JSP_EL_JSTL_Servlet

1. JSP

Java server pages : 主要是用来替代servlet程序回传html页面的数据，本质上是servlet。

它会被编译成Java程序，里面继承了HttpServlet类。

底层还是用输出流将页面数据回传 out.write();

JSP页面常用属性

i. language 属性 表示 jsp 翻译后是什么语言文件。暂时只支持 java。

ii. contentType 属性 表示 jsp 返回的数据类型是什么。也是源码中 response.setContentType()参数值

iii. pageEncoding 属性 表示当前 jsp 页面文件本身的字符集。

iv. import 属性 跟 java 源代码中一样。用于导包，导类。

给out输出流使用

v. autoFlush 属性 设置当 out 输出流缓冲区满了之后，是否自动刷新冲级区。默认值是 true。

vi. buffer 属性 设置 out 缓冲区的大小。默认是 8kb **别动**

vii. errorPage 属性 设置当 jsp 页面运行时出错，自动跳转去的错误页面路径

viii. isErrorPage 属性 设置当前 jsp 页面是否是错误信息页面。默认是 false。如果是 true 可以获取异常信息。

ix. session 属性 设置访问当前 jsp 页面，是否会创建 HttpSession 对象。默认是 true。

x. extends 属性 设置 jsp 翻译出来的 java 类默认继承谁

JSP页面声明脚本 不用

声明脚本的格式是： <%! 声明 java 代码 %>

JSP表达式脚本 常用

表达式脚本的格式是：<%=表达式%> 这个是直接out.print()输出到页面上，不能以分号结束

JSP代码脚本

代码脚本的格式是：

<%

    java 语句 实现想要的java语句功能，包括if else for while...

%

JSP注释

<!-- 这是 html 注释 --> <%-- 这是 jsp 注释 --%>

<%

// 单行 java 注释

/* 多行 java 注释 */

%

JSP九大内置对象

Tomcat 在翻译 jsp 页面成为 Servlet 源代码后，**内部提供的九大对象，叫内置对象**

1. **Request　　请求对象**
2. Response　　响应对象
3. **pageContext　JSP页面上下文对象**
4. **Session　　会话对象**
5. **Application　servletContext对象**
6. Config　　servletConfig对象
7. Out　　输出流对象
8. Page　　指向当前JSP的对象
9. Exception　　异常对象

JSP四大域对象

域对象可以像Map一样存储数据，主要是通过作用范围去决定使用。

1. **pageContext (PageContextImpl 类)** 当前 jsp 页面范围内有效
2. **request (HttpServletRequest 类)**　一次请求内有效
3. **session (HttpSession 类)**　一个会话范围内有效（打开浏览器访问服务器，直到关闭浏览器）
4. **application (ServletContext 类)** 整个web工程范围内都有效(只要 web 工程不停止数据都在)

优先级、作用范围从小到大是：pageContext ==>> request ==>> session ==>> application

JSP中out输出和response.getWriter输出的区别

Response & out 都是给用户做响应时使用的。

当JSP页面中所有代码执行完成后会做以下两个操作：

1. 执行out.flush()操作，会把out缓冲区的数据追加写入到response缓冲区末尾
2. 会执行response的属性操作，把全部数据写给客户端

注意：实际上jsp页面是通过out 统一输出的，所以建议使用 out 去输出。

out.write() 与 out.print() 的区别

out.write() 输出字符串没有问题

out.print() 输出任意数据都没有问题（**都转换成为字符串**后调用的 write 输出）

所以我们用 out.print() 就好啦。

上面说的是页面上的，这里是servlet中。

Java

```java
boolean isHasCert = userDao.checkIsHasCert(userDto, session);
resp.getWriter().println(isHasCert); // PrintWriter对象 HttpServletResponse
```

Lua

```lua
byte[] sealBytes = gmSignSeal.getGmSealImg();
response.getOutputStream().write(sealBytes); // 这个是struts2中的response对象
```

其他

<%@ include file="/include/footer.jsp"%> 静态包含，只是复制一份到输出的地方

<jsp:include page="/include/footer.jsp"> 动态包含，会翻译成java代码

    <jsp:param name="username" value="bbj"/>

    <jsp:param name="password" value="root"/>

</jsp:include>

<jsp:forward page="/scope2.jsp"></jsp:forward> 页面上的请求转发

JavaWeb 的三大组件分别是：Servlet 程序、Filter 过滤器、Listener 监听器。是规范也是接口。

2. EL

EL 表达式的全称是：Expression Language，是表达式语言

EL 表达式主要是代替 jsp 页面中的表达式脚本在 jsp 页面中进行数据的输出

对于四个域对象输出同样遵循从小到大的优先级去输出，还可以输出bean对象的属性，做运算

VBScript

```
<body>
    <% request.setAttribute("key"," 值"); %>
    表达式脚本输出 key 的值是:
    <%=request.getAttribute("key1")==null?"":request.getAttribute("key1")%>
<br/>
    EL 表达式输出 key 的值是: ${key1}
</body>
```

3. JSTL

JSTL 全称是指 JSP Standard Tag Library JSP 标准标签库。是一个不断完善的开放源代码的JSP 标签库。

EL 表达式主要是为了替换 jsp 中的表达式脚本，而**标签库则是为了替换代码脚本**。

PowerShell

```
<table border="1">
    <c:forEach begin="1" end="10" var="i">
        <tr>
            <td>第${i}行</td>
        </tr>
    </c:forEach>
</table>
如果按照原来的jsp页面写法，<% for(int i = 1; i < 10; i++) { %>.....各种难看的标签代码
```

4. Servlet

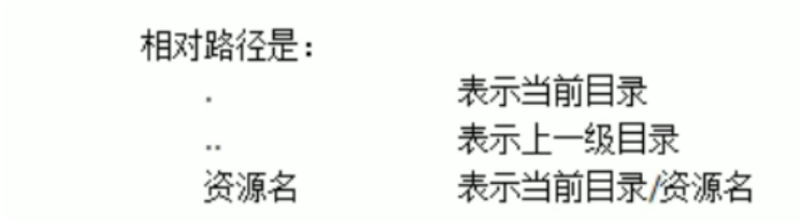
HttpServletRequest类

- 通过实现servlet接口并重写service方法（doGet() doPost() ）
- 通过继承 HttpServlet 实现servlet 程序，到web.xml中配置访问地址
- URI 是资源路径，是项目名和路径
- URL是访问路径
- 通常请求的参数出现乱码时，可以设置请求体的字符集为UTF-8
  - request.setCharacterEncoding("UTF-8"); **不过需要在获取参数之前**

- 请求的转发：服务器接收到请求时从一个资源跳转到另一个资源的操作。
  - 怎么走：request.getRequestDispatcher("/anotherServletPath"); // 需要斜杠
  - **/ 斜杠表示的地址是 http://ip:port/工程名/ ,映射到IDEA代码的web目录**
    - 那就走：requestDispatcher.forward(req, resp);

  特点：

    - 浏览器地址栏没有变化
    - 一次请求
    - 共享request域对象数据
    - **可以转发到WEB-INF目录下**，如果直接从页面上访问是不行的
    - base标签设置页面相对路径工作时参照的地址，href参照的地址值，如果不设置参照的就是当前访问路径的



- web中 / 斜杠的不同意义
  a. / 斜杠如果被浏览器解析，得到的地址是 http://ip:port/
  b. 如果被服务器解析，得到的地址是：http://ip:port/工程路径

HttpServletResponse 类

HttpServletRequest 表示请求过来的信息，HttpServletResponse 表示所有响应的信息，设置返回给客户端的信息。

1. **两个流**的说明（只能同时用一个）
   a. 字节流 getOutputStream(); 常用于下载（传递二进制数据）
   b. 字符流 getWriter(); 常用于回传字符串

C++

```cpp
// 解决乱码1（不推荐）
// 设置服务器字符集为 UTF-8
resp.setCharacterEncoding("UTF-8");
// 通过响应头，设置浏览器也使用 UTF-8 字符集
resp.setHeader("Content-Type", "text/html; charset=UTF-8");

// 解决乱码2（推荐）
// 它会同时设置服务器和客户端都使用 UTF-8 字符集，还设置了响应头
// 此方法一定要在获取流对象之前调用才有效
resp.setContentType("text/html; charset=UTF-8");

PrintWriter writer = resp.getWriter();
writer.write("response's content!!!");

resp.getOutputStream().write("二进制数据");
```

2. **请求重定向**

请求重定向，是指客户端给服务器发请求，然后服务器告诉客户端说。我给你一些地址。你去新地址访问。叫请求重定向（因为之前的地址可能已经被废弃）

特点：

- 地址发生变化
- 两次请求
- 不共享request域中的数据
- 不能访问WEB-INF下的资源
- 可以访问工程外的资源

两种写法：

a. resp.setStatus(302);
   resp.setHeader("Location", "http://localhost:8080");
b. **resp.setRedirect**("http://lcoalhost:8080"); // 推荐使用