Struts2

# action

1. 配置第一个struts的步骤：
2. 建立项目
   1. 打开myeclipse
   2. 新建项目
   3. 选择myeclipse-java enterprise projects-web project
   4. 项目名称：struts2\_0100\_introduction
   5. 选择java ee 5.0
3. 设定server
   1. window – preferences – myeclipse – servers – tomcat – 6.x
   2. 选择tomcat home directory
   3. 选择enable
   4. finish
4. 设定jdk环境
   1. window – preferences – java – installed jres
   2. 如果没有对应的JDK（不是JRE），选择add
   3. 选择standard VM -> next
   4. 选择JDK对应的Directory
   5. 将刚刚设定的JDK设为默认
5. 建立第一个struts2.1的程序
   1. 找到struts目录下对应的apps目录
   2. 解压struts2-blank-2.1.6.war
   3. Copy对应的lib的jar文件，需要除junit和spring-test之外的所有文件，其中commons-io的jar包会在文件上传和下载时需要，其他为必须
   4. Copy对应的struts.xml到src目录，在package explorer视图进行操作
   5. 注释掉struts.xml的多余内容
   6. 建立HelloStruts2\_1.jsp文件
   7. \*修改jsp文件的默认编码属性window-preferences-web-jspfiles-设为Chinese,National Standard
   8. 在struts.xml中照原配置进行对应的配置
   9. 修改对应的web.xml，建立struts2的filter（参考struts自带的项目）
6. 运行项目
   1. 可以首先部署到tomcat上项目
   2. 也可以项目右键debug as – myeclipse server app，选择刚刚建立好的server
7. 其他
   1. 学习建立struts的dev-mode，好处在于我们修改了配置文件的时候能够自动热替换
   2. 建立jar文件对应的源码

D:/share/tools/struts-2.1.6/src/core/src/main/java，以及xwork对应的源码，以及对应的javadoc location，（更好的方式是建立自己的user-library）

* 1. 浏览struts的目录
  2. 认识eclipse jee的项目部署目录，认识如何修改webapp的context-root，项目copy改名后可能会出现的问题
  3. package explorer – 建立jar的源文件和目标关联，navigator – 观察所有的内容
  4. 认识Action的后缀名，在struts2默认中，带不带action都可以

1. 安装myeclipse 配置
2. 建立WEB项目
3. 加载tomcat（window – preferences – myeclipse – servers – tomcat – 6.x），
4. 加载JDK，Window –preference-Java-install JREs
5. JRE System Libraries 指Java SE的常用集合，建立普通的Java项目均会使用；Java EE 5 libraries 指Java EE的常使用库文件的集合；建立web项目会使用到这个库集合与Java SE的库文件集合；

Reference Libraries 指你项目中所使用的第三方库文件集合，

Web App Libraries 对lib jar包进行动态的管理

在SSH中出现的jar包冲突问题：

编译冲突是由referenced libraries 中的jar包冲突引起，运行期间冲突是由webroot/web-inf/lib下的jar文件冲突引起的，referenced Libraries可以引用lib下的jar包，也可以引用第三方的jar包，referenced Libraries 是编译环境下的jar包，即源文件编写时用。

Lib下的jar包是在运行环境是使用的jar包，即在运行期间使用。

1. 在struts下apps 解压Struts-blank，拷贝Struts-blank的lib文件下的jar文件到工程里的lib目录

struts2-core-2.1.6.jar struts2的核心包

freemarker-2.3.13.jar freemarker是一个模版引擎，一个基于模版生成文本输出的通用工具

ognl-2.6.11.jar 支持ognl表达式

xwork-2.1.2.jar                    xwork的包 由于Struts2是由xwork的延伸 有些类依然关联着 xwork的类

commons-fileupload-1.2.1.jar ,

commons-io.jar     struts的上传下载 两者缺一不可

javassist-3.7.ga.jar 一个编辑Java字节码的类库

1. 拷贝struts.xml 到src，修改struts.xml
2. 改index.jsp 改Hello.jsp
3. 启动tomcat 访问http://locolhost/test/hello.action .action 可以省略
4. <constant name="struts.devMode" value="true" />设置成开发模式false改为true，自己会重新加载
5. 在struts2-core-2.3.12.jar 下找

org.apache.struts2.dispatcher.ng.filter 找

StrutsprepareAndExecuteFilter.class 查看源码，右键jar包属性java source attachment，struts-2.3.12/src/core/src/main/java，查看Javadoc 文档 右键jar包属性 Javadoclocation 选择地址到/struts-2.3.12/docs/struts2-core/apidoc

1. 尖括号提示，window 属性catalog 引入dtd（在lib里解压jar包）文件，第二项改成URI 第三项地址输入：http://struts.apache.org/dtds/struts-2.3.dtd
2. namespace问题 决定了action的访问路径，默认为“”（或者不写），意味着可以接收所有路径的action。namespace可以写为/，/xxx，/xxx/yyy，对应的访问路径为/index.action，/xxx/index.action，/xxx/yyy/index.action，namespace 最好也用模块来进行命名
3. package用来区分重名的情况，也是打包
4. 凡是名字叫success的result ，名字可以不写
5. copy项目时，改属性web Web Context-root 改成新的项目名称，设置不一样的话删除JRE 引入自己的JRE
6. 改Jsp编码 改为Chinese 在window 属性 搜jsp
7. 访问时 项目名称+ namespace + action name
8. struts1中action对象只有一个，struts2 每次访问都会new出一个新的对象
9. 加载xwork源码
10. action不写class 默认调用xwork源码，写class=Java文件的路径（包名+文件名）
11. class问题tomcat会根据action写的class找到对应的Java文件，根据其中的返回值确定调用什么名字的result，从而显示不同的视图，具体视图的返回可以由用户自己定义的Action来决定，具体是根据返回的字符串找到对应的配置项，来决定视图的内容，具体Action的实现可以是一个普通的java类，里面有public String execute方法即可，或者实现Action接口，不过最常用的是从ActionSupport继承，好处在于可以直接使用Struts2封装好的方法
12. path问题是根据action的路径而不是jsp路径来确定，所以使用绝对路径，手动添加绝对路径或者使用myeclipse指定basePath，

head里添加base标签，意思是页面中出现的所有链接默认都是在base指明的地址基础上

**<%**

**String path = request.****getContextPath();**

**String basePath = request.getScheme()+"://"+request.****getServerName()+":"+request.****getServerPort()+path+"/";**

**%>**

**getScheme()取得发出请求的协议方式，getServerName()取得服务器的域名，getServerPort()取得端口号, getContextPath()取得上下文路径**

**<head> <base href="<%=basePath%>"> </head>**

**<body>**

**<a href=”index.jsp”>index.jsp</a>**

**</body>**

1. method问题

继承ActionSupport类，重写execute()方法，即使不重写execute()方法，会默认调用Action中的返回success的方法

Action执行的时候并不一定要执行execute方法  
可以在配置文件中配置Action的时候用method=来指定执行哪个方法

<action name="userAdd" class="com.bjsxt.struts2.user.action.UserAction" method="add">

<a href="<%=context %>/user/userAdd">添加用户</a>

也可以在url地址中动态指定（动态方法调用DMI）（推荐）

<a href="<%=context %>/user/user!add">添加用户</a>

，前者会产生太多的action，所以不推荐使用

输入访问地址 → 交给web.xml 的filter处理 →不写namespace只写工程名 →返回web.xml找默认的设置 →访问index.jsp →链接到其他的jsp

1. 通配符问题

使用\*号，代替可变的部分，用｛1｝｛2｝指代

<action name="student\*" class="test.action.student{1}" method="{1}" >

<result">/My\_{1}.jsp</result>

<action name="\*\_\*" class="test.action.{1}{2}" method="{2}" >

<result>/My\_{2}.jsp</result>

1. 传参数问题

（1）写一个普通的model

url地址中写上要传的参数

写getter和setter

（2）demianmodel域模型，user不需要new 直接写成 private User

user;

建立一个Java文件创建一个User类并写上属性以及对应的getter 和 setter ，在useraction 创建User的对象，写上getUser() 和setUser()；在url传参数

http://localhost/Test /user/user!add?user.name=aa&user.age=100

DTO MVC

（3）实现ModelDriven接口，user需要new，

**public class UserAction extends ActionSupport**

**implements ModelDriven<User>{**

**private User user = new User();**

**public User getModel() {**

**return user;**

**}**

**}**

没有getUser()，setUser() 方法，有

public User getModel(){

return user;

}

Struts2自动维护着一个User对象，所以在action取servlet参数的时候不用request.getParameter()，直接用user.getId()就可以

1. 乱码问题（没有加对应的设置，没出现乱码问题）

查struts2帮助文档的方法

struts2的property配置:Referenced的struts2-core包下org.apache.struts2的default.properties

解决乱码问题在struts.xml中加上<constant name=”struts.i18n.encoding” value=”GBK” />

1. 输入校验问题

分为客户端校验和服务器端校验

客户端校验就是利用JavaScript进行校验

服务器端可以手动校验，也可以使用校验器

类型转换是数据校验的前提

手动校验：

1. 继承ActionSupport重写validate()方法

Action会在调用execute()之前先调用validate()方法，发现不合法输入时，会调用addFieldError()记录一个错误，在执行excute()之前检查有没有fielderror.

validate()方法中加上addFieldError(“name”,”name is error”); 相当于键值对

前者是Filedname后者是ErrorMessage

在jsp文件中添加标签应在文件头写：

<%@taglib uri="/struts-tags" prefix="s" %>

prefix="s" 表示标签的名字

取出对应的错误

<s:fielderror fieldName="name" theme="simple"/></s:fielderror>

fieldname=“name”是property-value 中键的名字

直接在property中取<s:property value="errors.name[0]"/>

直接取property-value中值，因为值是一个数组，errors是property-name

此时更像键值对的嵌套，property-name与property-value是外层，

name=[name is error, name is too long]是内层的键值对

<s:debug></s:debug>

判断字符串匹配问题用name.equals("admin")

1. 重写validateXxx()方法

用于专门校验XXX()方法，就是为对应的方法提供对应的校验方法

重写了validateXxx()，在对Action进行配置时，一定要指定method属性，否则系统默认调用execute()，如：处理逻辑方法为regist() 则对应的校验方法validateRegist()

如果既有validate()的校验器，又有validateXxx()，会先执行自己的校验器，再执行validate()

内置校验器

使用内置校验器需要配置校验文件，命名规则为：Actionname-validation.xml(不包含包名)

字段校验器

<validators>

<field name=”name”>

<field-validator type =”校验器类型”>

<message>姓名不能为空</message>

</field-validator>

</field>

</validators>

非字段校验器

<validators>

<validator type =”校验器类型”>

<param name=”字段名”>name</param>

<message>姓名不能为空</message>

</validator>

</validators>

必填校验器：required

字符串长度校验器：stringlength

整数校验器：int 划定一个整数的范围

日期校验器：date 划定一个日期范围

表达式校验器：expression

字段表达式校验器：fieldexpression

邮件地址校验器：email 必须满足邮件地址规则

网址校验器：url 必须是合法的网址

转换校验器：conversation 某个字段是否发生转换错误

正则表达式校验器：regex 使用正则表达式校验一个字符串字段

1. 接收web元素（前两种取得map类型的，后两种取得真实类型的，也就是http类型）第二种常用，用控制反转就需要加setter，用自己取的方式不用加

第一种方式的原理：

通过ActionContext获得ServletAPI（间接访问ServletAPI）

获得ActionContext

ActionContext ac= ActionContext.getContext()

ActionContext常用方法：

Object get(Object key)通过参数key查找当前ActionContext中的值

Map getApplication() 返回一个application级的map对象

static ActionContext getContext()获得当前线程的ActionContext对象

Map getParameters()返回包含HttpServletRequest参数的map对象

Map getSession() 返回map类型的httpsession 对象

void put(Object key, object value)向当前的ActionContext对象存入名值对信息

void setApplication(Map application) 设置application的上下文

void setSession(Map session) 设置一个Map类型的session

第二种方式的原理：

控制反转（IoC）将设计好的类交给系统控制，而不是在自己的内部控制

要先实现接口

第三种方式的原理：

ServletActionContext辅助类间接访问，这是非IoC方式

其静态方法有：getPageContext() 、getRequest()、getResponse()

getServletContext()

（1）第一种方式（自己拿出来request session application）

**private Map request;**

**private Map session;**

**private Map application;**

**类LoginAction1的构造方法：**

**//action不指明调用那个方法会默认调用execute()**

**public LoginAction1() {**

**request = (Map)ActionContext.getContext().get("request");**

**session = ActionContext.getContext().getSession();**

**application = ActionContext.getContext().getApplication();**

**}//getContext()取得执行的上下文环境**

**public String execute() {**

**request.put("r1", "r1");**

**session.put("s1", "s1");**

**application.put("a1", "a1");**

**return SUCCESS;**

**}**

**取得对应的值**

**<s:property value="#request.r1"/> | <%=request.getAttribute("r1") %>**

**<s:property value="#session.s1"/> | <%=session.getAttribute("s1") %>**

**<s:property value="#application.a1"/> | <%=application.getAttribute("a1") %>**

**<s:property value="#attr.a1"/><br />**

**<s:property value="#attr.s1"/><br />**

**<s:property value="#attr.r1"/><br />**

（2）第二种方式：DI/IOC 依赖注入/控制反转（等待容器给注入）

**public class LoginAction2 extends ActionSupport implements RequestAware,SessionAware, ApplincatioAware {**

**private** Map<String, Object> request;

**private** Map<String, Object> session;

**private** Map<String, Object> application;

**public** String execute() {

request.put("r1", "r1");

session.put("s1", "s1");

application.put("a1", "a1");

**return** *SUCCESS*;

}

**public** **void** setRequest(Map<String, Object> request) {

**this**.request = request;

}

**public** **void** setSession(Map<String, Object> session) {

**this**.session = session;

}

**public** **void** setApplication(Map<String, Object>application){

**this**.application = application;

}

（3）第三种方式：（自己取）

**private** HttpServletRequest request;

**private** HttpSession session;

**private** ServletContext application;

**public** LoginAction3() {

request = ServletActionContext.*getRequest*();

session = request.getSession();

application = session.getServletContext();

}

**public** String execute() {

request.setAttribute("r1", "r1");

session.setAttribute("s1", "s1");

application.setAttribute("a1", "a1");

**return** *SUCCESS*;

}

（4）第四种方式：（依赖注入）

**public** **class** **LoginAction4 extends** ActionSupport **implements** ServletRequestAware {

**private** HttpServletRequest request;

**private** HttpSession session;

**private** ServletContext application;

**public** String execute() {

request.setAttribute("r1", "r1");

session.setAttribute("s1", "s1");

application.setAttribute("a1", "a1");

**return** *SUCCESS*;

}

**public** **void** setServletRequest(HttpServletRequest request) {

**this**.request = request;

**this**.session = request.getSession();

**this**.application = session.getServletContext();

}

1. strut.xml包其他的xml文件

格式：< include file=”login.xml” />

1. 设置默认的action

<package name=”default” namespace=”/” extends=”struts-defult”>

<default-action-ref name=”index”> </ default-action-ref >

# Result

1. result类型

格式：<result type=”类型”> /\*.jsp </result>

dispatcher默认类型:服务器端跳转，不能跳到action

redirect 客户端跳转， 不能跳到action

chain 服务器端跳转 跳到action

redirectAction 客户端跳转， 跳到action

注意：跳转到Action 不写“/” 跳转到的Action必须定义了

Action跳转采用加参数的方式：

<param name = “actionName”>dashboard</param>

<param name=”namespace”>/secure</param>

<result type=*"chain"* name=*"modify"*>

<param name = *"actionName"*>channel</param>

<param name = *"method"*>list</param>

</result>

1. global result 全局结果集

定义全局的result

形式： **<global-results>**

**<result name="mainpage">/main.jsp</result>**

**</global-results>**

当一个局部结果与全局结果重名时，那么局部结果会覆盖全局结果

可以通过extends继承其他package，以获得global result

1. 动态结果

通过${r}定义动态结果，在Action里保存一个属性type，通过URL传递type值，$ 是取得值栈的标志

**public String execute() throws Exception {**

**if(type == 1) r="/user\_success.jsp";**

**else if (type == 2) r="/user\_error.jsp";**

**return "success";**

**}**

1. 传参数

客户端跳转的时候才需要传参数，一次request发给Action才会建立一个值栈，发起一个新的request会建立一个新的值栈，

**<action name="user" class="com.bjsxt.struts2.user.action.UserAction">**

**<result type="redirect">/user\_success.jsp?t=${type}**

**</result>**

**</action>**

向UserAction里传type=1，用redirect跳转，需要向jsp页面传参数，由于jsp没有值栈，所以靠在值栈取值的方式取不出来

<s:property value=*"t"*/>

需要靠取参数的方式，也就是在stack context 取值

<s:property value=*"#parameters.t"*/>

看标签：

index.xml Getting Start 下的guides Tag Developers Guide

Struts Tags Tag Reference Generic Tag Reference

# OGNL

1. Ognl意思是对象视图导航语言 cat.friend.name
2. value里边对应的是OGNL表达式，想初始化domain model，可以自己new，也可以传参数值（这种方式中只有传值才会给你初始化），但这时需要保持参数为空的构造方法

OgnlAction 里面 必须写getUser() 和setUser() 否则取不到User的参数

1. 没有构造方法时编译时会自动加上空的构造方法，如果有非空的构造方法，编译时就不会加空的构造方法，所以要想初始化，自己必须加上空的构造方法
2. 访问值栈中的action的普通属性:

**username = <s:property value="username"/>**

**访问值栈中对象的普通属性(get set方法)：**

**<s:property value="user.age"/>**

**| <s:property value="user['age']"/>**

**| <s:property value="user[\"age\"]"/>**

**| wrong: <s:property value="user[age]"/>**

**<s:property value="cat.friend.name"/>**

**访问值栈中属性的普通方法：**

**<s:property value="password.length()"/>**

**访问值栈中对象的普通方法：**

**<s:property value="cat.miaomiao()" />**

**访问值栈中action的普通方法：**

**<s:property value="m()" />**

**访问静态方法设置常量**

**<constant name="struts.ognl.allowStaticMethodAccess" value="true"></constant>**

访问静态属性或方法格式：<property value=“@路径.类名@方法名/属性名”>

**访问静态方法：**

**<s:property value="@com.bjsxt.struts2.ognl.S@s()"/>**

**访问静态属性：**

**<s:property value="@com.bjsxt.struts2.ognl.S@STR"/>**

**访问Math类的静态方法：两个@@**

**<s:property value="@@max(2,3)" />**

**访问集合**

**set里没有顺序**

**tostring的用法**

**访问List:<s:property value="users"/> //users 为集合的名字**

**访问List中某个元素:<s:property value="users[1]"/>**

**//把users的某个属性装在一个数组里**

**访问List中元素某个属性的集合:<s:property value="users.{age}"/>**

**访问List中元素某个属性的集合中的特定值:**

**<s:property value="users.{age}[0]"/>**

**| <s:property value="users[0].age"/>**

**访问Set:<s:property value="dogs"/> //dogs为集合的名字**

**访问Set中某个元素:<s:property value="dogs[1]"/>**

**访问Map:<s:property value="dogMap"/> //dogmap 为集合的名字**

**访问Map中某个元素:<s:property value="dogMap.dog101"/>**

**| <s:property value="dogMap['dog101']"/>**

**| <s:property value="dogMap[\"dog101\"]"/>**

**访问Map中所有的key:<s:property value="dogMap.keys"/>**

**访问Map中所有的value:<s:property value="dogMap.values"/>**

**访问容器的大小：<s:property value="dogMap.size()"/>**

**| <s:property value="users.size"/>**

**过滤**

**?#表示过滤条件**

**投影(过滤)：<s:property value="users.{?#this.age==1}[0]"/></li>**

**^#表示取出users这个集合里age>1 的第一个元素的age集合**

**投影：<s:property value="users.{^#this.age>1}.{age}"/></li>**

**$#表示取出users这个集合里age>1 的最后一个元素的age集合**

**投影：<s:property value="users.{$#this.age>1}.{age}"/></li>**

**判断返回true或false**

**投影：<s:property value="users.{$#this.age>1}.{age} == null"/></li>**

**访问值栈的object**

**//[0]表示从第零个一直访问到栈底**

**[]:<s:property value="[0].username"/>**

1. #号的用法

（1）访问OGNL上下文和ActionContext上下文 parameter request

session application

（2）用于过滤和投影集合

（3）用于构造Map #{‘book1’:’23’ , ’book2’:’55’}

1. %的用法

当标签属性为字符串时，把字符串当作OGNL表达式

1. $的用法
2. 在国际化资源文件中使用OGNL表达式
3. 在Struts2配置文件中使用OGNL表达式

# Tags

逻辑控制标签

1. if / elseif / else

<s:set name=”dengji” value=”85” />

<s:if test=”#dengji>=90”>优秀</s:if>

<s:elseif test=”#dengji>=80”>良好</s:elseif>

<s:elseif test=”#dengji>=60”>及格</s:elseif>

<s:else>不及格</s:else>

1. iterator 对list、set、数组、Map遍历

value 指定要输出的集合

id指定集合中元素的ID属性

status 指集合中元素的status属性，利用该属性可以实现一些很实用的功能，每次迭代都会产生一个ItetatorStatus实例对象。

可以如下使用：

#status.odd 返回当前迭代元素的索引值是否为奇数

#status.even 返回当前迭代元素的索引值是否为偶数

#status.count 返回当前迭代元素的个数

#status.first 返回当前迭代元素的是否为第一个元素

#status.last 返回当前迭代元素的是否为最后一个元素

#status.index返回当前迭代元素的索引值

<s:iterator value=”{‘a’, ’b’, ’c’ }” id=”word” status=”st”>

<s:if test=”#st.odd”>style=”color；red” </s:if>

<s:property value=”word” />

<s:property value=”#st.index” />

</iterator>

如果st是一个对象，迭代st的属性<s:property value=”name”/>

<s:iterator value=”{‘a’:’20’ , ’b’:’30’ , ’c’:’40’ }” status=”st”>

<s:if test=”#st.odd”>style=”color；red” </s:if>

<s:property value=”key” />

<s:property value=”value” />

</iterator>

1. append、 merge都是把两个集合合并， id指新集合的名字，子标签表示子集合，只是合并后元素的组织顺序不同，append输出顺序与添加顺序一致，merge会把这些元素交织在一起

<s:append id=”newlist”>

<s:param value”{‘元素1’,’元素2’,’元素3’}” />

<s:param value”{‘元素4’,’元素5’,’元素6’}” />

</s:append>

1. generator

将一个字符串按指定的分隔符分隔成多个子串

count生成集合中元素的总数

val 指定被解析的字符串

separatpr 指定分隔符

<s:genetator val=” ’a,b,c,d’ ” separatpr=”,” >遍历</s:genetator>

1. subset从一个集合截取一个新的集合

source 指定原集合 count 指定子集合元素个数 start 起始位置

decider 是否选中当前元素

<s:subset source=”{‘a’, ’b’, ’c’}” start=”1” count=”3”>

1. sort 对指定集合进行排序，排序规则要实现comparator接口

<s:bean id=”mycomparator” name=”com. mycomparator” />

<s:sort source=”{‘asd’,’s’,’dfdgg’}” comparator=” mycomparator”>

遍历

</s:sort>

数据标签

1. Action在JSP页面直接调用Action，包括结果包含和向Action传参数问题
2. bean 用于在当前页面创建javaBean实例对象，可以通过param为实例设置属性值

name 要实例化的javabean的实现类

id 可以通过该ID访问javabean实例

s:bean 开始时产生一个对象，压到栈顶，执行完从栈顶移除

1. debug标签
2. include

value 被包含的JSP或Servlet等资源文件

Id 指定该标签的引用

可以通过param向被包含的页面传参数

<s:include value=”includefile.jsp”>

<s:param name=”user” value=” ‘WHM’ ” />

<s:include>

1. Property

default 指定属性值为空时输出的值

escape指定是否显示标签代码

value 指定输出的属性值

Value 的type为object

<s:property value=”username”/> 此时的username默认解析为ognl表达式

<s:property value=” ’username’ ”/>此时username解析为字符串

1. Set

name 定义新变量的名字

scope 定义使用范围 application session request response page action

value 指定变量的值

设置属性值：<s:set name=”adminname” value=”username” />其范围默认为request和actioncontext，value 的返回值为object类型

从request中取值：<s:property value=”#request.adminname”/>

从actioncontext中取值：<s:property value=”#adminname”/>

1. url

value 和action 都是指定url地址，只不过后者是一个Action

Id是该引用的ID

后面可以使用该ID：<s:a href=”%{ID}”>text</s:a>

1. date指定日期输出格式

format指定日期格式

nice是否输出指定时间和当前时间的时差（format和nice不能同时指定）

name 要格式化的日期值

<s:date name=”” format=”yyyy-MM-dd” />

表单的UI标签

可以通过UI标签、form表单标签设置主题

表单标签的通用属性

label 设置表单元素的label属性

labelPosition 设置表单元素label显示的位置

name设置表单元素的name 与Action中的属性名相对应

value设置表单元素的值

1. checkboxlist 一次创建多个复选框

listKey 为复选框的值 listValue为标签的显示的文字

<s:form >

<s:checkboxlist list=”{‘jsp’,’servlet’,’struts2’}” name=”skills”

label=”熟悉技术” labelposition=”top” >

</s:checkboxlist>

</s:form>

1. radio 一次创建多个单选按钮

与checkboxlist用法相同

<s:form >

<s:radio list=”{‘jsp’,’servlet’,’struts2’}” name=”skills”

label=”熟悉技术” labelposition=”top” >

</s:radio>

</s:form>

1. combobox生成两个元素，分别是单行文本框和下拉列表，下拉列表中内容会自动显示在文本框中
2. select用法与combobox类似

<s:select name=”books\_map” label=”图书列表”

list=”#{‘ssh’:’sshc从入门到精通’,’jsf’:’jsf从入门到精通’}”/>

<s:bean name=”com.BookService” id=”bs”/>

<s:select name=”book\_java” label=”图书列表”

list=”#bs.books” listkey=”name” listvalue=”name” />

1. doubleselect生成相关联的下拉列表，第一个下拉列表的值改变会引起第二个下拉列表的改变，通过map实现这种关联

<s:set name="pc" value="# {'北京市':{'朝阳区','海淀区','大兴区'},'上海市':{'浦东区','南汇区','徐汇区'} }" />

<s:doubleselect name="city" list="#pc.keySet()" doubleName="area" doubleList="#pc[top]" label="所在城市"></s:doubleselect>

使用doubleselect时，必须放在form中且form表单指定name属性

1. optgroup 选项组

与select结合使用 label表示组名，不能被选中

1. datetimepicker生成一个动态日历
2. token用来防止多次提交表单，要结合token拦截器使用
3. updownselect 带有上移，下移，全选按钮的列表框
4. optiontransferselect 生成两个下拉列表，分别支持上移，下移，而且支持左右移动

非表单UI标签

1. actionerror和actionmessage分别输出getActionRrrors()和getActionMessage()返回的错误信息
2. component 使用该标签自定义组件
3. tree和treenode生成树状结构

Value stack 相当于根不加#号访问

Value context 存的是map 访问需加#

# BBS

数据库有关内容《54\_尚学堂马士兵\_Struts2\_BBS2009准备\_1》

Struts2命名规则

加载ext文件时复制ext-4.0.7-gpl到工程web-root下的admin文件夹

右键.js文件，属性，把编码格式改成UTF-8

访问<http://localhost/Ext/admin/docs/index.html>没有web-root那层路径

ext需部署到服务器上才能读docs下的index.html examples下的index.html

theme和ext有什么区别

只输入工程名，不写namespace默认调用下面的设置

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

BBS搭建过程

用现成的ext

建立对应的action和jsp，该struts.xml配置文件，建自己的index.jsp(即主页)前台的，后台的

链接数据库，备份自己的sql语句

建立model

建service

建action，链接model和service

用到ext的地方尽量都把编码形式改成UTF-8

# 声明式的异常处理

出现异常不是方法自己处理，而是抛出，由统一的接口进行处理

在容易出异常的action里加上，result=error，exception类型为什么

在struts.xml会出异常的action中加如下内容

<exception-mapping result="error" exception="java.sql.SQLException" />

<result name="error">/error.jsp</result>

配置全局的异常映射和处理，先写全局的结果处理在写异常处理

**<global-results>**

**<result name="error">/error.jsp</result>**

**</global-results>**

**<global-exception-mappings>**

**<exception-mapping result="error" exception="java.lang.Exception">**

**</exception-mapping>**

**</global-exception-mappings>**

异常的处理是通过拦截器实现的

设置默认的action <default-action-ref name=”index”> 不会执行对应的action的class ，而是直接跳到<result>/\*\*.jsp</result>

改web.xml配置把默认的

<welcome-file-list>

<welcome-file>index</welcome-file>

</welcome-file-list>

所以应避免找不到对应的action的情况

# I18N

**public static void main(String[] args) {**

**ResourceBundle res = ResourceBundle.getBundle("app", Locale.CHINA);**

**System.out.println(res.getString("welcome.msg" ));}**

app\_zh\_CN.properties 内容：welcome.msg=\u6B22\u8FCE\u4F60\uFF01

国际化：

action级别的 action的名字\_zh\_CN.properties，用

<s:property value=”getText’login.username’” />

包级别的 package\_zh\_CN.properties

application级别的 自定name\_zh\_CN.properties

在struts.xml 里配置 <constant name=”struts.custom.i18n.resources” value=”自定的name” > </constant>

可以使用占位符动态指定参数：.properties 里 welcome.msg=welcome:{0}

**<s:text name=”welcome.msg” >**

**<s:param value=”username”></s:param>**

**</s:text>**

动态的语言切换 在地址栏加上request\_locale=en\_US

# 拦截器

1. 面向切面编程：通过需求从不相关的类中分离出来，使得很多类共享一个行为，一旦发生变化，不必修改很多类，只需要修改这个行为即可。
2. 配置拦截器：(定义拦截器和拦截器栈都是放在interceptors标签中)

（1）拦截器

**<interceptors>**

**<interceptor name=” ” class=” ” />**

**</interceptors>**

（2）拦截器栈：将多个拦截器合并到一起组成一个拦截器栈

**<interceptor-stack name=””>**

**<interceptor-ref name=”之前配置好的拦截器的名字” />**

**<interceptor-ref name=”之前配置好的拦截器栈的名字” />**

**</interceptor-stack>**

1. 为拦截器指定参数；

（1）单独为拦截器定参数

**<interceptor-stack name=””>**

**<interceptor-ref name=”之前配置好的拦截器的名字” >**

**<param name=””>参数值</param>**

**</interceptor-ref>**

**</interceptor-stack>**

（2）为整个拦截器栈定参数

**<interceptor-stack name=””>**

**<param name=””>参数值</param>**

**<interceptor-ref name=”之前配置好的拦截器名字” >**

**</interceptor-ref>**

**</interceptor-stack>**

使用拦截器时指定的参数值会将覆盖定义拦截器时指定的参数

1. 使用拦截器

**<action name=”” class=”” >**

**<result name=””>success.jsp </result>**

**<interceptor-ref name=”拦截器名” />**

**<interceptor-ref name=”defaultStack” />**

**</action>**

一旦开发者为包中的某个action指定了某个拦截器，则默认的拦截器将不会起作用，那么就需要手动配置该默认的拦截器

1. 配置默认拦截器

**<default-interceptor-ref name=”” />**

<interceptors>定义拦截器，可以包含子元素<interceptor>和<interceptor-stack>,分别用来定义拦截器和拦截器栈

<interceptor-ref>用来使用拦截器或拦截器栈

intercept()是拦截器的核心方法，实现具体的拦截操作，其ActionInvocation类型的对象可获得Action对象，调用Action的invoke()方法，将控制权交给下一个拦截器，或者Action的execute()方法

请求先提交到Action，在execute()之前调用拦截器，执行完操作，再执行拦截器的invoke()方法，将控制权交给下一个拦截器，或者Action的execute()方法。若交给execute() ,返回字符串给最近调用的一个拦截器，然后层层向外传，最后在struts.xml里配置的Action找到对应的result，进行页面跳转

1. 自定义拦截器

继承AbstractInterceptor抽象类，重写intercept()方法

1. 方法过滤拦截器

继承MethodFilterInterceptor抽象类，重写doIntercept()方法

在拦截器的使用时可指定哪些方法是否被拦截

**<interceptor-ref name=””>**

**<param name=”excludeMethods”>execute </param>**

**</interceptor-ref>**

1. 拦截器的拦截顺序

按照从前往后实行到execute()再从后往前反向执行

1. 拦截器在页面重复提交时的应用：

（1）表单用post方式提交

（2）token拦截器

# 类型转换

默认的转换器

自定义转换器

继承DefaultTypeConverter，重写converterValue()方法

也可以继承StrutsTypeConverter() 重写convertFromString()和convertToString()

注册转化器，分局部和全局

局部：xxxAction-conversion.properties

p(属性名)=converter

全局： xwork-conversion.properties

com.xxx(类名)=converter

遇上非常复杂的用request.setAttribute()

关于类型转换的异常处理，struts2的拦截器可以处理

# 值栈value stack

action里定义个属性（有对应的getter和setter），放在Value Stack Contents中，可以直接用property标签value取

放在ActionContext中的属性方法，放在Stack Context，在用property标签取数据时需要加上#

struts2里的form的Action直接写Action的名字，不用写路径

Struts里的配置的Action是在工程名以下，跟包名没关系

basePath对form的Action不起作用