Struts 2基础

1. Struts2.xml

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

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

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

<!--false改成true 修改配置是不需要重启服务器 -->

<package name=*"user"* namespace=*"/user"* extends=*"struts-default"*>

<action name=*"user"*>

<result>/index.jsp</result>

</action>

</package>

</struts>

1. Web.xml

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

<web-app version=*"2.5"*

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

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

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*>

<filter>

<filter-name>struts2</filter-name> <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<welcome-file-list>

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

</welcome-file-list>

</web-app>

1. <http://localhost:8888/struts2/user/user.action>

<http://localhost:8888/struts2/user/user>

配置中全局变量

配置struts中放在src下，没有写Action都没有出错

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

<!--false改成true 修改配置是不需要重启服务器 -->

<package name=*"user"* namespace=*"/"* extends=*"struts-default"*>

**namespace=*"/"*** *一般定义和模块名一样（含义是namespace的空间命名可以随意命名，但不写的话就会启用默认的命名空间，即你的项目名为命名空间，写的话要以“/”来开头；在你JSP页面里面的form里要注明namespace，不然程序会报找不到命名空间的错误。*

*其中在其子路径下度可以找到 页面index.jsp*

*struts2/user/user/user）*

**extends=*"struts-default" 必须继承 找到核心包中struts-default.xml中需要复制到配置中，当全局变量是，也需要继承***

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

<result **name=*"ok"***>/index.jsp</result> 其中为跳转信息

</action>

</package>

**<result name=*"ok"*>**

ActionSupport 查看该类

**import** com.opensymphony.xwork2.ActionSupport;

**public** **static** **final** String *SUCCESS* = "success";

*NONE* = "none"; *ERROR* = "error"; *INPUT* = "input"; *LOGIN* = "login";

当出现以上 字段，name可以不写

**全局跳转：**

<package name=*"global"* extends=*"struts-default"*>

<global-results>

<result name=*"ok"*>/index.jsp</result>

</global-results>

</package>

<package name=*"user"* namespace=*"/user"* extends=*"global"*>

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

</action>

</package>

**实现Action有三种方式：**

1）、excete

**public** **class** UserAction{

**public** String delete(){

System.*out*.println("1111");

**return** "success";

}

**public** String add(){

System.*out*.println("addddddddddddddddd");

**return** "ok";

}

}

配置

<package name=*"user"* namespace=*"/user"* extends=*"struts-default"*>

<action name=*"delete"* class=*"com.wuliang.action.UserAction"* method=*"delete"*>

<result >/index.jsp</result>

</action>

<action name=*"add"* class=*"com.wuliang.action.UserAction"* method=*"add"*>

<result name=*"ok"*>/login.jsp</result>

</action>

</package>

<http://localhost:8888/struts2/user/add!add>

2）、实现Action

**Action中实现** UserAction **implements** Action{

<action name=*"user"* class=*"com.wuliang.action.UserAction"* method=*"add"*>

只需要在配置中加 method=“方法名

**public** String add(){

System.*out*.println("----------add------------");

**return** "ok";

}

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

// **TODO** Auto-generated method stub

System.*out*.println("----------execute------------");

**return** "success";

}

配置 其中execute方法不用写mothod方法 user/user 就可以

<action name=*"add"* class=*"com.wuliang.action.UserAction"* method=*"add"*>

<result name=*"ok"*>/login.jsp</result>

</action>

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

<result >/404.jsp</result>

</action>

3）继承ActionSupport

**public** **class** UserAction **extends** ActionSupport{

**public** String delete(){

System.*out*.println("----------delete------------");

**return** "success";

}

**public** String add(){

System.*out*.println("----------add------------");

**return** "ok";

}

}

struts2/user/User!delete {1}占位符User

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

<result >/index.jsp</result>

<result name=*"ok"*>/login.jsp</result>

</action>

缺点是需要写的<result name=*"ok"*>/login.jsp</result>，阅读性差

**老师说用第二种方法，但是我看，第一种方法完全可以取代第二种方法？？？**

**实现ActionFrom 接收表单数据的三中方法**

1. **直接加属性 其中set get方法**

**如下**

**private** String username;

**private** String password;

**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;

}

**public** String delete(){

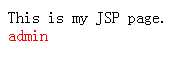
System.*out*.println(username);

System.*out*.println("----------delete------------");

**return** "success";

}

**user/delete!delete?username=admin**



**即set方法是传进去，get是拿出来**

1. **写vo**

**Vo中有**

**public** **class** User {

**private** String username;

**private** String password;

**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;

}

}

Action

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

**public** User getUser() {

**return** user;

}

**public** **void** setUser(User user) {

**this**.user = user;

}

传入user/delete!delete?user.username=admin

得到 ${user.username }

1. **ModelDiver**

**实现**ModelDriven 其中是在第二个方法中修改的不在需要user.username传参与接收参数

**public** **class** UserAction **implements** ModelDriven<User>{

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

**public** String delete(){

System.*out*.println(user.getUsername());

System.*out*.println("----------delete------------");

**return** "success";

}

**public** User getModel() {

// **TODO** Auto-generated method stub

**return** user;

}

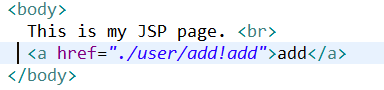
}

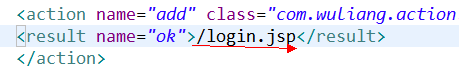
**适用与第三中方法**

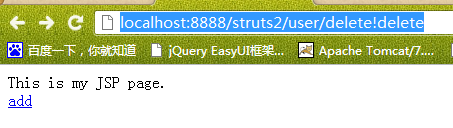
**Struts2中路径问题**

**/ 根路径** [**http://localhost:8888/struts2/**](http://localhost:8888/struts2/)

**./ 当前路径**



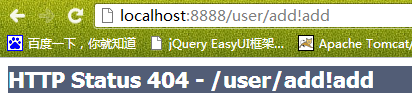






**出现正确的**

**当在页面上 /user/add!add ,就会出现错误**



**当其他的都正确是加入会出错**



**在Struts2中用高版本的jsp，**

String path = request.getContextPath();

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

这个jsp会自动在 <a href=*"user/add!add"*>add</a>加上basePath，就是[**http://localhost:8888/struts2/**](http://localhost:8888/struts2/)

**而且配置中只用 /user 既可**

**分布式开发：**

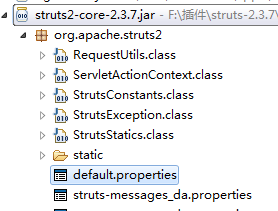
<struts>

<include file=*"wuliang.xml"*></include>

</struts>

**乱码问题解决：**

**各种常量放置地方**



**没有配置是乱码，配置**

<constant name=*"struts.i18n.encoding"* value=*"UTF-8"*></constant>

还是乱码？？？？？

**更改跳转方式：**

Struts2中默认是 forward跳转

<result type=*""*>/index.jsp</result>

type=*"redirect"*

*forward与redirect主要区别在与forward不会改变地址栏而redirect中接收不了参数*

*Action跳转到Action出现问题*

*1.forward 页面*

*2.redirect 页面 + Action*

<result type=*"redirect"*>/user/add!add</result>

*3.chain Action*

<result type=*"chain"*>

<param name=*"namespace"*>/user</param>

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

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

</result>

*4.redirectAction Action*

<result type=*"redirectAction"*>

<param name=*"namespace"*>/user</param>

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

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

</result>

*5.stream 下载 （图片 验证码 二进制数）*

6.plaintext 源代码

在实际

**public** **class** UserAction **extends** **ActionSupport{**

**public** String delete(){

this.addActionError("用户名小于6");

System.*out*.println("----------delete------------");

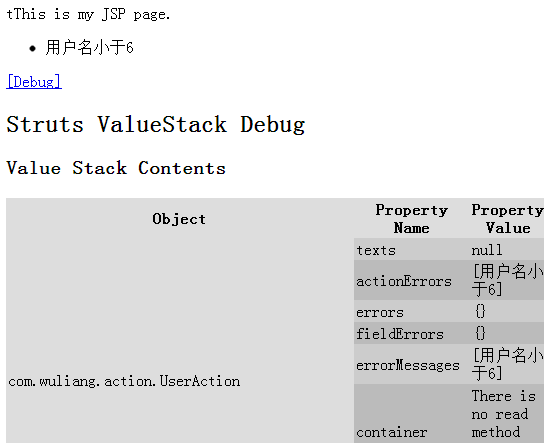
**return** "success";

}

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

<s:actionerror/>

<s:debug></s:debug>



怎么得到request session application 四种方法

由于struts2中有自定义的request等，用Map<String,Object> 来表示，所以

第一种方法：

**public** **class** UserAction **implements** Action{

Map<String,Object> request;

Map<String,Object> session;

Map<String,Object> application;

**public** UserAction(){

request = (Map<String, Object>) ActionContext.*getContext*().get("request");

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

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

}

**public** String delete(){

request.put("user", "看看");

System.*out*.println("----------delete------------");

**return** "success";

}

<font color=*"red"*>${user }</font>

第二种方法：

**public** **class** UserAction **implements** RequestAware,SessionAware,ApplicationAware{

Map<String,Object> request;

Map<String,Object> session;

Map<String,Object> application;

**public** String delete(){

request.put("user", "看11看");

System.*out*.println("----------delete------------");

**return** "success";

}

**public** **void** setApplication(Map<String, Object> arg0) {

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

}

**public** **void** setSession(Map<String, Object> arg0) {

// **TODO** Auto-generated method stub

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

}

**public** **void** setRequest(Map<String, Object> arg0) {

// **TODO** Auto-generated method stub

**this**.request = (Map<String, Object>) ActionContext.*getContext*().get("request");

}

}

？？？

怎么获得HttpServletRequest requet；

第三种：得到真实的request session application

使用第一种方法，使用构造方法

**public** **class** UserAction {

HttpServletRequest request;

**public** UserAction(){

request = ServletActionContext.*getRequest*();

HttpSession session = request.getSession();

ServletContext application = session.getServletContext();

}

**public** String delete(){

request.setAttribute("user", "aaa");

System.*out*.println("----------delete------------");

**return** "success";

}

}

第四种：得到真实request等 实现ServletRequestAware

**public** **class** UserAction **implements** ServletRequestAware{

HttpServletRequest request = ServletActionContext.*getRequest*();

**public** String delete(){

request.setAttribute("user", "aaa");

System.*out*.println("----------delete------------");

**return** "success";

}

**public** **void** setServletRequest(HttpServletRequest arg0) {

// **TODO** Auto-generated method stub

**this**.request = request;

}

}

一般来说用第二种，第一种耦合型太强，修改成其他的模板难，第三种与第四种都是得到实际request，但是我们可以看出页面上得到是map中的request，所以需要转换，效率低