JSP标准动作小记

**1.<jsp:useBean /> <jsp:getProperty />**

与bean相关的标准动作

没有标准动作：

<html><body>

<% foo.Person p = (foo.Person) request.getAttribute(“person”); %>

Person is <%= p.getName() %>

</body></html>

有标准动作：

<html><body>

<**jsp:useBean** id=”person” class=”foo.Person” scope=”request” />

Person is <**jsp:getProperty** name=”person” property=”name” />

</body></html>

（1）jsp:getProperty标签的name属性对应jsp:useBean的id属性，是bean名的意思，用id属性是为了唯一性

（2）jsp:getProperty标签的property属性值对应bean的属性名

（3）jsp:getProperty这一整句表名取bean person的name属性值

（4） jsp:useBean这里是在request范围找名字为person的foo.Person类的bean

**2. <jsp:useBean>还能创建一个bean**

这里，如果<jsp:useBean>找不到一个名为person的request属性对象，那么就会创建一个！（就如之前学的request.getSession，没有就自动创建~）

我们来看在\_jspService方法中对应的代码：

foo.Person person = null;

synchronized(request){

person = (foo.person)\_jspx\_page\_context.**getAttribute**(“person”, PageContext.REQUEST\_SCOPE);

if(person == null){

person = new foo.Person();

\_jsp\_page\_context.**setAttribute**(“person”, person, PageContext.REQUESET\_SCOPE);

}

}

**3. <jsp:setProperty>**

<**jsp:useBean** id=”person” class=”foo.Person” scope=”request” />

Person is <**jsp:setProperty** name=”person” property=”name” value=”Alex” />

如果这样用的话，假设我们想对一个新的bean设置属性值，但是useBean指定的**不一定是一个新的bean**！**这样程序就有了二义性**

因此我们需要一些条件来判断，但怎么把useBean的条件和setProperty关联呢？

**我们可以把setProperty标签放入useBean标签体中！**

<**jsp:useBean** id=”person” class=”foo.Person” scope=”request”>

<**jsp:setProperty** name=”person” property=”name” value=”Alex” />

</jsp:useBean>

**这样的话，只有创建新bean时才会设置bean的属性值**

为什么呢？让我们来看**<jsp:useBean>有体时，\_jspService中的代码**：

foo.Person person = null;

person = (foo.Person) \_jspx\_page\_context.getAttribute(“person”, PageContext.PAGE\_SCOPE);

//如果person==null，说明不存在该bean，然后**有体时**会对体中信息进行解析注入

if(person == null){

person = new foo.Person();

\_jspx\_page\_context.setAttribute(“person”, person, PageContext.PAGE\_SCOPE);

//**有体时**，利用解析后的信息对新bean的属性进行设置

org.apache.jasper.runtime.JspRuntimeLibrary.introspecthelper(

\_jspx\_page\_context.**findAttribute**(“person”),

“name”,

“Alex”,

null,

null,

false

);

}

也就是说，当需要创建一个新bean（需要注明class）时，标记体才有意义。

因为由源代码看，标记体只为新bean服务。

4.可以建立多态的bean引用

如果Person是一个抽象类，person是抽象类引用，但指向了一个子类对象，那么如果不增加type属性说明引用类型，而把这个抽象类写在class的话，容器会一直找抽象类的实例类，但这是不可能存在的。

（**type是引用类型，class是对象类型**）

然而，我们只需为<jsp:useBean>增加一个type属性，就能支持多态了，把抽象类写在**type**，把对象的类型写在class

<jsp:useBean id=”person” type=”foo.Person” class=”foo.Employee” scope=”page” />

**type可以class类型、抽象类型、接口**

对应的\_jspService代码为：

**foo.Person** person = null; **//指定的引用类型**

if(person == null){

person = new **foo.Employee**(); **//指定的对象类型**

}

如上，如果只是用type而没有class，那么容器不知道找什么构造器来创对象。

会报错。

5.另外，

class必须有一个无参的构造器。

scope默认为page

type **引用类型**

class**对象类型**

**6.如果只指定了type，那么容器知道，你必须已经有一个指定的bean属性，而且引用类型就是type**

**7.bean标记会自动转换基本类型的性质**

即把http请求中的String数据，按照bean性质的基本类型转换（如果只有基本类型的话）

但是如果使用脚本就不会自动转换 <%= request.getParameter(“empID”) %>

**8.<jsp:include>**

与include指令<%@ include file=”haha.jsp” %>用法一致原理不同

include指令静态预编译加载，include动作动态运行时加载

<jsp:include page=”haha.jsp” />

**9.<jsp:param>**

用来定制<jsp:include>的内容，求同存异~

<jsp:include page=”Header.jspf” >

<jsp:param name=”subTitle” value=”What the fuck” />

</jsp:include>

这相当于为标签增加一个属性，这种做法不适用于include指令（静态）

**为什么说是定制呢？**

你看原来的Header.jspf:

<img src=”image/Web-Services.jpg” ><br>

<em><strong>**What the fuck**</strong></em><br>

而现在我们可以用属性动态替换：

<img src=”image/Web-Services.jpg” ><br>

<em><strong>${**param.subTitle**}/strong></em><br>

这个<jsp:param>应该是属于pageScope的param，用**EL表达式**时，会从pageScope取到值。

**10.<jsp:forward>**

在JSP中进行转发，不推荐使用。因为根据MVC的原则，JSP页面不应有业务逻辑。

这个标签的作用应该就相当于：

<% request.getRequestDispacher(“Hola.jsp”).forward(request,response)

%>

EL是

${ pageContext.request.~~getRequestDispatcher~~ } **//这是错误的**

EL设计的初衷就是方便取值，何况是**在JSP中，不能包含业务逻辑**