霜
举头望明月
低头思故乡
缓冲区大小: 8192byte
缓冲区剩余:8140byte
缓冲区是否已满:true

单选练习题

执行以下jsp脚本输出效果是()

```
<%  
out.println("床前明月光");  
out.flush();  
out.clear();  
out.println("疑是地上霜");  
%>
```

 你的答案是A，答对了！

解析

在浏览器中输出“床前明月光”，控制台会输出异常信息。

5.request内置对象

get与post区别

```
<form name="regForm" action="动作" method="提交方式">  
</form>
```

表单有两种提交方式：get与post

- 1、get: 以明文的方式通过URL提交数据，数据在URL中可以看到。提交的数据最多不超过2KB。安全性较低但效率比post方式高。适合提交数据量不大，安全性不高的数据。比如：搜索、查询等功能。
- 2、post: 将用户提交的信息封装在HTML HEADER内。适合提交数据量大，安全性高的用户信息。比如：注册、修改、上传等功能。

Get方法：登录的用户名和密码会显示在地址栏中，非常不安全

```
<%@ page language="java" contentType="text/html; charset=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>
    <h1>用户登录</h1>
    <hr>
    <form action="dologin.jsp" name="LoginForm" method="get">
        <table>
            <tr>
                <td>用户名:</td>
                <td><input type="text" name="username"/></td>
            </tr>
            <tr>
                <td>密码:</td>
                <td><input type="password" name="password"/></td>
            </tr>
            <tr>
                <td colspan="2"><input type="submit" value="登录"></td>
            </tr>
        </table>
    </form>>
</body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=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>
    <h1>登录成功</h1>
    <hr>
</body>
</html>
```



登录成功

利用post方法，地址栏中没有显示用户名和密码



登录成功

5.request对象

request对象

客户端的请求信息被封装在request对象中，通过它才能了解到客户的需求，然后做出响应。它是HttpServletRequest类的实例。request对象具有请求域，即完成客户端的请求之前，该对象一直有效。常用方法如下：

- String getParameter(String name) 返回name指定参数的参数值
- String[] getParameterValues(String name) 返回包含参数name的所有值的数组
- void setAttribute(String, Object) 存储此请求中的属性。
- Object getAttribute(String name) 返回指定属性的属性值
- String getContentType() 得到请求体的MIME类型
- String getProtocol() 返回请求用的协议类型及版本号
- String getServerName() 返回接受请求的服务器主机名

request对象

- int getServerPort() 返回服务器接受此请求所用的端口号
- String getCharacterEncoding() 返回字符编码方式
- void setCharacterEncoding() 设置请求的字符编码方式；
- int getContentLength() 返回请求体的长度（以字节数）
- String getRemoteAddr() 返回发送此请求的客户端IP地址
- String getRealPath(String path) 返回一虚拟路径的真实路径
- String request.getContextPath() 返回上下文路径

实现一个用户注册的注册表单

```
<meta http-equiv="expires" content="0" />
<meta http-equiv="keywords" content="keyword1, keyword2, keyword3">
<meta http-equiv="description" content="This is my page">
<!--
<link rel="stylesheet" type="text/css" href="styles.css">
-->
</head>

<body>
    <h1>用户注册</h1>
    <hr>
    <form name="regForm" action="request.jsp" method="post">
        <table>
            <tr>
                <td>用户名: </td>
                <td><input type="text" name="username"/></td>
            </tr>
            <tr>
                <td>爱好: </td>
                <td>
                    <input type="checkbox" name="favorite" value="read">读书
                    <input type="checkbox" name="favorite" value="read">读书
                    <input type="checkbox" name="favorite" value="read">读书
                </td>
            </tr>
        </table>
    </form>
</body>
```

解决中文乱码问题

```

<body>
    <h1>request内置对象</h1>
    <%>
        request.setCharacterEncoding("utf-8"); //解决中文乱码问题
    %>
    用户名: <%=request.getParameter("username") %><br>
    爱好 : <%
        String[] favorites = request.getParameterValues("favorite");
    %>
    </body>
</html>

```

创建超链接 :

```

</table>
</form>
<br>
<br>
<a href="request.jsp?username=lisi">测试URL传参数</a>
</body>
</html>

```

解决抛出异常问题

```

    request.setCharacterEncoding("utf-8"); //解决中文乱码问题
    %>
    用户名: <%=request.getParameter("username") %><br>
    爱好 : <%
        if(request.getParameterValues("favorite")!=null)
        {
            String[] favorites = request.getParameterValues("favorite");
            for(int i=0;i<favorites.length;i++)
            {
                out.println(favorites[i]+" &ampnbsp");
            }
        }
    %>
</body>
</html>

```

解决URL传递中文时遇到的乱码问题需要改变server.xml文件中的参数，改变了之后tomcat服务器必须要重启

```

Define a non-SSL HTTP/1.1 Connector on port 8080
-->
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" URIEncoding="utf-8"/>
<!-- A "Connector" using the shared thread pool-->

```

```

<body>
    <h1>用户注册</h1>
    <hr>
    <form name="regForm" action="request.jsp" method="post">
        <table>
            <tr>
                <td>用户名: </td>
                <td><input type="text" name="username"></td>
            </tr>
            <tr>
                <td>爱好: </td>
                <td>
                    <input type="checkbox" name="favorite" value="read" >读书
                    <input type="checkbox" name="favorite" value="music" >音乐
                    <input type="checkbox" name="favorite" value="run" >跑步
                    <input type="checkbox" name="favorite" value="sport" >运动
                </td>
            </tr>
            <tr>
                <td colspan="2" ><input type="submit" value="提交"></td>
            </tr>
        </table>
    </form>
    <br>
    <br>
    <a href="request.jsp?username=lisi">测试URL传递参数</a>
</body>
</html>

```

```

<body>
    <h1>request内置对象</h1>
    <hr>
    <%//解决中文乱码问题,这是解决中文乱码问题的最简单的方法
    request.setCharacterEncoding("utf-8");
    request.setAttribute("password", "123456");
    %>
    用户名: <%=request.getParameter("username") %><br>
    爱好: <% //返回字符串数组
    if(request.getParameterValues("favorite")!=null){
        String[] favorites=request.getParameterValues("favorite");
        for(int i=0;i<favorites.length;i++){
            out.println(favorites[i]+" &ampnbsp");
        }
    }
    %><br>
    密码: <%=request.getAttribute("password") %><br>
    请求体的MIME类型:<%=request.getContentType() %><br>
    协议类型及版本号: <%=request.getProtocol() %><br>
    服务器主机名: <%=request.getServerName() %><br>
    服务器端口号: <%=request.getServerPort() %><br>
    请求文件的长度:<%=request.getContentLength() %><br>
    发送此请求的客户端IP地址: <%=request.getRemoteAddr() %><br>
    返回一虚拟路径的真实路径:<%=request.getRealPath("request.jsp") %><br>
    返回上下文路径: <%=request.getContextPath() %><br>

```

request内置对象

用户名：张三
爱好：read music run
密码：123456
请求体的MIME类型：application/x-www-form-urlencoded
协议类型及版本号：HTTP/1.1
服务器主机名：localhost
服务器端口号：8080
请求文件的长度：69
发送此请求的客户端IP地址：127.0.0.1
返回一虚拟路径的真实路径：C:\Users\Administrator\eclipse-workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\MyfirstJSP\request.jsp
返回上下文路径：/MyfirstJSP

6.response内置对象

(虽然重要，但是JSP中很少直接用到)

因为研究的是response对象，所以简化不用考虑任何html标签

response对象

response对象包含了响应客户请求的有关信息，但在JSP中很少直接用到它。它是HttpServletResponse类的实例。response对象具有页面作用域，即访问一个页面时，该页面内的response对象只能对这次访问有效，其它页面的response对象对当前页面无效。常用方法如下：

- String getCharacterEncoding() 返回响应应用的是何种字符编码
- void setContentType(String type) 设置响应的MIME类型
- PrintWriter getWriter() 返回可以向客户端输出字符的一个对象（注意比较：
PrintWriter与内置out对象的区别）
- sendRedirect(java.lang.String location) 重新定向客户端的请求

(1) response对象的输出流PrintWriter打印出来的对象的输出是优先于内置对象的

```
<%@ page language="java" import="java.util.* , java.io.*" contentType="text/html; charset=utf-8"%>
<%
    response.setContentType("text/html; charset=utf-8"); //设置响应的MIME类型
    out.println("<h1>response内置对象</h1>");
    out.println("<hr>"); //设置换行标签
    PrintWriter outer=response.getWriter(); //获得输出流对象
    outer.println("大家好，我是response对象生成的输出流outer对象");
    //response.sendRedirect("reg.jsp");
%>
```

大家好，我是response对象生成的输出流outer对象

response内置对象

(2) 为了使标题出现在输出流对象的前面，可以调用out内置对象的flush()方法

```
<%@ page language="java" import="java.util.* ,java.io.*" contentType="text/html; charset=utf-8"%>
<%
    response.setContentType("text/html; charset=utf-8");//设置响应的MIMI类型
    out.println("<h1>response内置对象</h1>");
    out.println("<hr>");//设置换行标签
    out.flush();
    PrintWriter outer=response.getWriter();//获得输出流对象
    outer.println("大家好，我是response对象生成的输出流outer对象");
    //response.sendRedirect("reg.jsp");
%>
```

response内置对象

大家好，我是response对象生成的输出流outer对象

```
<%@ page language="java" import="java.util.* ,java.io.*" contentType="text/html; charset=utf-8"%>
<%
    response.setContentType("text/html; charset=utf-8");//设置响应的MIMI类型
    out.println("<h1>response内置对象</h1>");
    out.println("<hr>");//设置换行标签
    //out.flush();
    PrintWriter outer=response.getWriter();//获得输出流对象
    outer.println("大家好，我是response对象生成的输出流outer对象");
    response.sendRedirect("reg.jsp");
%>
```

(3) 直接跳转到用户注册的页面

用户注册

用户名：

爱好： 读书 音乐 跑步 运动

[测试URL传递参数](#)

.请求重定向与请求转发的区别（这个问题非常重要，经常是面试的考点）

请求转发与请求重定向

请求重定向：客户端行为，`response.sendRedirect()`，从本质上讲等同于两次请求，前一次的请求对象不会保存，地址栏的URL地址会改变。

请求转发：服务器行为，`request.getRequestDispatcher().forward(req,resp);`是一次请求，转发后请求对象会保存，地址栏的URL地址不会改变。





假设你去办理某个护照

重定向：你先去了A局，A局的人说：“这个事不归我们管，去B局”，然后，你就从A退了出来，自己乘车去了B局。

转发：你去了A局，A局看了以后，知道这个事情其实应该B局来管，但是他没有把你退回来，而是让你坐一会儿，自己到后面办公室联系了B的人，让他们办好后，送了过来。

(1) 首先测试请求重定向 ,

```
request.jsp    reg.jsp    response.jsp    Insert title here
4● <head>
5  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
6  <title>Insert title here</title>
7  </head>
8● <body>
9      <h1>用户注册</h1>
10     <hr>
11●   <form name="regForm" action="response.jsp" method="post">
12●     <table>
13●       <tr>
```

```
request.jsp    reg.jsp    response.jsp    Insert title here
1  <%@ page language="java" import="java.util.* , java.io.*" %>
2● <%
3  response.setContentType("text/html; charset=utf-8"); //设置响应头
4  out.println("<h1>response内置对象</h1>");
5  out.println("<hr>"); //设置换行标签
6  //out.flush();
7  PrintWriter outer=response.getWriter(); //获得输出流对象
8  outer.println("大家好，我是response对象生成的输出流outer对象");
9  //response.sendRedirect("reg.jsp");
10 response.sendRedirect("request.jsp");
11 %>
12
```



request内置对象

用户名: null

爱好:

密码: 123456

请求体的MIME类型: null

协议类型及版本号: HTTP/1.1

服务器主机名: localhost

服务器端口号: 8080

请求文件的长度: -1

发送此请求的客户端IP地址: 127.0.0.1

返回一虚拟路径的真实路径: C:\Users\Administrator\eclipse-

workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\MyfirstJSP\

返回上下文路径: /MyfirstJSP

运行reg.jsp本来地址栏中应该是response.jsp，但是由于后者实现的请求重定向方法，所以显示的是request.jsp

(2)

```
<%@ page language="java" import="java.util.* , java.io.*" contentType="text/html; charset=utf-8" %>
<%
    response.setContentType("text/html; charset=utf-8"); // 设置响应的MIME类型
    out.println("<h1>response内置对象</h1>");
    out.println("<hr>"); // 设置换行标签
    //out.flush();
    PrintWriter outer=response.getWriter(); // 获得输出流对象
    outer.println("大家好，我是response对象生成的输出流outer对象");
    //response.sendRedirect("reg.jsp");
    //实现请求重定向
    //response.sendRedirect("request.jsp");
    //请求转发，并向后传递
    request.getRequestDispatcher("request.jsp").forward(request, response);
%>
```



request内置对象

用户名: 张三

爱好: read

密码: 123456

请求体的MIME类型: application/x-www-form-urlencoded

协议类型及版本号: HTTP/1.1

服务器主机名: localhost

服务器端口号: 8080

请求文件的长度: 41

发送此请求的客户端IP地址: 127.0.0.1

返回一虚拟路径的真实路径: C:\Users\Administrator\eclipse-

workspace\metadata\plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\MyfirstJSP\request.jsp

返回上下文路径: /MyfirstJSP

单选练习题

关于请求重定向与请求转发的区别以下说法不正确的是()

A 请求重定向相当于两次请求，地址栏地址会发生变化

B 重定向是response对象的方法而请求转发是request对象的方法

C 请求重定向是服务器端行为而请求转发是客户端行为

D 请求重定向不会保存原有request对象而请求转发会保存原有request对象

请求重定向是客户端行为而请求转发是服务器端行为

response跳转属于客户端跳转，response.sendRedirect()跳转后，地址栏的页面地址改变了

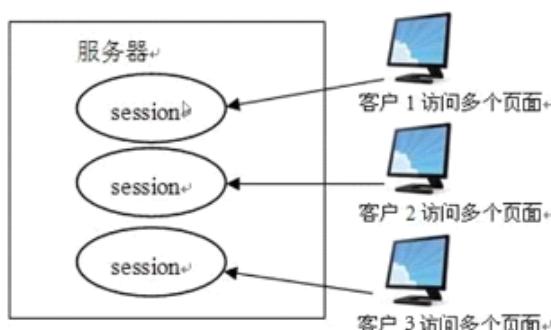
答案应该选择C

7.session内置对象



• 什么是Session

在服务器的内存中保存着不同用户的session。



session是保存在服务器的内存当中，和用户是一一对应的

session对象

- session对象是一个JSP内置对象。
- session对象在第一个JSP页面被装载时自动创建，完成会话期管理。
- 从一个客户打开浏览器并连接到服务器开始，到客户关闭浏览器离开这个服务器结束，被称为一个会话。
- 当一个客户访问一个服务器时，可能会在服务器的几个页面之间切换，服务器应当通过某种办法知道这是一个客户，就需要session对象。
- session对象是HttpSession类的实例。

session是保存用户的状态

session对象常用方法如下：

- `long getCreationTime()`：返回SESSION创建时间
- `public String getId()`：返回SESSION创建时JSP引擎为它设的唯一ID号
- `public Object setAttribute(String name, Object value)`：使用指定名称将对象绑定到此会话
- `public Object getAttribute(String name)`：返回与此会话中的指定名称绑定在一起的对象，如果没有对象绑定在该名称下，则返回null
- `String[] getValueNames()`：返回一个包含此SESSION种所有可用属性的数组
- `int getMaxInactiveInterval()`：返回两次请求间隔多长时间此SESSION被取消（单位秒）

最后一个如果停留超过这个时间说明开始的是一个新的对话

```
</head>
<body>
    <h1>session内置对象</h1>
    <hr>
    <%
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy年MM月dd日 HH:MM:ss");
        Date d=new Date(session.getCreationTime());
        session.setAttribute("username","admin");
        session.setAttribute("password", "123456");
        session.setAttribute("sex", "women");
        //设置在页面中最长的停留时间，单位是秒
        //session.setMaxInactiveInterval(10);
    %>
    session创建时JSP引擎设置的ID编号:<%=session.getId()%><br>
    session的指定名称绑在一起的对象:<%=session.getAttribute("username")%><br>
    session中获取的属性有:<%
        String[] names=session.getValueNames();
        for(int i=0;i<names.length;i++){
            out.println(names[i]+"&nbsp;&nbsp");
        }
    %><br>

    <a href="session2.jsp" target="_blank">跳转到session2.jsp</a><br>
</body>
</html>
```

session内置对象

session创建时JSP引擎设置的ID编号:C2B02FB661CAC24E903526F3F3E13BFB

session的指定名称绑在一起的对象:admin

session中获取的属性有: password sex username

[跳转到session2.jsp](#)

跳转之后

session内置对象

session创建时JSP引擎设置的ID编号:C2B02FB661CAC24E903526F3F3E13BFB

session的指定名称绑在一起的对象:admin

加上最后一个时间限制之后，然后等待10秒，发现跳转到session2的时候ID号已经不一样了，即跳转到了新的页面

```

<body>
    <h1>session内置对象</h1>
    <hr>
    <%
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy年MM月dd日 HH:MM:ss");
        Date d=new Date(session.getCreationTime());
        session.setAttribute("username", "admin");
        session.setAttribute("password", "123456");
        session.setAttribute("sex", "women");
        //设置在页面中最长的停留时间，单位是秒
        session.setMaxInactiveInterval(10);
    %>
    session创建时JSP引擎设置的ID编号:<%=session.getId() %><br>
    session的指定名称绑在一起的对象:<%=session.getAttribute("username") %><br>
    session中获取的属性有: <%
        String[] names=session.getValueNames();
        for(int i=0;i<names.length;i++){
            out.println(names[i]+" &ampnbsp");
        }
    %><br>

    <a href="session2.jsp" target="_blank">跳转到session2.jsp</a><br>
</body>
</html>

```

session内置对象

session创建时JSP引擎设置的ID编号:C2B02FB661CAC24E903526F3F3E13BFB

session的指定名称绑在一起的对象:admin

session中获取的属性有: password sex username

[跳转到session2.jsp](#)

session内置对象

session创建时JSP引擎设置的ID编号:8EB640F08B4B373F86B65C8C34B49267

session的指定名称绑在一起的对象:admin

练习题

单选练习题

下面有关会话(session)描述不正确的是()。

A 会话是用来保存用户状态的一种机制

B 会话保存在客户端的内存里

C 会话保存在服务器的内存里

D 每一个会话对应一个唯一的sessionId

10.session的生命周期，分为三个阶段：创建、活动、销毁

Session的生命周期

创建：

当客户端第一次访问某个jsp或者Servlet时候，服务器会为当前会话创建一个SessionId,每次客户端向服务端发送请求时，都会将此SessionId携带过去，服务端会对此SessionId进行校验。

Session的生命周期

活动：

- 某次会话当中通过超链接打开的新页面属于同一次会话。
- 只要当前会话页面没有全部关闭，重新打开新的浏览器窗口访问同一项目资源时属于同一次会话。
- 除非本次会话的所有页面都关闭后再重新访问某个Jsp或者Servlet将会创建新的会话。

注意事项：注意原有会话还存在，只是这个旧的SessionId仍然存在于服务端，只不过再也没有客户端会携带它然后交予服务端校验。

Session的生命周期

销毁：

Session的销毁只有三种方式：

- 1、调用了session.invalidate()方法
- 2、Session过期（超时）
- 3、服务器重新启动

销毁的第一种方式：

注意销毁语句书写的位置，这样的话每次刷新获得的地址都是不相同的；如果没有添加销毁语句，则每次收到的地址都是相同的

测试通过调用session.invalidate()方法的方式对session进行销毁

```
</head>
<body>
    <h1>session内置对象</h1>
    <hr>
    <%
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy年MM月dd日 HH:MM:ss")
        Date d=new Date(session.getCreationTime());
        session.setAttribute("username","admin");
        session.setAttribute("password", "123456");
        session.setAttribute("sex", "women");
        //设置在页面中最长的停留时间，单位是秒
        //session.setMaxInactiveInterval(10);
    %>
    session创建时JSP引擎设置的ID编号:<%=session.getId() %><br>
    session的指定名称绑在一起的对象:<%=session.getAttribute("username") %><br>
    <%
        session.invalidate(); //销毁当前会话
    %>

    <a href="session2.jsp" target="_blank">跳转到session2.jsp</a><br>
</body>
</html>
```

Session对象

- Tomcat默认session超时时间为30分钟。
 - 设置session超时有两种方式：
 - 1.session.setMaxInactiveInterval(时间); //单位是秒
 - 2.在web.xml配置

```
<session-config>
  <session-timeout>
    10
  </session-timeout>
</session-config> //单位是分钟。
```

在进行测试过程中：

测试某次会话中通过超链接打开的新页面属于同一次会话

测试只要当前会话页面没有全部关闭，重新打开新的浏览器窗口访问同一项目资源时，属于同一次会话。

测试本次会话的所有页面都关闭的情况下，再重新访问某个JSP或Servlet时会创建新的会话。

注意：原有会话仍存在，但只是这个旧的sessionId仍存在于服务器端，但再也没有客户端会携带它然后交予服务器端校验。

练习：

单选练习题

有关session生命周期下列说法不正确的是()。

A session的生命周期分为创建、活动、销毁三个阶段

B 调用session.invalidate()方法可以销毁当前会话

C 重启web服务器会销毁所有的会话

D 每次重新打开新的浏览器窗口相当于开启了一次新的会话

除非本次会话的所有页面都关闭后再重新访问某个JSP或者Servlet将会创建新的会话

11.application对象

application对象:

- application对象实现了用户间数据的共享，可存放全局变量。
- application开始于服务器的启动，终止于服务器的关闭。
- 在用户的前后连接或不同用户之间的连接中，可以对application对象的同一属性进行操作。
- 在任何地方对application对象属性的操作，都将影响到其他用户对此的访问。
- 服务器的启动和关闭决定了application对象的生命。
- application对象是ServletContext类的实例。

application对象:

常用方法如下：

- public void setAttribute(String name, Object value) 使用指定名称将对象绑定到此会话。
- public Object getAttribute(String name) 返回与此会话中的指定名称绑定在一起的对象，如果没有对象绑定在该名称下，则返回 null。
- Enumeration.getAttributeNames() 返回所有可用属性名的枚举
- String getServerInfo() 返回JSP(SERVLET)引擎名及版本号

```
<body>
<h1>application内置对象</h1>
<hr>
<%
    application.setAttribute("username", "admin");
    application.setAttribute("password", "12346");
    application.setAttribute("sex", "woman");
%>
application对象的姓名: <%=application.getAttribute("username") %><br>
application对象所属的所有属性值: <%
    Enumeration names=application.getAttributeNames();
    while(names.hasMoreElements()){
        out.println(names.nextElement()+" &ampnbsp");
    }
%><br>
application对象返回JSP(Servlet)引擎名及版本号: <%=application.getServerInfo() %><br>
</body>
</html>
```

application内置对象

application对象的姓名：admin

application对象所属的所有属性值： javax.servlet.context.tempdir org.apache.catalina.resources org.apache.tomcat.util.scan.MergedWebXml org.apache.tomcat.InstanceManager sex

org.apache.catalina.jsp_classpath org.apache.jasper.compiler.ELInterpreter org.apache.jasper.compiler.TldLocationsCache org.apache.tomcat.JarScanner password

javax.websocket.server.ServerContainer org.apache.jasper.runtime.JspApplicationContextImpl username

application对象返回JSP(Servlet)引擎名及版本号：Apache Tomcat/7.0.85

练习

单选练习题

使用application 实现计数器效果，在application 中保存整型变量num，需要调用application对象的setAttribute方法，以下选项正确的是（ ）。

A setAttribute(" counter" ,num);

B setAttribute(num," counter");

C setAttribute(new Integer(num)," counter");

D setAttribute("counter" , " num");

12.page对象

page对象

page对象就是指向当前JSP页面本身，有点像类中的this指针，它是java.lang.Object类的实例。常用方法如下：

- `class getClass()` 返回此Object的类
- `int hashCode()` 返回此Object的hash码
- `boolean equals(Object obj)` 判断此Object是否与指定的Object对象相等
- `void copy(Object obj)` 把此Object拷贝到指定的Object对象中
- `Object clone()` 克隆此Object对象
- `String toString()` 把此Object对象转换成String类的对象
- `void notify()` 唤醒一个等待的线程
- `void notifyAll()` 唤醒所有等待的线程
- `void wait(int timeout)` 使一个线程处于等待直到timeout结束或被唤醒
- `void wait()` 使一个线程处于等待直到被唤醒

13.pageContext对象

pageContext对象

- pageContext对象提供了对JSP页面内所有的对象及名字空间的访问
- pageContext对象可以访问到本页所在的session，也可以取本页面所在的application的某一属性值
- pageContext对象相当于页面中所有功能的集大成者
- pageContext对象的本类名也叫pageContext。

pageContext对象

常用方法如下：

- `JspWriter getOut()` 返回当前客户端响应被使用的JspWriter流(`out`)
- `HttpSession getSession()` 返回当前页中的HttpSession对象(`session`)
- `Object getPage()` 返回当前页的Object对象(`page`)
- `ServletRequest getRequest()` 返回当前页的ServletRequest对象(`request`)
- `ServletResponse getResponse()` 返回当前页的ServletResponse对象(`response`)
- `void setAttribute(String name, Object attribute)` 设置属性及属性值
- `Object getAttribute(String name, int scope)` 在指定范围内取属性的值
- `int getAttributeScope(String name)` 返回某属性的作用范围
- `void forward(String relativeUrlPath)` 使当前页面重导到另一页面
- `void include(String relativeUrlPath)` 在当前位置包含另一文件

14.

Config对象

`config`对象是在一个Servlet初始化时，JSP引擎向它传递信息用的，此信息包括Servlet初始化时所要用到的参数（通过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），常用方法如下：

- `ServletContext getServletContext()` 返回含有服务器相关信息的ServletContext对象
- `String getInitParameter(String name)` 返回初始化参数的值
- `Enumeration getInitParameterNames()` 返回Servlet初始化所需所有参数的枚举

15.exception对象

Exception对象

`exception`对象是一个异常对象，当一个页面在运行过程中发生了异常，就产生这个对象。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。他实际上是`java.lang.Throwable`的对象，常用方法如下：

- `String getMessage()` 返回描述异常的消息
- `String toString()` 返回关于异常的简短描述消息
- `void printStackTrace()` 显示异常及其栈轨迹
- `Throwable FillInStackTrace()` 重写异常的执行栈轨迹

其中前面两个是比较常用的

`errorCode`属性表示当界面出现异常的时候这个异常交给那个界面去处理

最后的阶段案例

阶段案例

实现用户登录小例子。

用户名`admin`，密码`admin`，登录成功使用服务器内部转发到`login_success.jsp`页面，并且提示登录成功的用户名。如果登录失败则请求重定向到`login_failure.jsp`页面。



```
<%@ page language="java" contentType="text/html; charset=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>
<div id="container">
    <div class="login">
        <a href="#"></a>
    </div>
    <div id="box">
        <form action="dologin.jsp" method="post">
            <p class="main">
                <label>用户名:</label>
                <input name="username" value="" />
                <label>密码:</label>
                <input type="password" name="password" value="" />
            </p>
            <p class="space">
                <input type="submit" value="登录" class="login" style="cursor:pointer" />
            </p>
        </form>
    </div>
</div>
</body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<%
    String username="";
    String password="";
    request.setCharacterEncoding("utf-8");//防止中文乱码
    username=request.getParameter("username");
    password=request.getParameter("password");
    //如果用户名和密码都是admin，则说明登录成功
    if("admin".equals(username)&&"admin".equals(password)){
        //实现请求转发
        session.setAttribute("loginUsername", username);//用session返回用户名
        request.getRequestDispatcher("login_success.jsp").forward(request, response);
    }
    else{//登录失败
        response.sendRedirect("login_failure.jsp");
    }
%>
```

```
<%@ page language="java" contentType="text/html; charset=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>
<div id="container">
    <div class="login">
        <a href="#"></a>
    </div>
    <div id="box">
        <%
            String loginUser= " ";
            if(session.getAttribute("loginUsername")!=null){
                loginUser=session.getAttribute("loginUsername").toString();
            }
        %>
        欢迎您<font color="red"><%=loginUser %></font>登陆成功!
    </div>
</div>
</body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=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>
<div id="container">
    <div class="login">
        <a href="#"></a>
    </div>
    <div id="box">
        登录失败，请检查用户名或密码！<br>
        <a href="login.jsp">返回登录</a>
    </div>
</div>
</body>
</html>
```



用户名: 密码:

[登录](#)



欢迎您admin登陆成功!

登录失败，请检查用户或密码!

[返回登录](#)

第四章 Javabean

2018年4月9日 9:49

- Javabean简介
- Javabean设计原则
- Jsp中如何使用Javabean
- <jsp:useBeans>
- <jsp:getProperty>
- <jsp:setProperty>
- Javabean的四个作用域范围
- Model1简介
- 项目案例

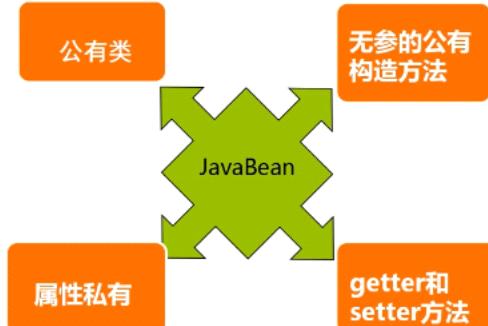
1. Javabeans简介

Javabean简介

Javabeans就是符合某种特定的规范的Java类。使用Javabeans的好处是解决代码重复编写，减少代码冗余，功能区分明确，提高了代码的维护性。

Javabean

Javabean的设计原则



符合特定规范的Java类，这就是一个JavaBeans

Javabean设计原则

```
//设计学生类  
public class Students  
{  
    private String name;  
    private int age;  
    public Students()  
    {  
        public void setName(String name) {this.name=name;}  
        public String getName() {return this.name;}  
        public void setAge(int age) {this.age = age;}  
        public int getAge() {return this.age;}  
    }  
}
```

2.JSP动作元素

什么是JSP动作元素？

什么是Jsp动作

JSP动作元素（action elements），动作元素为请求处理阶段提供信息。动作元素遵循 XML 元素的语法，有一个包含元素名的开始标签，可以有属性、可选的内容、与开始标签匹配的结束标签。

什么是Jsp动作

第一类是与存取JavaBean有关的，包括：

<jsp:useBean><jsp:setProperty><jsp:getProperty>

第二类是JSP1.2就开始有的基本元素，包括6个动作元素

<jsp:include><jsp:forward><jsp:param><jsp:plugin><jsp:params><jsp:fallback>

第三类是JSP2.0新增加的元素，主要与JSP Document有关，包括六个元素

<jsp:root><jsp:declaration><jsp:scriptlet><jsp:expression><jsp:text><jsp:output>

第四类是JSP2.0新增的动作元素，主要是用来动态生成XML元素标签的值，包括3个动作

<jsp:attribute><jsp:body><jsp:element>

第五类是JSP2.0新增的动作元素，主要是用在Tag File中，有2个元素

<jsp:invoke><jsp:doBody>

3.使用普通方式创建JavaBeans

在Jsp页面中如何使用Javabeans

1.像使用普通java类一样，创建javabean实例。

2.在Jsp页面中通常使用jsp动作标签使用javabean。

■useBeans动作

■setProperty动作

■getProperty动作

(1)

```
package com.jb;

public class Users {//新建一个公共类
    private String username;
    private String password;//成员变量全部都是私有
    public Users() {//公有的构造方法，不包含方法体

    }
    //对所有的私有成员属性设置get和set方法
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<%@page import="com.jb.Users" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "ht
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
<title>Insert title here</title>
</head>
<body>
<%
    Users users=new Users();
    users.setUsername("张三");
    users.setPassword("123456");
%>
<h1>使用普通方式创建javabean</h1>
<hr>
用户名: <%=users.getUsername() %><br>
密码: <%=users.getPassword() %><br>
</body>
</html>
```

使用普通方式创建javabean

用户名: 张三

密码: 123456

(2) ①useBean动作元素

```
<jsp:useBeans>
```

作用：在jsp页面中实例化或者在指定范围内使用javabean：

```
<jsp:useBean id="标示符" class="java类名" scope="作用范围" />
```

这里scope的，默认值是page

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<%@page import="com.jb.Users" %>
<!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>
    <jsp:useBean id="Myuser" class="com.jb.Users" scope="page"/>
    <h1>使用useBean创建javabean</h1>
    <hr>
    用户名: <%=Myuser.getUsername() %><br>
    密码: <%=Myuser.getPassword() %><br>
</body>
</html>
```

使用useBean创建javabean

用户名: null

密码: null

②setProperty

```
<jsp:setProperty>
```

作用：给已经实例化的Javabean对象的属性赋值，一共有四种形式。

```
<jsp:setProperty name = "JavaBean实例名" property = "*" /> (跟表单关联)
<jsp:setProperty name = "JavaBean实例名" property = "JavaBean属性名" />
(跟表单关联)
<jsp:setProperty name = "JavaBean实例名" property = "JavaBean属性名"
value = "BeanValue" /> (手工设置)
<jsp:setProperty name = "JavaBean实例名" property = "propertyName" param
= "request对象中的参数名"/> (跟request参数关联)
```

第一种方式是自动匹配，那个星号的意思，由于涉及到表单，所以重新写一个用户登录的程序

```

<%@ page language="java" contentType="text/html; charset=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>
    <h1>用户登录</h1>
    <hr>
    <form action="dologin.jsp" name="LoginForm" method="post">
        <table>
            <tr>
                <td>用户名: </td>
                <td><input type="text" name="username" value=""></td>
            </tr>
            <tr>
                <td>密码: </td>
                <td><input type="password" name="password" value=""></td>
            </tr>
            <tr>
                <td colspan="2"><input type="submit" value="登录"/></td>
            </tr>
        </table>
    </form>
</body>
</html>

```

```

<body>
    <jsp:useBean id="MyUser" class="com.jb.Users" scope="page"/>
    <h1>setProperty动作元素</h1>
    <hr>
    <!-- 通过自动匹配表单自动为属性赋值 -->
    <%--<jsp:setProperty name="MyUser" property="*"/> --%>
    <!-- 位数姓名赋值 -->
    <%--<jsp:setProperty name="MyUser" property="username"/>
    <jsp:setProperty name="MyUser" property="password"/> --%>
    <!-- 通过手动赋值 -->
    <jsp:setProperty name="MyUser" property="username" value="lisi"/>
    <jsp:setProperty name="MyUser" property="password" value="123456"/>
    <%-->
    <!-- 通过URL传递参数的方式对属性进行赋值 -->
    <%--<jsp:setProperty name="MyUser" property="username"/>
    <jsp:setProperty name="MyUser" property="password" param="mypass"/> --%>
    用户名: <%=MyUser.getUsername() %><br>
    密码: <%=MyUser.getPassword() %>
</body>
</html>

```

setProperty动作元素

用户名: zhangsan
密码: 123456

第二种方式，在语句中只指定固定的属性值，那么输出也就是固定的属性值

用户登录

用户名:

密码:

第三种方式，不论输入表单的值是什么，输出都是lisi 123456

setProperty动作元素

用户名: lisi
密码: 123456

第四种方式，通过URL传参数给属性进行赋值，在login这里做如下的修改

```
body>
    <h1>用户登录</h1>
    <hr>
    <form action="doLogin.jsp?mypass=999999" name="LoginForm" method="post">
        <table>
            <tr>
                <td>用户名:</td>
                <td><input type="text" name="username" value=""></td>
            </tr>
        </table>
    </form>
```

setProperty动作元素

用户名: zhangsan
密码: 999999

4.getProperty

<jsp:getProperty>

作用：获取指定Javabean对象的属性值。

```
<jsp:getProperty name="JavaBean实例名" property="属性名" />
```

```
<body>
<jsp:useBean id="MyUser" class="com.jb.Users" scope="page"/>
<h1>setProperty动作元素</h1>
<hr>
<!-- 通过自动匹配表单自动为属性赋值 -->
<jsp:setProperty name="MyUser" property="*"/>
<!-- 位数姓名赋值 -->
<%--<jsp:setProperty name="MyUser" property="username"/>
<jsp:setProperty name="MyUser" property="password"/> --%>
<!-- 通过手动赋值 -->
<%--
<jsp:setProperty name="MyUser" property="username" value="lisi"/>
<jsp:setProperty name="MyUser" property="password" value="123456"/>
--%>

<%--
<!-- 通过URL传递参数的方式对属性进行赋值 -->
<%--<jsp:setProperty name="MyUser" property="username"/>
<jsp:setProperty name="MyUser" property="password" param="mypass"/> --%>
<!-- 使用传统的表达式方式来获取用户名和密码 -->
<%--用户名: <%=MyUser.getUsername() %><br>
密码: <%=MyUser.getPassword() %> --%>
<!-- 使用getProperty方式获取用户名和密码 -->
用户名: <jsp:getProperty name="MyUser" property="username"/><br>
密码:<jsp:getProperty name="MyUser" property="password"/>
```

setProperty动作元素

用户名: zhangsan

密码:123456

5.Javabean四个作用域范围

Javabean的四个作用域范围

说明：使用useBeans的scope属性可以用来指定javabean的作用范围。

- page //仅在当前页面有效
- request //可以通过HttpRequest.getAttribute()方法取得JavaBean对象。
- session //可以通过HttpSession.getAttribute()方法取得JavaBean对象。
- application //可以通过application.getAttribute()方法取得Javabean对象

用户登录

用户名:

密码:

setProperty动作元素

用户名: zhangsan
密码:123456

[测试application的作用域范围](#)

测试Javabeans的四个作用域范围

用户名:zhangsan
密码:123456

测试Javabeans的四个作用域范围

用户名:zhangsan
密码:123456
用户名:zhangsan
密码: 123456

二.使用session内置对象获取

setProperty动作元素

用户名: zhangsan
密码:123456

[使用session的用户名和密码](#)

测试Javabeans的四个作用域范围

用户名:zhangsan

密码:123456

用户名:zhangsan

密码: 123456

三、使用request内置对象获取

测试Javabeans的四个作用域范围

用户名:null

密码:null

用户名:null

密码: null

不能使用请求转发，只能使用请求重定向

```
<br>
<br>
<a href="testscope.jsp">使用request的用户名和密码</a>
<%
    request.getRequestDispatcher("testscope.jsp").forward(request,response);
%>
</body>
```

没有第二步出现超链接的步骤直接跳转到这个界面

测试Javabeans的四个作用域范围

用户名:zhangsan

密码:123456

用户名:zhangsan

密码: 123456

四、使用page内置对象获取（这里要注意防止空指针yichang）

无论page怎么改都不能获取，因为它只对当前页面有效

```

<hr>
用户名:<jsp:getProperty name="MyUser" property="username"/><br>
密码:<jsp:getProperty name="MyUser" property="password"/><br>
<!-- 使用内置对象获取用户名和密码 -->
<%-->
用户名:<%=((Users)application.getAttribute("MyUser")).getUsername()%><br>
密码:<%=((Users)application.getAttribute("MyUser")).getPassword()%><br>
--%>
<!-- 使用session内置对象获取用户名和密码 -->
<%-->
用户名:<%=((Users)session.getAttribute("MyUser")).getUsername()%><br>
密码:<%=((Users)session.getAttribute("MyUser")).getPassword()%><br>
--%>
<!-- 使用request内置对象获取用户名和密码 -->
<%-->
用户名:<%=((Users)request.getAttribute("MyUser")).getUsername()%><br>
密码:<%=((Users)request.getAttribute("MyUser")).getPassword()%><br>
--%>
<!-- 使用page内置对象获得用户名和密码 -->
<%
    String username=" ";
    String password=" ";
    if(pageContext.getAttribute("MyUser")!=null){
        username=((Users)pageContext.getAttribute("MyUser")).getUsername();
        password=((Users)pageContext.getAttribute("MyUser")).getPassword();
    }
%>
用户名: <%=username%><br>
密码: <%=password%><br>
</body>
</html>

```

测试Javabeans的四个作用域范围

用户名:**null**
 密码:**null**
 用户名: **null**
 密码: **null**

单选练习题

在Java Web 应用程序的jsp页面中有如下一行代码：要使user对象可以作用于整个Web应用，下划线中应填入（ ）。

A page

B request

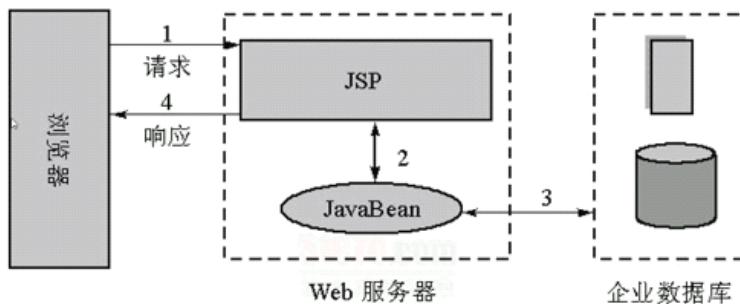
C session

D application

6. Model1

- Model 1模型出现前，整个Web应用的情况：几乎全部由JSP页面组成，JSP页面接收处理客户端请求，对请求处理后直接做出响应。
- 弊端：在界面层充斥着大量的业务逻辑的代码和数据访问层的代码，Web程序的可扩展性和可维护性非常差。
- Javabean的出现可以使jsp页面中使用Javabean封装的数据或者调用Javabean的业务逻辑代码,这样大大提升了程序的可维护性。

Model1简介



其实就是JSP+JavaBean

单选练习题

以下关于Javabean说法不正确的是()。

A Javabean就是符合某种设计规范的java类。

B 在Model1中，由Jsp页面去调用Javabean。

C Javabean只能封装数据不能封装业务逻辑。

D Javabean一般把属性设计为私有，使用setter和getter访问属性。

阶段案例

使用jsp+javabean完成用户登录功能



第五章 JSP的无状态管理

2018年4月10日 15:01

Jsp状态管理

- ① **http协议无状态性**
- ② 保存用户状态的两大机制
- ③ Cookie简介
- ④ Cookie的创建与使用
- ⑤ Session与Cookie的对比

1、http超文本传输协议的先天不足：http协议的无状态性

http协议的无状态性

无状态是指，当浏览器发送请求给服务器的时候，服务器响应客户端请求。

但是当同一个浏览器再次发送请求给服务器的时候，服务器并不知道它就是刚才那个浏览器。

简单地说，就是服务器不会去记得你，所以就是无状态协议。

2、Cookie概述

保存用户的状态的两大机制

■Session

■Cookie

什么是Cookie？

Cookie：中文名称为“小甜饼”，是Web服务器保存在客户端的一系列文本信息。

典型应用一：判定注册用户是否已经登录网站。

典型应用二：“购物车”的处理。

Cookie的作用

- 对特定对象的追踪
- 保存用户网页浏览记录与习惯
- 简化登录

安全风险：容易泄露用户信息

3.JSP页面中创建与使用Cookie

Jsp中创建与使用Cookie

创建Cookie对象

```
Cookie newCookie = new Cookie(String key, Object value);
```

写入Cookie对象

```
response.addCookie(newCookie);
```

读取Cookie对象

```
Cookie[] cookies = request.getCookies();
```

读取Cookie返回的是一个数组

Jsp中创建与使用Cookie

● 常用方法

方法名称	说 明
void setMaxAge(int expiry)	设置cookie的有效期，以秒为单位
void setValue(String value)	在cookie创建后，对cookie进行赋值
String getName()	获取cookie的名称
String getValue()	获取cookie的值
int getMaxAge()	获取cookie的有效时间，以秒为单位

对Cookie添加的参数以及获取到的值都是字符串的形式，Cookie本身就是文本文档的形式

4.案例：Cookie在登录中的应用

Jsp中创建与使用Cookie

- 实现记忆用户名和密码功能



```
<body>
    <h1>用户登录</h1>
    <hr>
    <%
        request.setCharacterEncoding("utf-8");
        String username= " ";
        String password= " ";
        Cookie[] cookies=request.getCookies();
        if(cookies!=null && cookies.length>0){
            for(Cookie c:cookies){
                if(c.getName().equals("username")){
                    username=URLDecoder.decode(c.getValue(), "utf-8");
                }
                if(c.getName().equals("password")){
                    password=URLDecoder.decode(c.getValue(), "utf-8");
                }
            }
        }
    %>
    <form action="dologinC.jsp" name="LoginForm" method="post">
        <table>
            <tr>
                <td>用户名:</td>
                <td><input type="text" name="username" value="<%=username %>"></td>
            </tr>
            <tr>
                <td>密码:</td>
                <td><input type="password" name="password" value="<%=password %>"></td>
            </tr>
            <tr>
                <td colspan="2"><input type="checkbox" name="isUseCookie" checked="checked">十天内记住我的登录状态</td>
            </tr>
            <tr>
                <td colspan="2" align="center"><input type="submit" name="登录"><input type="reset" name="取消"></td>
            </tr>
        </table>
    </form>
</body>
</html>
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Insert title here</title>
</head>
<body>
    <h1>登录成功</h1>
    <br>
    <br>
    <br>
    <%
        request.setCharacterEncoding("utf-8");
        String[] isUseCookies=request.getParameterValues("isUseCookie");
        if(isUseCookies!=null &&isUseCookies.length!=0){
            String username=URLEncoder.encode(request.getParameter("username"),"utf-8");
            String password=URLEncoder.encode(request.getParameter("password"),"utf-8");
            Cookie usernameCookie=new Cookie("username",username);
            Cookie passwordCookie=new Cookie("password",password);
            usernameCookie.setMaxAge(86400);
            passwordCookie.setMaxAge(86400);
            response.addCookie(usernameCookie);
            response.addCookie(passwordCookie);
        }
        else{//如果用户不想保存
            Cookie[] cookies=request.getCookies();
            if(cookies!=null &&cookies.length>0){
                for(Cookie c:cookies){
                    if(c.getName().equals("username")||c.getName().equals("password")){
                        c.setMaxAge(0);
                        response.addCookie(c);
                    }
                }
            }
        }
    %>
    <a href="userC.jsp">查看用户信息</a>
</body>
</html>
```

```

<%@page import="java.net.URLDecoder,java.util.*"%>
<%@page import="org.apache.tomcat.util.http.Cookies"%>
<%@ page language="java" contentType="text/html; charset=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>
    <h1>用户信息</h1>
    <hr>
    <%
        request.setCharacterEncoding("utf-8");
        String username="";
        String password="";
        Cookie[] cookies=request.getCookies();
        if(cookies!=null && cookies.length>0){
            for(Cookie c:cookies){
                if(c.getName().equals("username")){
                    username=URLDecoder.decode(c.getValue(),"utf-8");
                }
                if(c.getName().equals("password")){
                    password=URLDecoder.decode(c.getValue(),"utf-8");
                }
            }
        }
    %>
    <br>
    <br>
    <br>
    用户名: <%=username %><br>
    密码:<%=password %><br>
</body>
</html>

```

5.session和cookie对比

Session与Cookie对比



关于第一点session更精确的来讲是保存在服务器端的内存中，cookie是以文本文件的形式保存在客户端

单选练习题

下面关于Cookie与Session的对比描述不正确的是
()

A Cookie是客户端保存用户状态的机制

B Session是服务器端保存用户状态的机制

C Cookie与Session都可以设置生存期限

D Cookie与Session都可以保存任意大小的对象类型



你的答案是D，答对了！

解析

Cookie中对保存对象的大小是有限制的

单选练习题

如果想要设置当前Cookie的生存期限为24小时，
以下代码正确的是

A setMaxAge(86400);

B setMaxAge(1440);

C getMaxAge(86400);

D getMaxAge(1440);

这个函数的单位是秒

第六章 JSP指令与动作元素

2018年4月10日 18:49

JSP一共有三大指令

1.include指令

指令与动作

- include指令
- include动作
- include指令与include动作的区别
- <jsp:forward>动作
- <jsp:param> 动作
- <jsp:plugin> 动作

指令与动

include指令

语法：

```
<%@ include file="URL"%>
```

最常用的属性是file属性

实例：用include指令包含一个页面，并且在页面上显示被包含的内容

```
<%@page import="java.text.* ,java.util.*"%>
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<%
    SimpleDateFormat sdf=new SimpleDateFormat("yyyy年MM月dd日");
    Date d=new Date();
    out.println(sdf.format(d));
%>
```

```
<%@ page language="java" contentType="text/html; charset=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>
    <h1>include指令</h1>
    <hr>
    <%@include file="date.jsp"%>
</body>
</html>
```

注意调用SDF的格式，这种简单的都不会

2.include动作

include动作

语法：

```
<jsp:include page="URL" flush="true|false"/>
```

page

要包含的页面

flush

被包含的页面是否从缓冲区读取

flush的默认值是false

```
<%@ page language="java" contentType="text/html; charset=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>
    <h1>include动作</h1>
    <hr>
    <jsp:include page="date.jsp" flush="false" />
</body>
</html>
```

3.include指令与include动作的区别(关于JSP面试的典型的问题)

include指令与include动作比较

	include指令	jsp:include动作
语法格式	<%@ include file= ".." %>	<jsp:include page= ".." >
发生作用的时间	页面转换期间	请求期间
包含的内容	文件的实际内容	页面的输出
转换成的Servlet	主页面和包含页面转换为一个Servlet	主页面和包含页面转换为独立的Servlet
编译时间	较慢——资源必须被解析	较快
执行时间	稍快	较慢——每次资源必须被解析

3.指令包含的是代码，动作包含的是结果

4.转换成同一个Servlet可以理解为转换成同一个类

Servlet可以简单的理解为就是一个类

```
Date d = new Date();
SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日");
String s = sdf.format(d);
out.println(s);

out.write('\r');
out.write('\n');
out.write("\r\n");
out.write(" </body>\r\n");
out.write("</html>\r\n");
} catch (java.lang.Throwable t) {
    if (!(t instanceof javax.servlet.jsp.SkipPageException)){
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
            try { out.clearBuffer(); } catch (java.io.IOException e) {}
        if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
    }
} finally {
    _jspxFactory.releasePageContext(_jspx_page_context);
}
```

下两图是include动作的源代码

date_jsp.class	2014/10/23 15:28	CLASS 文件	4 KB
date_jsp.java	2014/10/23 15:28	JAVA 文件	4 KB
Include_005fAction.jsp.class	2014/10/23 15:33	CLASS 文件	6 KB
Include_005fAction_jsp.java	2014/10/23 15:33	JAVA 文件	5 KB
Include_005fCommand.jsp.class	2014/10/23 15:28	CLASS 文件	6 KB
Include_005fCommand_jsp.java	2014/10/23 15:28	JAVA 文件	5 KB

```

out.write("<hr>\r\n");
out.write(" ");
org.apache.jasper.runtime.JspRuntimeLibrary.include(request, response, "date.jsp", out
out.write("\r\n");
out.write("</body>\r\n");
out.write("</html>\r\n");
} catch (java.lang.Throwable t) {
if (!(t instanceof javax.servlet.jsp.SkipPageException)){
out = _jspx_out;
if (out != null && out.getBufferSize() != 0)
try { out.clearBuffer(); } catch (java.io.IOException e) {}
if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
}
} finally {
_jspxFactory.releasePageContext(_jspx_page_context);

```

4.forward动作

forward动作

语法：

<jsp:forward page="URL" />

等同于：

request.getRequestDispatcher("/url").forward(request,response);

等同于服务器内部跳转指令

```

<%@ page language="java" contentType="text/html; charset=utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Insert title here</title>
</head>
<body>
    <h1>用户登录</h1>
    <hr>
    <form action="forward_action.jsp" name="loginForm" method="post">
        <table>
            <tr>
                <td>用户名:</td>
                <td><input type="text" name="username"/></td>
            </tr>
            <tr>
                <td>密码:</td>
                <td><input type="password" name="password"/></td>
            </tr>
            <tr>
                <td colspan="2"><input type="submit" value="登录"></td>
            </tr>
        </table>
    </form>
</body>
</html>

```

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/1999/xhtml">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Insert title here</title>
</head>
<body>
    <h1>用户资料</h1>
    <hr>
    <%
        request.setCharacterEncoding("utf-8");
        String username= " ";
        String password= " ";
        if(request.getParameter("username")!=null){
            username=request.getParameter("username");
        }
        if(request.getParameter("password")!=null){
            password=request.getParameter("password");
        }
    %>
    用户名: <%=username %><br>
    密码: <%=password %><br>
</body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/1999/xhtml">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Insert title here</title>
</head>
<body>
    <h1>forward指令</h1>
    <hr>
    <!--<jsp:forward page="user.jsp"/><!--&gt;
    &lt;%
        request.getRequestDispatcher("user.jsp").forward(request, response);
    %&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>
```

用户资料

用户名: 张三
密码: 999999

用户资料

用户名: 张三
密码: 123456

5.param动作

param动作

语法 :

```
<jsp:param name="参数名" value="参数值">
```

常常与 <jsp:forward> 一起使用，作为其的子标签。

该动作可以增加原表单中没有的属性值，也可以修改原来有的参数值，下面的操作结果第二个，输入的密码是“123456”，但是输出是“888888”，输入不会影响密码值

```
<%@ page language="java" contentType="text/html; charset=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>
    <h1>用户登录</h1>
    <hr>
    <form action="dologin.jsp" name="loginForm" method="post">
        <table>
            <tr>
                <td>用户名:</td>
                <td><input type="text" name="username"/></td>
            </tr>
            <tr>
                <td>密码:</td>
                <td><input type="password" name="password"/></td>
            </tr>
            <tr>
                <td colspan="2"><input type="submit" value="登录"></td>
            </tr>
        </table>
    </form>
</body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=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>
    <jsp:forward page="user.jsp">
        <jsp:param value="admin!126.com" name="email"/>
        <jsp:param value="888888" name="password"/>
    </jsp:forward>
</body>
</html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h1>用户资料</h1>
    <hr>
    <%
        request.setCharacterEncoding("utf-8");
        String username=" ";
        String password=" ";
        String email=" ";
        if(request.getParameter("username")!=null){
            username=request.getParameter("username");
        }
        if(request.getParameter("password")!=null){
            password=request.getParameter("password");
        }
        if(request.getParameter("email")!=null){
            email=request.getParameter("email");
        }
    %>
    用户名: <%=username %><br>
    密码: <%=password %><br>
    电子邮箱: <%=email %><br>
</body>
</html>
```

用户资料

用户名: 张三
密码: 123456
电子邮箱: admin@126.com

用户资料

用户名: 张三
密码: 888888
电子邮箱: admin@126.com

单选练习题

下面关于include指令与include动作的对比描述不正确的是()

A <jsp:include>动作在编译期间被执行，而include指令在页面请求期间被执行

B 页面内容经常变化时更适合使用<jsp:include>动作

C 页面内容不经常变化时更适合使用include指令

D <jsp:include>动作包含的是执行结果，而include指令包含的是文件内容

选A，说反了

第七章 项目实例

2018年4月10日 20:22

童鞋们，对JAVA数据库编程知识
JDBC的掌握有利于更好地学习本章
内容呦！

案例：商品浏览记录的实现

使用Cookie实现

采用Model1(Jsp+JavaBean)实现

- 实现DBHelper类
- 创建实体类
- 创建业务逻辑类 (DAO)
- 创建页面层

创建实体类==创建商品类，就是和商品表是一一对应的

创建业务逻辑类 (DAO) 即JavaBean，也就是非常重要的DAO设计模式

创建页面层

项目原型：首先在webcontent文件夹下面建立两个静态页面index.jsp和detail.jsp

Index.jsp负责显示所有的商品信息，显示商品的详情，image是图片，sql放的是商品的脚本

第二步：DBHelper类设计

实现DBHelper类型

```
String driver = "com.mysql.jdbc.Driver"; //数据库驱动
//连接数据库的URL地址
String url="jdbc:mysql://localhost:3306/shopping?useUnicode=true&characterEncoding=UTF
String username="root";//数据库的用户名
String password="";//数据库的密码
```

这四个是作为属性存在的

第三步：商品实体类设计

导入sql脚本，这里是items.sql

	id	name	city	price	number	picture
▶	1	沃特篮球鞋	佛山	180	500	001.jpg
	2	安踏运动鞋	福州	120	800	002.jpg
	3	耐克运动鞋	广州	500	1000	003.jpg
	4	阿迪达斯T血衫	上海	388	600	004.jpg
	5	李宁文化衫	广州	180	900	005.jpg
	6	小米3	北京	1999	3000	006.jpg
	7	小米2S	北京	1299	1000	007.jpg
	8	thinkpad笔记本	北京	6999	500	008.jpg
	9	dell笔记本	北京	3999	500	009.jpg
	10	ipad5	北京	5999	500	010.jpg

采用Model1 (JSP+JavaBean) 实现

- 实现DBHelper类
- 创建实体类
- 创建业务逻辑类 (DAO)
- 创建页面层

4. 获取所有商品资料方法的实现

采用Model1 (JSP+JavaBean) 实现

- 实现DBHelper类
- 创建实体类
- 创建业务逻辑类 (DAO)

获得商品的业务信息并在页面上显示

Java web中的DAO的设计模式



如何把浏览记录保存在cookie中？



把每次浏览的商品编号保存在字符串中，编号和编号之间用分隔符分隔，每次取出前五条记录。

```
String list = "";
//从客户端获得Cookies集合
Cookie[] cookies = request.getCookies();
//遍历这个Cookies集合
for(Cookie c:cookies)
{
    if(c.getName().equals("ListViewCookie"))
    {
        list = c.getValue();
    }
}
list+=request.getParameter("id")+",";
//如果浏览记录超过1000条，清零。
String[] arr = list.split(",");
if(arr!=null&&arr.length>0)
{
    if(arr.length>=1000)
    {
        list="";
    }
}
```

The screenshot shows the MyEclipse IDE interface with several open files:

- DBHelper.java**: Contains Java code for item retrieval logic.
- Items.java**: Contains Java code for item representation.
- ItemsDAO.java**: Contains Java code for item data access.
- details.jsp**: A JSP page template.
- index.jsp**: A JSP page template.

The code in **DBHelper.java** is as follows:

```
    {
        ArrayList<Items> itemlist = new ArrayList<Items>();
        int iCount=5; //每次返回前五条记录
        if(list!=null&&list.length()>0)
        {
            String[] arr = list.split(",");
            //如果商品记录大于等于5条
            if(arr.length>=5)
            {
                for(int i=arr.length-1;i>=arr.length-iCount;i--)
                {
                    itemlist.add(getItemsById(i));
                }
            }
            else
            {
                for(int i=arr.length-1;i>=0;i--)
                {
                    itemlist.add(getItemsById(i));
                }
            }
        }
    }
```

The code in **index.jsp** is as follows:

```
String list = ;
//从客户端获得Cookies集合
Cookie[] cookies = request.getCookies();
//遍历这个Cookies集合
if(cookies!=null&&cookies.length>0)
{
    for(Cookie c:cookies)
    {
        if(c.getName().equals("ListViewCookie"))
        {
            list = c.getValue();
        }
    }
}

list+=request.getParameter("id")+",";
//如果浏览记录超过1000条，清零。
String[] arr = list.split(",");
if(arr!=null&&arr.length>0)
{
    if(arr.length>=1000)
    {
        list="";
    }
}
Cookie cookie = new Cookie("ListViewCookie",list);
response.addCookie(cookie);
```

```

// 获取最近访问的商品列表
public ArrayList<Items> getViewList(String list)
{
    System.out.println("list:" + list);
    ArrayList<Items> itemlist = new ArrayList<Items>();
    int iCount=5; //每次返回前五条记录
    if(list!=null&&list.length()>0)
    {
        String[] arr = list.split(",");
        System.out.println("arr.length=" + arr.length);
        //如果商品记录大于等于5条
        if(arr.length>=5)
        {
            for(int i=arr.length-1;i>=arr.length-iCount;i--)
            {
                itemlist.add(getItemsById(Integer.parseInt(arr[i])));
            }
        }
        else
        {
            for(int i=arr.length-1;i>=0;i--)
            {
                itemlist.add(getItemsById(Integer.parseInt(arr[i])));
            }
        }
        return itemlist;
    }
    else
    {
        return null;
    }
}

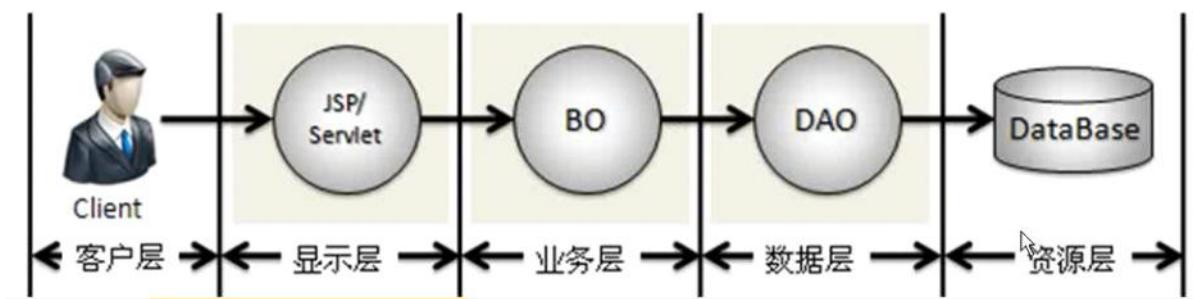
```

本章目标

- ❖ 掌握程序的分层定义及各层的主要功能；
- ❖ 掌握DAO的开发；
- ❖ 掌握JSP + DAO的开发模式

企业分层架构

- ❖ 客户层：因为现在都采用了B/S开发架构，所以一般都使用浏览器进行访问。
- ❖ 显示层：使用JSP/Servlet进行页面效果的显示。
- ❖ 业务层（Business Object，业务对象）：会将多个原子性的DAO操作进行组合，组合成一个完整的业务逻辑。
- ❖ 数据层（DAO）：提供多个原子性的DAO操作，例如：增加、修改、删除等，都属于原子性的操作。



资源层主要是数据库的操作层，里面可以进行各种的数据存储，但是这些数据存储操作的时候肯定要靠 SQL 语句，之前也提到了，如果在一个程序中出现了过多的 SQL 语句的直接操作，则 JSP 页面变得相当的复杂，而且也不便于程序的可重用性。
可以通过一个专门的数据库的操作组件完成，那么这个就是数据层的功能。
业务层是整个项目的核心。

DAO 组成

- ◆ 在整个DAO的中实际上都是以接口为操作标准的，即：客户端依靠DAO实现的接口进行操作，而服务端要将接口进行具体的实现，DAO由以下几个部分组成：
 - ◆ DatabaseConnection：专门负责数据库的打开与关闭操作的类；
 - ◆ VO：主要由属性、setter、getter方法组成，VO类中的属性与表中的字段相对应，每一个VO类的对象都表示表中的每一条记录；
 - ◆ DAO：主要定义操作的接口，定义一系列数据库的原子性操作，例如：增加、修改、删除、按ID查询等；
 - ◆ Impl：DAO接口的真实实现类，完成具体的数据库操作，但是不负责数据库的打开和关闭；
 - ◆ Proxy：代理实现类，主要完成数据库的打开和关闭并且调用真实实现类对象的操作；
 - ◆ Factory：工厂类，通过工厂类取得一个DAO的实例化对象。

雇员表			
No.	列名称	描述	
1	empno	雇员编号，使用数字表示，长度是四位数字	<pk>
2	ename	雇员姓名，使用字符串表示，长度是十位字符串	
3	job	雇员工作	
4	hiredate	雇佣日期，使用日期形式表示	
5	sal	基本工资，使用小数表示，其中小数位 2 位，整数位 5 位	

Servlet生命周期之自动装载

2018年4月17日 20:00

一共有三种情况：

(1)

Servlet生命周期

- 在下列时刻Servlet容器装载Servlet：

Servlet容器启动时自动装载某些Servlet，实现它只需要在web.xml文件中的<Servlet></Servlet>之间添加如下代码：<loadon-startup>1</loadon-startup>数字越小表示优先级别越高。

(2)

Servlet生命周期

- 在下列时刻Servlet容器装载Servlet：

- 在Servlet容器启动后，客户首次向Servlet发送请求。

(3)

- Servlet类文件被更新后，重新装载Servlet。

当客户端首次访问程序运行顺序

```
信息: Server startup in 2756 ms
构造方法被执行!
1. Servlet初始化 init()
2. Servlet服务 doGet()
2. Servlet服务 doGet()
```