jsf

一、jsf（javaServer faces）是用于构建web应用程序的新的标准java框架。它提供了一种以组件为中心来开发java web用户界面的方法，从而简化了开发。

jsf将良好的构建模型—视图—控制器（mvc）设计继承到他的体系结构中，确保了应用程序具有更高的可维护性。

二、bean

jsf使用java bean来达到程序业务逻辑和视图的分离的目的。bean调用业务逻辑层或者将业务处理结果放置在bean中，供ui组件取出

jsf将bean的管理集中在faces-config.xml中

<managed-bean>

     <managed-bean-name>cartManagedBean</managed-bean-name>//名称

     <managed-bean-class>com.cart.bean.CartManagedBean</managed-bean-class>//类的全路径

     <managed-bean-scope>session</managed-bean-scope>//作用范围，取值范围request、session、application

   </managed-bean>

三、事件处理

1、值变事件

如果使用者改变了输入值后，就会发生值变事件。

<:h:selectOneMenu value=”#{user.locale}” onchange=”this.form.submit();” valueChangeListerner=”#{user.changeLocale}”>

<f:selectItem itemValue=”中文” itemLabel=”中文”/>

<f:selectItem itemValue=”英文” itemLabel=”英文”/>

</h:selectOneMenu>

2、动作事件

使用者点击一个按钮或一个链接，触发动作事件

<h:commandButton action="#{cartManagedBean.forShowShoppingCart}" value="浏览购物车" styleClass="input\_button"></h:commandButton>

<h:commandLink action=*"*#{cartManagedBean.forShowProductDetails}*"* immediate=*"false"*>

                  <h:outputText value=*"  详细信息  "* />

                  <t:updateActionListener property=*"*#{cartManagedBean.dataid}*"* value=*"*#{p.id}*"* />

               </h:commandLink>

四、el语言

jsf el是用来存取信息的简易语言

语法是以#开始，将变量或者运算表达式，放在{}之间

如果要输出字符串，必须以单引号括起来

五、标签

<h:dataTable>

⒈<h:dataTable>

* 阵列
* java.util.List的实例
* java.sql.ResultSet的实例
* javax.servlet.jsp.jstl.sql.Result的实例
* javax.faces.model.DataModel的实 例

　　对于前四种型态，JSF实际上是以javax.faces.model.DataModel加以包装，DataModel是个抽象类别，其子类别都是位于javax.faces.model这个package下：

* ArrayDataModel
* ListDataModel
* ResultDataModel
* ResultSetDataModel
* ScalarDataModel

　　如果您想要对表格资料有更多的控制，您可以直接使用DataModel来设定表格资料，呼叫DataModel的 setWrappedObject()方法可以让您设定对应型态的资料，呼叫getWrappedObject()则可以取回资料，例如：

**TableBean.java**

package onlyfun.caterpillar;

 import java.util.\*;

import javax.faces.model.DataModel;

import javax.faces.model.ListDataModel;

 public class TableBean {

    private DataModel model;

    private int rowIndex = -1;

    public DataModel getUsers() {

        if(model == null) {

            model = new ListDataModel();

            model.setWrappedData(getUserList());

        }

        return model;

    }

    private List getUserList() {

        List userList = new ArrayList();

        userList.add(new UserBean("caterpillar", "123456"));

        userList.add(new UserBean("momor", "654321"));

        userList.add(new UserBean("becky", "7890"));

        return userList;

    }

    public int getSelectedRowIndex() {

        return rowIndex;

    }

    public String select() {

        rowIndex = model.getRowIndex();

        return "success";

    }

}

　　在这个Bean中，我们直接设定DataModel，将userList设定给它，如您所看到的，我们还可以取得DataModel?的各个变项，在这个例子中，select()将作为点选表格之后的事件处理方法，我们可以即由DataModel?的getRowIndex ()来取得所选的是哪一row的资料，例如：

**index.jsp**

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<html>

<link href="styles.css" rel="stylesheet" type="text/css"/>

<body>

<f:view>

    <h:form>

        <h:dataTable value="#{tableBean.users}" var="user"

                  styleClass="orders"

                  headerClass="ordersHeader"

                  rowClasses="evenColumn,oddColumn">

            <h:column>

                <f:facet name="header">

                    <h:outputText value="Name"/>

                </f:facet>

                <h:commandLink action="#{tableBean.select}">

                    <h:outputText value="#{user.name}"/>

                </h:commandLink>

                <f:facet name="footer">

                    <h:outputText value="\*\*\*\*"/>

                </f:facet>

            </h:column>

            <h:column>

                <f:facet name="header">

                    <h:outputText value="Password"/>

                </f:facet>

                <h:outputText value="#{user.password}"/>

                <f:facet name="footer">

                    <h:outputText value="\*\*\*\*"/>

                </f:facet>

            </h:column>

        </h:dataTable>

    </h:form>

    Selected Row: <h:outputText

                     value="#{tableBean.selectedRowIndex}"/>

</f:view>

</body>

</html>

　　DataModel的rowIndex是从0开始计算，当处理ActionEvent时，JSF会逐次递增rowIndex的值，这让您可以得知目前正在处理的是哪一个 row的资料，一个执行的图示如下：



**表头与表尾**

<h:dataTable>配合<h:column>来以表格的方式显示资料，< h:column>中只能包括 JSF组件或者是<f:facet>，JSF支援两种facet：header与footer。分别用以设定表格的表头与表尾文字，一个设定的例子如下：

<h:dataTable value="#{tableBean.userList}" var="user">

    <h:column>

        <f:facet name="header">

            <h:outputText value="Name"/>

        </f:facet>

        <h:outputText value="#{user.name}"/>

        <f:facet name="footer">

            <h:outputText value="\*\*\*\*"/>

        </f:facet>

    </h:column>

    <h:column>

        <f:facet name="header">

            <h:outputText value="Password"/>

        </f:facet>

        <h:outputText value="#{user.password}"/>

        <f:facet name="footer">

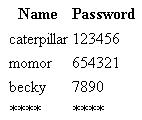
            <h:outputText value="\*\*\*\*"/>

        </f:facet>

    </h:column>

 </h:dataTable>

　　所产生的表格如下所示：



　　另外，对于表头、表尾仍至于每一行列，都可以分别设定CSS风格

**styles.css**

.orders {

   border: thin solid black;

 }

 .ordersHeader {

   text-align: center;

   font-style: italic;

   color: Snow;

   background: Teal;

 }

 .evenColumn {

   height: 25px;

   text-align: center;

   background: MediumTurquoise;

 }

 .oddColumn {

   text-align: center;

   background: PowderBlue;

 }

　　可以在我们的页面中如下加入：

....

 <link href="styles.css" rel="stylesheet" type="text/css"/>

 ....

 <h:dataTable value="#{tableBean.userList}" var="user"

                 styleClass="orders"

                 headerClass="ordersHeader"

                 rowClasses="evenColumn,oddColumn">

    <h:column>

        <f:facet name="header">

            <h:outputText value="Name"/>

        </f:facet>

        <h:outputText value="#{user.name}"/>

        <f:facet name="footer">

            <h:outputText value="\*\*\*\*"/>

        </f:facet>

    </h:column>

    <h:column>

        <f:facet name="header">

            <h:outputText value="Password"/>

        </f:facet>

        <h:outputText value="#{user.password}"/>

        <f:facet name="footer">

            <h:outputText value="\*\*\*\*"/>

        </f:facet>

    </h:column>

 </h:dataTable>

　　则显示的表格结果如下：



⒉命令类标签

命令类标签包括commandButton与commandLink，其主要作用在于提供一个命令按钮或连结，以下举例说明：

commandButton

　　显示一个命令按钮，即输出<input> HTML标签，其type属性可以设定为button、submit或reset，预设是submit，按下按钮会触发 javax.faces.event.ActionEvent，使用例子如下：

<h:commandButton value="送出" action="#{user.verify}"/>

　　您可以设定image属性，指定图片的URL，设定了image属性的话，<input>标签的type属性会被设定为image，例如：

<h:commandButton value="#{msgs.commandText}"

                  image="images/logowiki.jpg"

                  action="#{user.verify}"/>

**commandLink**

　　产生超链接，会输出<a> HTML标签，而href属性会有'#'，而onclick属性会含有一段JavaScript程序，这个JavaScript的目的是按下连结后自动提交表单，具体来说其作用就像按钮，但外观却是超链接，包括在本体部份的内容都会成为超链接的一部份，一个使用的例子如下：

<h:commandLink value="#{msgs.commandText}"

                action="#{user.verify}"/>

　　产生的HTML输出范例如下：

<a href="#" onclick="document.forms['\_id3']['\_id3:\_idcl'].value='\_id3:\_id13'; document.forms['\_id3'].submit(); return false;">Submit</a>

　　如果搭配<f:param>来使用，则所设定的参数会被当作请求参数一并送出，例如：

<h:commandLink>

   <h:outputText value="welcome"/>

   <f:param name="locale" value="zh\_TW"/>

 </h:commandLink>

⒊其他标签

**<h:graphicImage>**

　　这个卷标会绘制一个HTML <img>卷标，value可以指定路径或图片URL，路径可以指定相对路径或绝对路径，例如：

<h:graphicImage value="/images/logowiki.jpg"/>

**<h:panelGrid>**

　　这个标签可以用来作简单的组件排版，它会使用HTML表格卷标来绘制表格，并将组件置于其中，主要指定columns属性，例如设定为 2：

<h:panelGrid columns="2">

    <h:outputText value="Username"/>

    <h:inputText id="name" value="#{userBean.name}"/>

    <h:outputText value="Password"/>

    <h:inputText id="password" value="#{userBean.password}"/>

    <h:commandButton value="submit" action="login"/>

    <h:commandButton value="reset" type="reset"/>

 </h:panelGrid>

　　则自动将组件分作 2 个 column来排列，排列出来的样子如下：



　　<h:panelGrid>的本体间只能包括JSF组件，如果想要放入非JSF组件，例如简单的样版（template）文字，则要使用<f:verbatim>包括住，例如：

<h:panelGrid columns="2">

    <f:verbatim>Username</f:verbatim>

    <h:inputText id="name" value="#{userBean.name}"/>

    <f:verbatim>Password</f:verbatim>

    <h:inputText id="password" value="#{userBean.password}"/>

    <h:commandButton value="submit" action="login"/>

    <h:commandButton value="reset" type="reset"/>

 </h:panelGrid>

**<h:panelGroup>**

　　这个组件用来将数个JSF组件包装起来，使其看来像是一个组件，例如：

<h:panelGrid columns="2">

    <h:outputText value="Username"/>

    <h:inputText id="name" value="#{userBean.name}"/>

    <h:outputText value="Password"/>

    <h:inputText id="password" value="#{userBean.password}"/>

    <h:panelGroup>

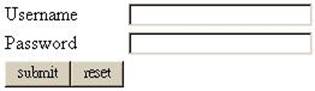
        <h:commandButton value="submit"action="login"/>

        <h:commandButton value="reset" type="reset"/>

    </h:panelGroup>

 </h:panelGrid>

　　在<h:panelGroup>中包括了两个<h:commandButton>，这使得< h:panelGrid>在处理时，将那两个<h:commandButton>看作是一个组件来看待，其完成的版面配置如下所示：



⒋输出类标签

**outputLabel**

　　产生<label> HTML卷标，使用for属性指定组件的client ID，例如：

<h:inputText id="user" value="#{user.name}"/>

<h:outputLabel for="user" value="#{user.name}"/>

　　这会产生像是以下的标签：

<input id="user" type="text" name="user" value="guest"/>

<label for="user">

**outputLink**

　　产生<a> HTML标签，例如：

<h:outputLink value="../index.jsp">

    <h:outputText value="Link to Index"/>

    <f:param name="name" value="MyName"/>

</h:outputLink>

　　你可搭配<f:param>帮链结加上参数，所有的参数都会变成 name=value 的型态附加在链接后。

　　value所指定的内容也可以是JSF EL绑定。

**outputForm0at**

　　产生指定的文字讯息，可以搭配<f:param>来设定讯息的参数以格式化文本讯息，例如：

<f:loadBundle basename="messages" var="msgs"/>

 <h:outputFormat value="#{msgs.welcomeText}">

     <f:param value="Hello"/>

     <f:param value="Guest"/>

 </h:outputFormat>

　　如果您的messages.properties包括以下的内容：

welcomeText={0}, Your name is {1}.

　　则{0}与{1}会被取代为<f:param>设定的文字，最后显示的文字会是：

Hello, Your name is Guest.

　　另一个使用的方法则是：

<h:outputFormat value="{0}, Your name is {1}.">

     <f:param value="Hello"/>

     <f:param value="Guest"/>

 </h:outputFormat>

**outputText**

　　简单的显示指定的值或绑定的讯息，例如：

<h:outputText value="#{user.name}"/>

⒌输入类标签

　输入类标签包括了inputText、inputTextarea、inputSecret、 inputHidden，分别举例说明如下：

**inputText**

　　显示单行输入字段，即输出<input> HTML标签，其type属性设定为text，例如：

<h:inputText value="#{user.name}"/>

**inputTextarea**

　　显示多行输入文字区域，即输出<textarea> HTML标签，例如：

<h:inputTextarea value="#{user.command}"/>

**inputSecret**

　　显示密码输入字段，即输出<input> HTML标签，其type属性设定为password，例如：

<h:inputSecret value="#{user.password}"/>

　　您可以设定redisplay属性以决定是否要显示密码字段的值，默认是false。

**inputHidden**

　　隐藏域，即输出<input> HTML标签，其type属性设定为hidden，隐藏域的值用于保留一些讯息于客户端，以在下一次发送表单时一并送出，例如：

<h:inputHidden value="#{user.hiddenInfo}"/>

⒍选则类标签

　选择类的标签可略分为单选卷标与多选卷标，依外型的不同可以分为单选钮（Radio）、复选框（CheckBox）、列示方块（ListBox）与选单（Menu），以下分别先作简单的说明。

**<h:selectBooleanCheckbox>**

　　在视图上呈现一个复选框，例如：

我同意 <h:selectBooleanCheckbox value="#\{user.aggree\}"/>

　　value所绑定的属性必须接受与传回boolean型态。这个组件在网页上呈现的外观如下：

https://preview111.tower.im/data/test/2017/0914/12/120009_99131_bVNKKTw/index.files/image006.jpg

**<h:selectOneRadio>、<h:selectOneListbox>、<h: selectOneMenu>**

　　这三个标签的作用，是让使用者从其所提供的选项中选择一个项目，所不同的就是其外观上的差别，例如：

<h:selectOneRadio value="#{user.education}">

   <f:selectItem itemLabel="高中" itemValue="高中"/>

   <f:selectItem itemLabel="大学" itemValue="大学"/>

   <f:selectItem itemLabel="研究所以上" itemValue="研究所以上"/>

 </h:selectOneRadio><p>

　　value所绑定的属性可以接受字符串以外的型态或是自定义型态，但记得如果是必须转换的型态或自定义型态，必须搭配标准转换器 或 自定义转换器 来转换为对象，<h:selectOneRadio>的外观如下：

https://preview111.tower.im/data/test/2017/0914/12/120009_99131_bVNKKTw/index.files/image007.jpg

　　您也可以设定layout属性，可设定的属性是lineDirection、pageDirection，预设是lineDirection，也就是由左到右来排列选项，如果设定为pageDirection，则是由上至下排列选项，例如设定为：

<h:selectOneRadio layout="pageDirection"

                   value="#{user.education}">

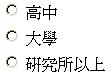
   <f:selectItem itemLabel="高中" itemValue="高中"/>

   <f:selectItem itemLabel="大学" itemValue="大学"/>

   <f:selectItem itemLabel="研究所以上" itemValue="研究所以上"/>

 </h:selectOneRadio><p>

　　则外观如下：



　　<h:selectOneListbox>、<h:selectOneMenu>的设定方法类似于<h: selectOneRadio>，以下分别列出<h:selectOneListbox>、<h: selectOneMenu>的外观：

https://preview111.tower.im/data/test/2017/0914/12/120009_99131_bVNKKTw/index.files/image009.jpg

https://preview111.tower.im/data/test/2017/0914/12/120009_99131_bVNKKTw/index.files/image010.jpg

**<h:selectManyCheckbox>、<h:selectManyListbox>、<h: selectManyMenu>**

　　这三个标签提供用户复选项目的功能，一个<h:selectManyCheckbox>例子如下：

<h:selectManyCheckbox layout="pageDirection"

                       value="#{user.preferColors}">

     <f:selectItem itemLabel="红"itemValue="false"/>

     <f:selectItem itemLabel="黄"itemValue="false"/>

     <f:selectItem itemLabel="蓝"itemValue="false"/>

 </h:selectManyCheckbox><p>

　　value所绑定的属性必须是数组或集合（Collection）对象，在这个例子中所使用的是boolean数组，例如：

**UserBean.java**

package onlyfun.caterpillar;

 public class UserBean {

    private boolean[] preferColors;

    public boolean[] getPreferColors() {

        return preferColors;

    }

    public void setPreferColors(boolean[] preferColors) {

        this.preferColors = preferColors;

    }

    ......

 }

　　如果是其它型态的对象，必要时必须搭配转换器（Converter）进行字符串与对象之间的转换。

　　下图是<h:selectManyCheckbox>的外观，这是将layout设定为pageDirection的外观：

https://preview111.tower.im/data/test/2017/0914/12/120009_99131_bVNKKTw/index.files/image011.jpg

　　<h:selectManyListbox>的设定方法类似，其外观如下：

https://preview111.tower.im/data/test/2017/0914/12/120009_99131_bVNKKTw/index.files/image012.jpg

　　<h:selectManyMenu>在不同的浏览器中会有不同的外观，在Mozilla Firefox中是这样的：

https://preview111.tower.im/data/test/2017/0914/12/120009_99131_bVNKKTw/index.files/image013.jpg

　　在Internet Explorer则是这样的：

https://preview111.tower.im/data/test/2017/0914/12/120009_99131_bVNKKTw/index.files/image014.jpg

　选择类标签可以搭配<f:selectItem>或<f:selectItems>标签来设定选项，例如：

<f:selectItem itemLabel="高中"

               itemValue="高中"

               itemDescription="学历"

               itemDisabled="true"/>

　　itemLabel属性设定显示在网页上的文字，itemValue设定发送至伺服端时的值，itemDescription 设定文字描述，它只作用于一些工具程序，对HTML没有什么影响，itemDisabled设定是否选项是否作用，这些属性也都可以使用JSF Expression Language来绑定至一个值。

　　<f:selectItem>也可以使用value来绑定一个传回javax.faces.model.SelectItem的方法，例如：

<f:selectItem value="#{user.sex}"/>

　　则绑定的Bean上必须提供下面这个方法：

....

     public SelectItem getSex()  {

        return new SelectItem("男");

     }

 ....

　　如果要一次提供多个选项，则可以使用<f:selectItems>，它的value绑定至一个提供传回SelectItem?的数组、集合，或者是Map对象的方法，例如：

<h:selectOneRadio value="#{user.education}">

     <f:selectItems value="#{user.educationItems}"/>

 </h:selectOneRadio><p>

　　这个例子中<f:selectItems>的value绑定至user.educationItems，其内容如下：

....

     private SelectItem[] educationItems;

    public SelectItem[] getEducationItems() {

        if(educationItems == null) {

            educationItems = new SelectItem[3];

            educationItems[0] =

                  new SelectItem("高中", "高中");

            educationItems[1] =

                  new SelectItem("大学", "大学");

            educationItems[2] =

                  new SelectItem("研究所以上", "研究所以上");

        }

        return educationItems;

    }

 ....

　　在这个例子中，SelectItem的第一个建构参数用以设定value，而第二个参数用以设定label，SelectItem还提供有数个建构函式，记得可以参考一下线上API文件。

　　您也可以提供一个传回Map对象的方法，Map的key-value会分别作为选项的label-value，例如：

<h:selectManyCheckbox layout="pageDirection"

                       value="#{user.preferColors}">

    <f:selectItems value="#{user.preferColorItems}"/>

 </h:selectManyCheckbox><p>

　　您要提供下面的程序来搭配上面这个例子：

....

    private Map preferColorItems;

    public Map getPreferColorItems() {

        if(preferColorItems == null) {

            preferColorItems = new HashMap();

            preferColorItems.put("红","Red");

            preferColorItems.put("黄", "Yellow");

            preferColorItems.put("蓝", "Blue");

        }

        return preferColorItems;

    }

 ....

⒎组件标签

完成Component的自定义，接下来要设定一个自定义Tag与之对应，自定义Tag的目的，在于设定 Component属性，取得Componenty型态，取得Renderer型态值等；属性的设定包括了设定静态值、设定绑定值、设定验证器等等。

　　要自定义与Component对应的Tag，您可以继承UIComponentTag，例如：

**TextWithCmdTag.java**

package onlyfun.caterpillar;

 import javax.faces.application.Application;

 import javax.faces.component.UIComponent;

 import javax.faces.context.FacesContext;

 import javax.faces.el.ValueBinding;

 import javax.faces.webapp.UIComponentTag;

 public class TextWithCmdTag extends UIComponentTag {

    private String size;

    private String value;

    public String getComponentType() {

        return"onlyfun.caterpillar.TextWithCmd";

    }

    public String getRendererType() {

        return null;

    }

    public void setProperties(UIComponent component) {

        super.setProperties(component);

        setStringProperty(component, "size", size);

        setStringProperty(component, "value", value);

    }

    private void setStringProperty(UIComponent component,

                       String attrName, String attrValue) {

        if(attrValue == null)

            return;

        if(isValueReference(attrValue)) {

            FacesContext context =

                         FacesContext.getCurrentInstance();

            Application application =

                         context.getApplication();

            ValueBinding binding =

                   application.createValueBinding(attrValue);

            component.setValueBinding(attrName, binding);

        }

        else {

            component.getAttributes().

                          put(attrName, attrValue);

        }

    }

    public void release() {

        super.release();

        size = null;

        value = null;

    }

    public String getSize() {

        return size;

    }

    public void setSize(String size) {

        this.size = size;

    }

    public String getValue() {

        return value;

    }

    public void setValue(String value) {

        this.value = value;

    }

 }

　　首先看到这两个方法：

public String getComponentType() {

        return"onlyfun.caterpillar.TextWithCmd";

    }

    public String getRendererType() {

        return null;

    }

　　由于我们的Component目前不使用Renderer，所以getRendererType()传回null值，而 getComponentType()在于让JSF取得这个Tag所对应的Component，所传回的值在faces-config.xml中要有定义，例如：

....

 <component>

    <component-type>

        onlyfun.caterpillar.TextWithCmd

    </component-type>

    <component-class>

        onlyfun.caterpillar.UITextWithCmd

    </component-class>

 </component>

 ....

　　藉由faces-config.xml中的定义，JSF可以得知 onlyfun.caterpillar.TextWithCmd的真正类别，而这样的定义方式很显然的，您可以随时换掉<component- class>所对应的类别，也就是说，Tag所对应的Component是可以随时替换的。

　　在设定Component属性值时，可以由component.getAttributes()取得Map对象，并将卷标属性值存入Map 中，这个Map对象可以在对应的Component中使用getAttributes()取得，例如在上一个主题中的UITextWithCmd中可以如下取得存入Map的size属性：

**UITextWithCmd.java**

package onlyfun.caterpillar;

 import java.io.IOException;

 import java.util.Map;

 import javax.faces.component.UIInput;

 import javax.faces.context.FacesContext;

 import javax.faces.context.ResponseWriter;

 public class UITextWithCmd extends UIInput {

    ....

    private void encodeTextField(ResponseWriter writer,

                        String clientId) throws IOException {

        ....

        String size = (String) getAttributes().get("size");

        if(size != null) {

            writer.writeAttribute("size", size, null);

        }

        .....

    }

    ....

 }

　　可以使用isValueReference()来测试是否为JSF Expression Language的绑定语法，如果是的话，则我们必须建立ValueBinding对象，并设定值绑定：

....

 private void setStringProperty(UIComponent component,

                        String attrName, String attrValue) {

        if(attrValue == null)

            return;

        if(isValueReference(attrValue)) {

            FacesContext context =

                          FacesContext.getCurrentInstance();

            Application application =

                          context.getApplication();

            ValueBinding binding =

                   application.createValueBinding(attrValue);

            component.setValueBinding(attrName, binding);

        }

        else {

            component.getAttributes().

                         put(attrName, attrValue);

        }

    }

 ....

　　如果是value属性，记得在上一个主题中我们提过，从UIOutput继承下来的getValue()方法可以取得 Component的value设定值，这个值可能是静态的属性设定值，也可能是JSF Expression的绑定值，默认会先从组件的属性设定值开始找寻，如果找不到，再从绑定值（ValueBinding对象）中找寻。

　　最后，我们必须提供自定义Tag的tld檔：

**textcmd.tld**

<?xml version="1.0" encoding="UTF-8"?>

 <taglib version="2.0"

     xmlns="http://java.sun.com/xml/ns/j2ee"

     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

 xsi:schemaLocation=

 "http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary\_2\_0.xsd">

 <tlib-version>1.0</tlib-version>

 <jsp-version>2.0</jsp-version>

 <short-name>textcmd</short-name>

 <uri>http://caterpillar.onlyfun.net/textcmd</uri>

 <tag>

     <name>textcmd</name>

     <tag-class>onlyfun.caterpillar.TextWithCmdTag</tag-class>

     <body-content>empty</body-content>

     <attribute>

        <name>size</name>

     </attribute>

     <attribute>

        <name>value</name>

        <required>true</required>

     </attribute>

 </tag>

 </taglib>