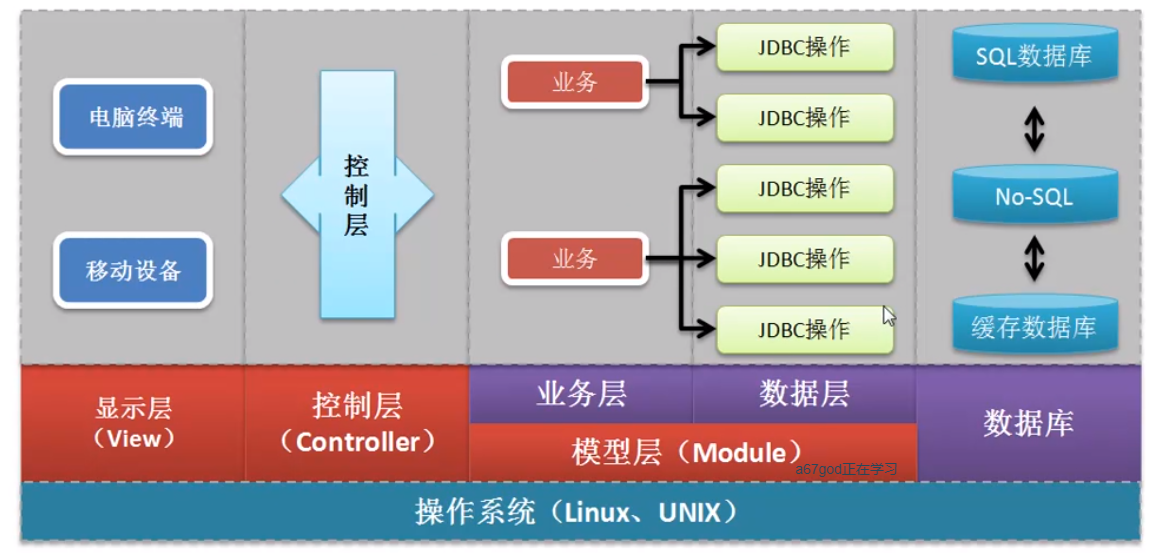
1. 传统的MVC



1. 前端页面：html、css、js等。
2. 控制层：servlet
3. 模型层：纯粹的Java程序类

使用传统的MVC，开发中最难控制的就是控制层，对于控制层的功能：

* 可以接收参数，由于整个vo结构的发展，所以控制层接收的数据往往需要转换为vo类,整个过程需要通过反射完成。

|- request.getParamaterNames()、一序列的Class、Method、Filed类的调用。

* 数据验证处理：必须要服务器端和客户端一起验证，要保证到业务层的数据安全。
* 需要自己实现资源文件的定义（读取），包括国际化的实现。
* Session处理时需要依靠cookie机制，若果客户端禁用的cookie，则控制层需要保证session的正常使用（传递一个JSESSIONID的参数，但每次传递就比较麻烦）。
* Servlet的实例化只有一次，但不同用户的处理有多次，那么久需要合理的安排vo类对象的定义。
* 很多时候的开发，我们不需要知道request和resopne.
* 文件的上传处理应该跟简单一些。

以上的问题虽然经验丰富的开发可以解决的很好，但从实质上来讲这样太耗费时间了。

1. Struts2开发框架

1、MVC和Struts的关系：

MVC就是一个设计模式，Struts是一个实现了MVC的开发框架。MVC是相当于一个标准，Struts是这个标准的具体实现。

搭建框架：

导入jar包

jar包下载地址：http://www.apache.org/官网中选择struts，然后点击download下载。将jar包导入到WEB-INF下的lib文件目录下。

asm-5.2、asm-commons-5.2、asm-tree-5.2：反射的替代方案。asm是小巧便捷的java字节码操控框架，它能方便的动态生成和改造java代码

commons-fileupload-1.3.3：上传文件

commons-io-2.5：对本地文件、流进行操作

commons-lang3-3.6：基础文件包

commons-logging-1.1.3：日志包

freemarker-2.3.23：生成各种文本：html、xml、rtf、java源代码等

javassist-3.20.0-GA：使java字节码操控更加简便，是一个编译java字节码的类库，提高效率

log4j-api-2.8.2：日志

ognl-3.1.15：struts2独有的标签库

struts2-core-2.5.13：struts2的核心包

复制struts.xml文件到src文件夹下任意位置

在没网络情况下，无法下载struts-2.5.dtd文件，从而无法进行代码编写提示。这种情况需要在IDE中手动配置，Window-->Preferences-->XML Catalog-->add-->key type：选择URI；key：即下载struts-2.5.dtd文件的地址，也是struts.xml文件中DOCTYPE配置的地址-->File System：加载本地struts-2.5.dtd文件-->点击添加

在web.xml文件中加入过滤器

复制代码

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

<!-- 设置代码提示所需文档位置 -->

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd" version="3.1">

　　<display-name>StudyStruts2</display-name>

　　<filter>

　　　　<filter-name>StrutsFilter</filter-name>

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

　　</filter>

　　<filter-mapping>

　　　　<filter-name>StrutsFilter</filter-name>

　　　　<url-pattern>/\*</url-pattern>

　　</filter-mapping>

</web-app>

复制代码

注意：这里加入过滤器是filter标签，不是servlet标签

1. 观察web.xml:

<**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**>

配置文件中使用的过滤器，之所以使用过滤器有两个原因：

* 过滤器可以像普通servlet那样处理用户请求，使用RequestDispatcher实现跳转；
* Struts 1.x用的是servlet处理请求,所以Struts2.x 为了不同filter（登录检测麻烦，只有页面登录检查可以使用过滤器，服务器端必须使用自己的拦截器）。

1. 第一个Struts2.x 程序
   1. 创建Class程序：

**public class** EchoAction **extends** ActionSupport {*//必须继承ActionSupport  
 //用户传递的参数名称* **private** String **msg**;  
  
 **public** String getMsg() {  
 *//顶替了msg=requeest.getParameter("msg")功能* **return msg**;  
 }  
  
 **public void** setMsg(String msg) {  
 *//页面读取的时候使用* **this**.**msg** = msg;  
 }  
  
 @Override  
 **public** String execute() **throws** Exception {*// 执行操作* **this**.**msg** = **"ECHO : "** + **this**.**msg** ;  
 **return "echo.page"**; *//返回路径映射的key* }  
}

* 1. 在Strus中配置路径映射：

*<?***xml version="1.0" encoding="UTF-8"***?>***<!DOCTYPE struts PUBLIC  
 "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"  
 "http://struts.apache.org/dtds/struts-2.5.dtd"*>***<**struts**>  
 *<!--所有的Struts2x 都要求设置命名空间-->* <**package name="root" namespace="/"**>  
 *<!--配置action程序的名称的及对应的路径-->* <**action name="EchoAction" class="cn.mldn.controller.EchoAction"**>  
 *<!--定义跳转路径-->* <**result name="echo.jsp"**>echo.jsp</**result**>  
 </**action**>  
 </**package**>  
</**struts**>

* 1. 创建页面：

<%@ **page language**="**java**" **import**="**java.util.\***" **pageEncoding**="**UTF-8**"%>  
<%@ **taglib prefix**="**s**" **uri**="**/struts-tags**"%>  
**<%** String path = request.getContextPath();  
 String basePath = request.getScheme() + **"://"** + request.getServerName() + **":"** + request.getServerPort()  
 + path + **"/"**;  
**%>**<**html**>  
<**head**>  
 <**title**>第一个Struts2.x 程序</**title**>  
</**head**>  
<**body**>  
 <**h1**> <**s:property value="msg"**></**s:property**></**h1**>  
 <**s:form action="EchoAction" method="POST"**>  
 <**s:textfield key="msg" label="请输入信息"**/>  
 <**s:submit value="发送"**></**s:submit**>  
 </**s:form**>  
</**body**>  
</**html**>

* 1. 异常处理：

严重: Exception starting filter struts2

java.lang.ClassNotFoundException: org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter

at org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase.java:1907)

at org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase.java:1750)

at org.apache.catalina.core.DefaultInstanceManager.loadClass(DefaultInstanceManager.java:534)

at org.apache.catalina.core.DefaultInstanceManager.loadClassMaybePrivileged(DefaultInstanceManager.java:516)

at org.apache.catalina.core.DefaultInstanceManager.newInstance(DefaultInstanceManager.java:148)

at org.apache.catalina.core.ApplicationFilterConfig.getFilter(ApplicationFilterConfig.java:264)

at org.apache.catalina.core.ApplicationFilterConfig.<init>(ApplicationFilterConfig.java:108)

at org.apache.catalina.core.StandardContext.filterStart(StandardContext.java:4949)

at org.apache.catalina.core.StandardContext.startInternal(StandardContext.java:5651)

at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:145)

at org.apache.catalina.core.ContainerBase.addChildInternal(ContainerBase.java:1015)

at org.apache.catalina.core.ContainerBase.addChild(ContainerBase.java:991)

at org.apache.catalina.core.StandardHost.addChild(StandardHost.java:652)

at org.apache.catalina.startup.HostConfig.manageApp(HostConfig.java:1899)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

at java.lang.reflect.Method.invoke(Method.java:498)

at org.apache.tomcat.util.modeler.BaseModelMBean.invoke(BaseModelMBean.java:301)

at com.sun.jmx.interceptor.DefaultMBeanServerInterceptor.invoke(DefaultMBeanServerInterceptor.java:819)

at com.sun.jmx.mbeanserver.JmxMBeanServer.invoke(JmxMBeanServer.java:801)

at org.apache.catalina.mbeans.MBeanFactory.createStandardContext(MBeanFactory.java:618)

at org.apache.catalina.mbeans.MBeanFactory.createStandardContext(MBeanFactory.java:565)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

at java.lang.reflect.Method.invoke(Method.java:498)

at org.apache.tomcat.util.modeler.BaseModelMBean.invoke(BaseModelMBean.java:301)

at com.sun.jmx.interceptor.DefaultMBeanServerInterceptor.invoke(DefaultMBeanServerInterceptor.java:819)

at com.sun.jmx.mbeanserver.JmxMBeanServer.invoke(JmxMBeanServer.java:801)

at javax.management.remote.rmi.RMIConnectionImpl.doOperation(RMIConnectionImpl.java:1468)

at javax.management.remote.rmi.RMIConnectionImpl.access$300(RMIConnectionImpl.java:76)

at javax.management.remote.rmi.RMIConnectionImpl$PrivilegedOperation.run(RMIConnectionImpl.java:1309)

at javax.management.remote.rmi.RMIConnectionImpl.doPrivilegedOperation(RMIConnectionImpl.java:1401)

at javax.management.remote.rmi.RMIConnectionImpl.invoke(RMIConnectionImpl.java:829)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

at java.lang.reflect.Method.invoke(Method.java:498)

at sun.rmi.server.UnicastServerRef.dispatch(UnicastServerRef.java:357)

at sun.rmi.transport.Transport$1.run(Transport.java:200)

at sun.rmi.transport.Transport$1.run(Transport.java:197)

at java.security.AccessController.doPrivileged(Native Method)

at sun.rmi.transport.Transport.serviceCall(Transport.java:196)

at sun.rmi.transport.tcp.TCPTransport.handleMessages(TCPTransport.java:568)

at sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run0(TCPTransport.java:826)

at sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.lambda$run$0(TCPTransport.java:683)

at java.security.AccessController.doPrivileged(Native Method)

at sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run(TCPTransport.java:682)

at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)

at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)

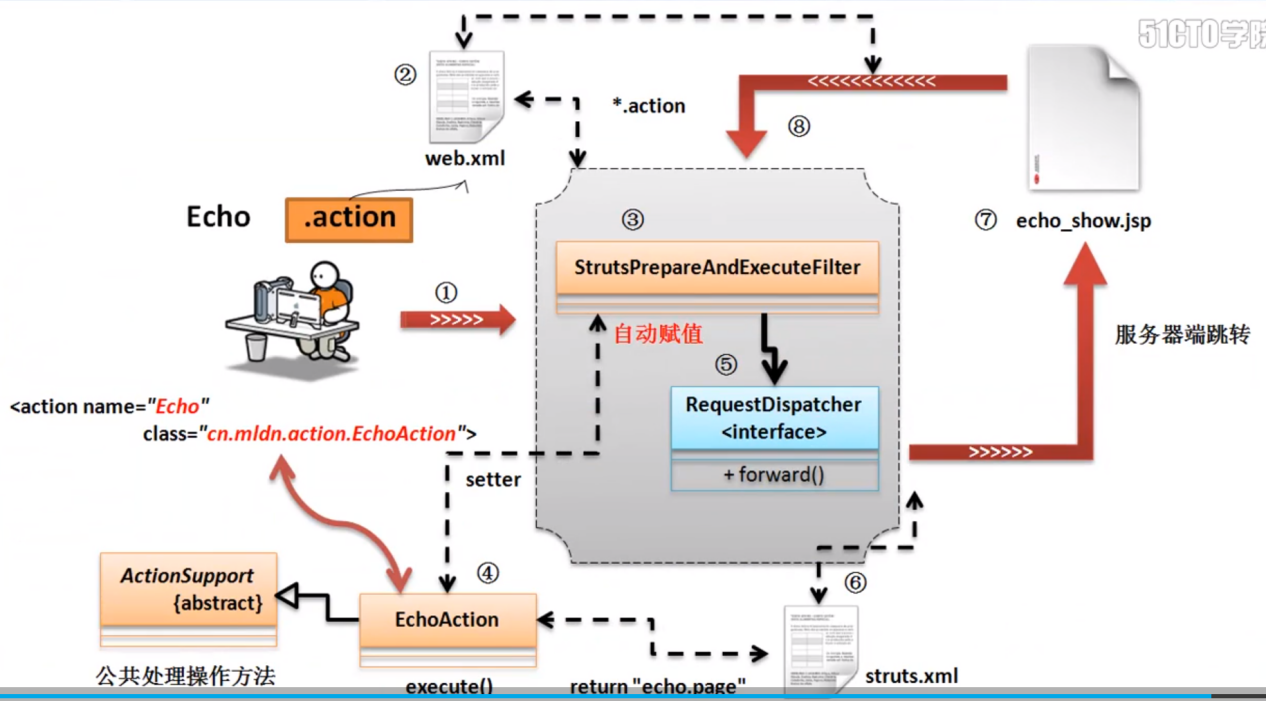
at java.lang.Thread.run(Thread.java:748)

大部分查出来的都可能是jar包问题，如果你确认jar包没问题的话，看一下你struts2 core包的版本

2.5以上的版本，请把org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter修改为org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter。

1. Strust2.x 工作原理

一旦使用了Strust2 标签，就必须由过滤器的路径跳转到jsp页面。



1. 用户首先发请求到Strust的过滤器上，输入路径“\*.action”,匹配到到web.xm配置的“<url-pattern>\*.action</url-pattern>”
2. 而后请求提交到过滤器中后，实际上并不是由过滤器来负责处理，而是过滤器会根据Strust.xml配置文件中的Action的名字，“<action name="XxxAction" class="cn.mldn.controller.XxxAction">”，根据相应的类型通过反射取得与之对应的Action处理类的名称。
3. 而后调用ActionSupport(抽象类)类中的execute()方法，但此方法是被子类所覆写，所以本质上是调用XxxAction类中的execute()方法。
4. 当XxxAction中的execute()的方法执行完毕后将返回一个字符串“return "echo.page";”
5. 根据<package>元素中namespace属性结合定位出完整路径，而后利用RequestDispatcher接口提供的forward跳转到指定的页面。
6. 跳转类型配置

默认情况下Strust2使用的是forward的服务器端跳转，，若果想要使用客户端跳转，则需要修改跳转类型，Strust2里面一共有三种类型：

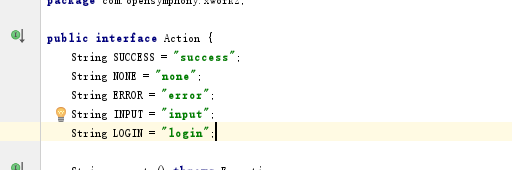
1. type="dispatcher",为默认类型表示服务器端跳转，此时可以通过标签使用XxxAction类中的属性，<result name="echo.page" type="dispatcher">/echo.jsp</result>。
2. type=”redirect”,表示客户端跳转，此时不能通过标签使用类中的属性。
3. type=”redirect-action”、”redirectAction”表示跳转到另外一个action。此时依然是服务器端跳转。
4. 过滤器配置
5. 新过滤器：
   1. org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter：在页面执行之前和页面执行之后都要执行此过滤器。
   2. org.apache.struts2.dispatcher.filter.StrutsPrepareFilter:在页面执行（页面还能生成）之前执行过滤器。
   3. org.apache.struts2.dispatcher.filter.StrutsExecuteFilter:在页面执行之后（页面已生成）执行过滤器。

以上三种过滤器均实现了javax.servlet.filter接口;

1. 老过滤器：

org.apache.struts2.dispatcher.FilterDispatcher,此过滤器是来自于webWork框架的。目前新版本的strust2已不使用。

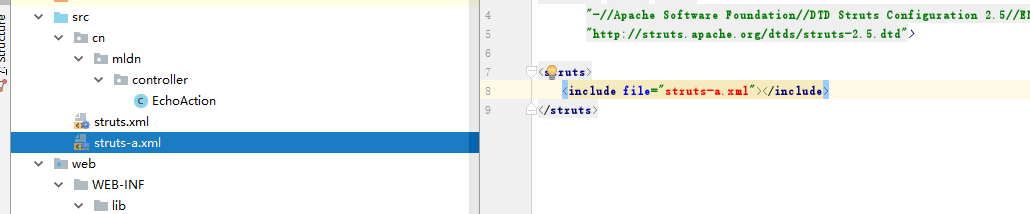
1. 内置跳转名称



1. JSP内置对象

要取得JSP内置对象，需要通过ServletActionContext，Strust2隐藏了JSP内置对象。

1. 多人开发



1. 乱码解决

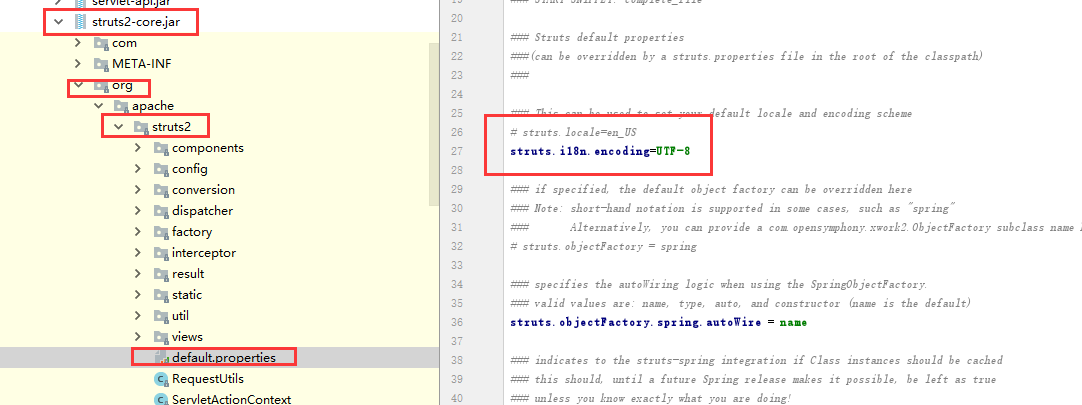
在类路径下创建文件struts.properties：

struts.i18n.encoding = UTF-8

Struts.local=zh\_CN

若果没有配置，则取默认资源文件的编码：

struts2-core.jar下



1. 配置资源文件
2. 创建以properties为后缀的文件，并修改struts.properties文件。

Struts.custom.i18n.resources = 文件名称1,,文件名称2...

1. 使用getText()方法对取对应key的内容，此方法不区分文件，按key值读取。
2. 结合VO类输入

Struts2.x 实现了参数与vo类的转换操作，包括多级对象赋值，但需要时用new实例化对象。

1. Struts2核心标签库

3.1、标签与属性

在Struts2中标签默认只能输出Action类中的属性，是通过OGNL语法完成的。若果想使用OGNL表达式获取request、page、session、application中的属性内容，则需使用“#request.属性”的格式。

OGNL和EL表达式类似，之所以在Struts2中重复提到，是因为Struts2比JSP设计的早，并且EL表达式是在JSP2.0才开始有的。可以发现Struts2标签这么复杂，还不如直接使用EL表达式。

<h1>Action属性"msg"<s:property value="msg"/></h1>

<h1>Action属性"dept.deptno"<s:property value="dept.deptNo"/></h1>

<h1>Action属性"dept.dname"<s:property value="dept.dname"/></h1>

<h1>request属性"dept.dname"<s:property value="#request.mydept.dname"/></h1>

四、Struts2.x多业务处理

若想要Action处理用户的多个请求，则不能覆写execute()方法。这样每次请求都将实例化Action，即多例。

4.1、定义Action类：

**public class** NewsAction **extends** ActionSupport {  
 **private** News **news** = **new** News(); *// 目的是为了接收数据* **public** News getNews() {  
 *//反射设置属性是通过getter实现的* **return news**;  
 }  
  
 **public** String insert() { *// 增加操作但是不跳转* System.***out***.println(**"\*\*\*\*\*\* 【增加新闻】"** + **this**.**news**);  
 **return null**;  
 }  
  
 **public void** update() { *// 修改操作不跳转* System.***out***.println(**"\*\*\*\*\*\* 【修改新闻】"** + **this**.**news**);  
 }  
  
 **public void** list() {  
 System.***out***.println(**"\*\*\*\*\*\* 【新闻列表】"** + **this**.**news**);  
 }  
  
 **public void** delete() {  
 System.***out***.println(**"\*\*\*\*\*\* 【删除新闻】"** + **this**.**news**);  
 }  
}

4.2、配置action

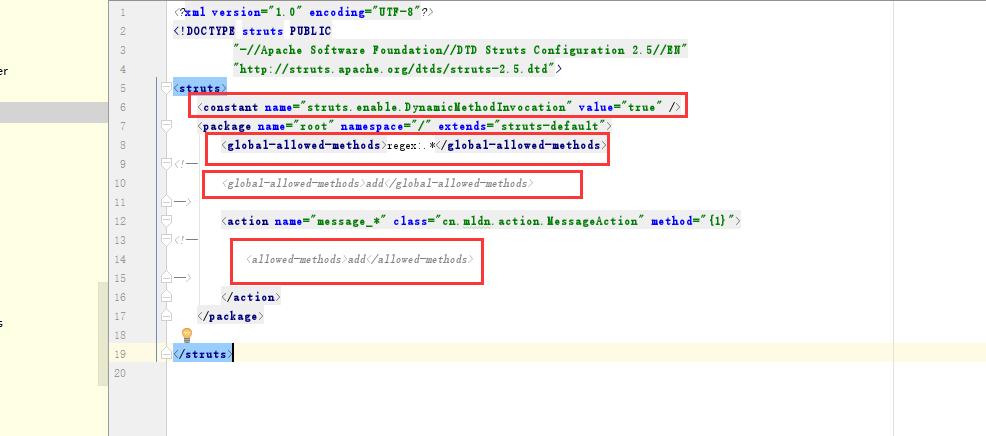
4.2.1 使用配置文件的方式配置

<**struts**>  
 *<!--所有的Struts2x 都要求设置命名空间-->* <**package name="root" namespace="/" extends="struts-default"**>  
 *<!--配置action程序的名称的及对应的路径-->* <**action name="EchoAction" class="cn.mldn.controller.EchoAction"**>  
 *<!--定义跳转路径-->* <**result name="echo.page"**>/echo.jsp</**result**>  
 </**action**>  
 <**action name="NewsAction\_\*" class="cn.mldn.controller.NewsAction" method="{1}"**></**action**>  
 </**package**>  
</**struts**>

访问list()方法：

<http://localhost:8080/NewsAction_list.action>

2.5版本及以后要添加允许访问的方法：



在Struts2.3.x或更高的某些版本时出现There is no Action mapped for namespace [/user] and action name [user!add] associated with context path错误，原因是（1）DMI可能导致安全问题（2）DMI与通配符方法功能有重叠，因此该版本Struts2默认关闭DMI（动态方法调用），需要在struts.xml中加一句<constant name="struts.enable.DynamicMethodInvocation" value="true" /> 来打开动态方法调用。动态方法调用官方推荐的做法是，使用通配符的形式。不要使用actionName!methodName的方式。

4.2.1 零配置action(推荐)

<**action name="NewsAction" class="cn.mldn.controller.NewsAction"**></**action**>

访问list()方法：

[http://localhost:8080/NewsAction!](http://localhost:8080/NewsAction!list.action)**[list](http://localhost:8080/NewsAction!list.action)**[.action](http://localhost:8080/NewsAction!list.action)

使用注解时出现以下错误：

There is no Action mapped for namespace [/] and action name [member] associated with context path []. - [unknown location]。

我日了，这个问题，各种排查，结果是：

以下是Struts2的原文文档：

First the Convention plugin finds packages named struts, struts2, action or actions. Any packages that match those names are considered the root packages for the Convention plugin. Next, the plugin looks at all of the classes in those packages as well as sub-packages and determines if the classes implementcom.opensymphony.xwork2.Action or if their name ends with Action (i.e. FooAction).

我的包名称是xxx.controller，然后扫描的包配置了xxx.controller。真是醉了，花了快4小时的时间去找问题。

结果把controller改为action就好了。

1. 数据验证

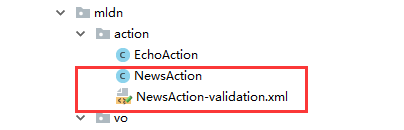
Struts2 要进行数据验证，必须重写validate()方法，此方法在所有业务方法前执行。

@Override  
**public void** validate() {  
 System.***out***.println(**"数据验证"**);  
 **if**(**this**.**news**.getNid() == **null**){  
 **this**.addFieldError(**"news.nid"**,**super**.getText(**"error.data.null"**,**new** String[]{**"新闻编号"**}));  
 }  
 **if**(**this**.**news**.getTitle() == **null**){  
 **this**.addFieldError(**"news.title"**,**super**.getText(**"error.data.null"**,**new** String[]{**"新闻标题"**}));  
 }  
 **if**(**this**.**news**.getContent() == **null**){  
 **this**.addFieldError(**"news.content"**,**super**.getText(**"error.data.null"**,**new** String[]{**"新闻内容"**}));  
 }  
 **if**(**this**.**news**.getPubdate() == **null**){  
 **this**.addFieldError(**"news.pubdate"**,**super**.getText(**"error.data.null"**,**new** String[]{**"新闻时间"**}));  
 }  
}

1. 清楚addFiledError()的方法的使用，此方法上前面跟上的hi参数名称。
2. 使用addFiledError()方法相当于在Map集合里面追加了新的错误提示信息，若果有内容则会跳转到input配置的错误页面。

错误信息Map集合返回值类型为 Map<String, List<String>>

可以使用Struts2提供的验证框架，

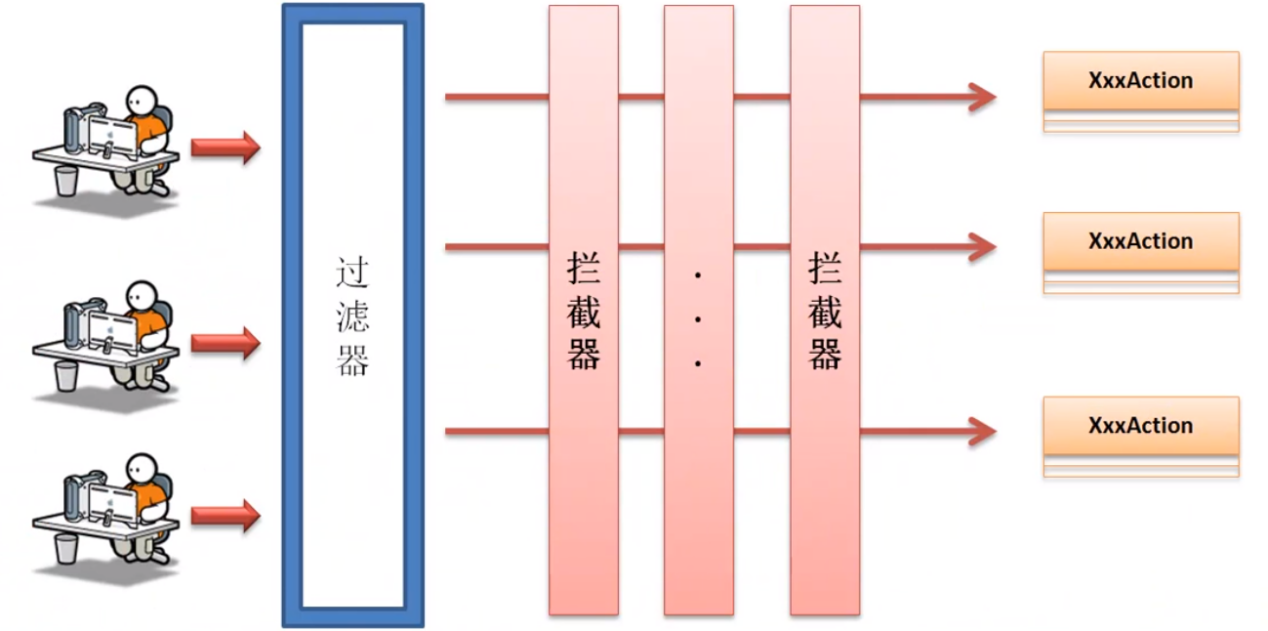


*<?***xml version="1.0" encoding="UTF-8"** *?>***<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd"*>***<**validators**>  
 <**field name="news.nid"**>  
 <**field-validator type="int"**>  
 <**param name="min"**>10</**param**>  
 <**param name="max"**>30</**param**>  
 <**message**>新闻编号成都为10-30字</**message**>  
 </**field-validator**>  
 <**field-validator type="requiredstring"**>  
 <**message**>新闻编号不允许为空！</**message**>  
 </**field-validator**>  
 </**field**>  
</**validators**>

不论是validate()和验证框架都是在参数转换为VO后在验证。

1. 拦截器

5.1、所谓拦截器就是AOP(面向切面编程)的一种实现，就是代理设计模式的实现。在请求真正的到达某一个action之前做一些处理操作，而这就属于拦截器的功能。



只有在拦截器存在的前提下，才能对数据转换及验证进行具体的操作。由于Struts2 使用的是过滤器处理接收的所有请求，所以session登录检测不可能只在过滤器里面处理了，需要进行两次处理：

1. 在过滤器中处理页面登录检测
2. 使用拦截器处理action里的登录检测。

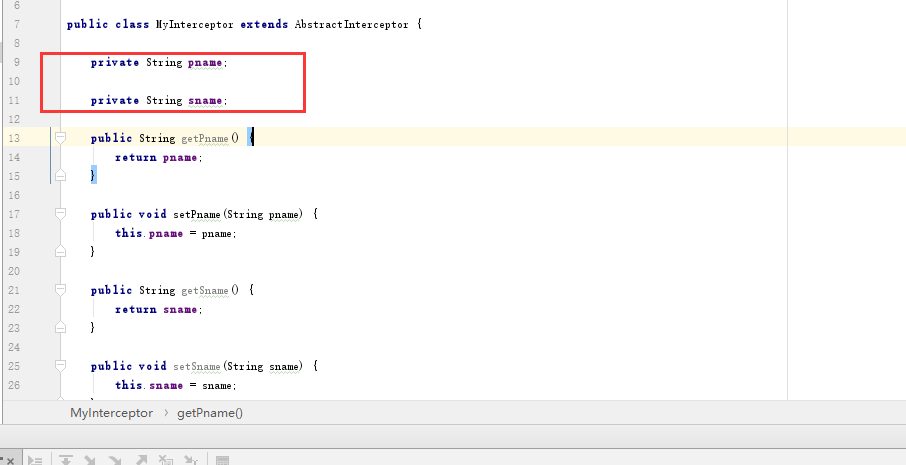
5.2、自动赋值

一旦使用了拦截器后悔发现，不能为属性自动赋值了，这是因为没有定义使用设置内容的拦截器。需要使用defaultStack才能恢复自动设置属性。

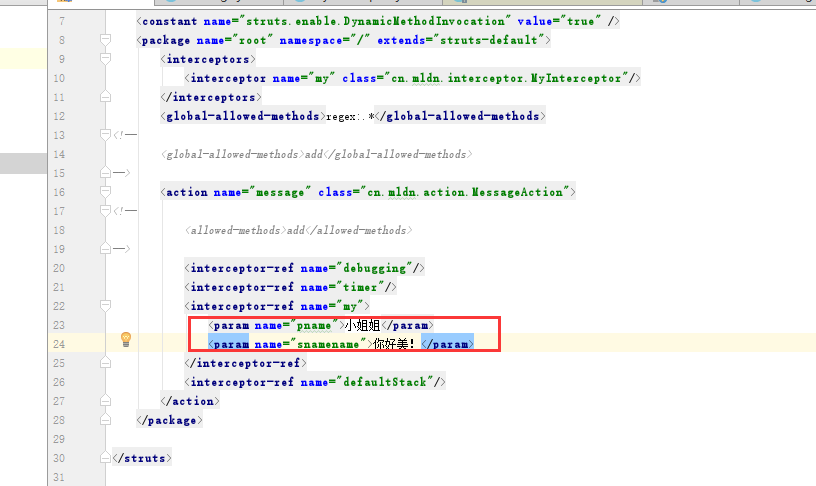


5.2、属性设置

5.2.1、如果拦截器中需要接受一些属性的话，那么可以通过配置参数完成。在拦截器定义属性



5.2.2、在配置文件中传递参数：





5.3、登录拦截

由于Struts2.x 过滤器用于处理用户请求，对于登录拦截，只能对页面进行处理，所以，必须是用拦截器进行action的登录拦截。

5.3.1 获取Session

Struts2.x获取Session的两种方式：

1. ServletActionContext.getRequest().getSession();
2. ActionContext actionContext = actionInvocation.getInvocationContext();String username=actionContext.getSession().get("username") == null ?"":(String)actionContext.getSession().get("username");

5.4 拦截器栈

所谓拦截器栈是指为一组拦截器配置统一的名字，而后在需要的位置上使用一个拦截器栈的名字即可操作。

配置拦截器栈：

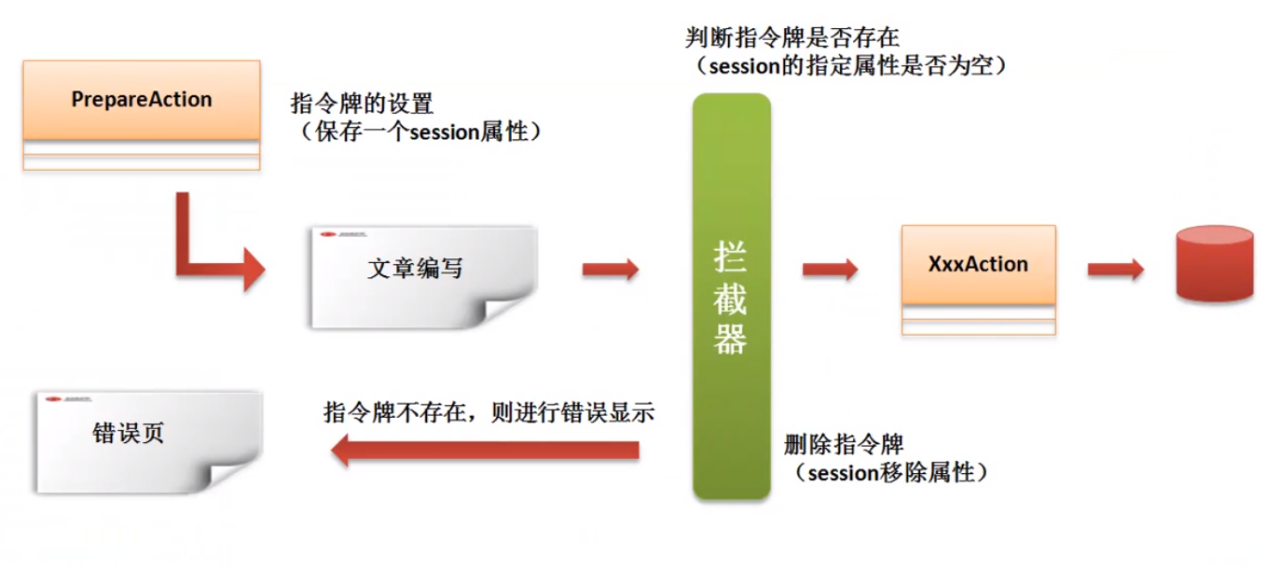
<**interceptors**>  
 <**interceptor name="my" class="cn.mldn.interceptor.MyInterceptor"**/>  
 <**interceptor name="login" class="cn.mldn.interceptor.LoginInterceptor"** />  
 <**interceptor-stack name="xiaojiejieStack"**>  
 <**interceptor-ref name="debugging"**/>  
 <**interceptor-ref name="timer"**/>  
 <**interceptor-ref name="login"**/>  
 <**interceptor-ref name="my"**>  
 <**param name="pname"**>小姐姐</**param**>  
 <**param name="snamename"**>你好美！</**param**>  
 </**interceptor-ref**>  
 <**interceptor-ref name="defaultStack"**/>  
 </**interceptor-stack**>

使用拦截器栈：

<**action name="message" class="cn.mldn.action.MessageAction"**>  
*<!--  
 <allowed-methods>add</allowed-methods>  
-->* <**interceptor-ref name="xiaojiejieStack"**/>  
 </**action**>

1. 指令牌的验证作用

指令牌可以用于防止重复提交：页面进入必须使用Action，此时在PrepareAction中设置指令牌（保存一个session属性），Struts2 会自动在提交到XxxAction前的过程中添加一个拦截器，判断指令牌是否存在（session的指定属性是否为空）,操作完成后删除指令牌（移除session属性）,假如指令牌不存在，则跳转到错误页面(提交页也是可以的)。



范例：

6.1 创建两个Action：

**public class** PrepareAction **extends** ActionSupport {  
 **public** String addPre() {  
  
 **return "add.page"**;  
 }  
}

**public class** MessageAction **extends** ActionSupport {  
  
 **private** String **msg**;  
  
 **public void** setMsg(String msg) {  
 **this**.**msg** = msg;  
 }  
  
 **public void** add() {  
 System.***out***.println(**"\*\*\*\*正常接收数据"** + **this**.**msg**);  
 }  
}

配置struts.xml文件：

<**struts**>  
 <**constant name="struts.enable.DynamicMethodInvocation" value="true"**/>  
 <**package name="root" namespace="/" extends="struts-default"**>  
 <**global-allowed-methods**>regex:.\*</**global-allowed-methods**>  
 <**action name="message" class="cn.mldn.action.MessageAction"**>  
 <**result name="invalid.token"**>/errors.jsp</**result**>  
 <**interceptor-ref name="token"**/>  
 <**interceptor-ref name="defaultStack"**/>  
 </**action**>  
 <**action name="pre" class="cn.mldn.action.PrepareAction"**>  
 <**result name="add.page"**>/msg\_add.jsp</**result**>  
 </**action**>  
 </**package**>  
</**struts**>

创建相关页面：

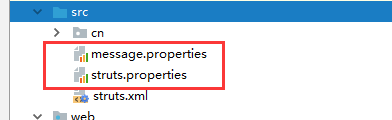
msg\_add.jsp：

<%@ **page language**="**java**" **import**="**java.util.\***" **pageEncoding**="**UTF-8**" %>  
<%@ **taglib prefix**="**s**" **uri**="**/struts-tags**" %>  
**<%** String path = request.getContextPath();  
 String basePath = request.getScheme() + **"://"** + request.getServerName() + **":"** + request.getServerPort() + path + **"/"**;  
 String addUrl = basePath + **"message!add.action"**;  
**%>**<**html**>  
<**head**>  
 <**title**>令牌指令处理</**title**>  
 <**base href="<%=**basePath**%>"**>  
</**head**>  
<**body**>  
<**form action="<%=**addUrl**%>" method="post"**>  
 请输入消息：<**input type="text" name="msg" id="msg"**/>  
  **<s:token></s:token>**  
 <**input type="submit" value="发送"**/>  
</**form**>  
</**body**>  
</**html**>

Errors.jsp:

<**body**>  
<**h1**><**s:actionerror**></**s:actionerror**></**h1**>  
</**body**>  
</**html**>

实现重复提交信息中文化：



在message资源文件中配置提示信息：

**struts.messages.invalid.token**=**=你的的表单已经提交了，请勿重复提交！**

1. 文件上传

在Struts2.x以及后面的springMVC都是使用组件实现。

范例：

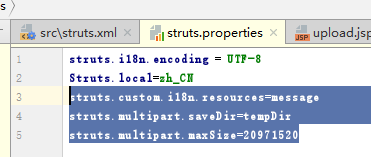
7.1、创建上传工具类：

**public class** UploadFileUtil {  
 *//上传的文件内容* **private** File **uploadFile**;  
  
 *//上传的文件类型* **private** String **contentType**;  
  
 *//生成文件名称* **private** String **fileName**;  
  
 **public** UploadFileUtil(File uploadFile, String contentType) {  
 **this**.**uploadFile** = uploadFile;  
 **this**.**contentType** = contentType;  
 }  
  
 */\*\*  
 \* 取得文件的名称，通过UUID命名  
 \*  
 \** ***@return*** *\*/* **public** String getFileName() {  
 **if** (**null** == **this**.**fileName** || **""**.equals(**this**.**fileName**)) {  
 String fileExt = **null**; *// 保存文件的扩展类型* **if** (**"image/bmp"**.equalsIgnoreCase(**this**.**contentType**)) {  
 fileExt = **"bmp"**;  
 } **else if** (**"image/gif"**.equalsIgnoreCase(**this**.**contentType**)) {  
 fileExt = **"gif"**;  
 } **else if** (**"image/jpeg"**.equalsIgnoreCase(**this**.**contentType**)) {  
 fileExt = **"jpg"**;  
 } **else if** (**"image/png"**.equalsIgnoreCase(**this**.**contentType**)) {  
 fileExt = **"png"**;  
 }  
 **this**.**fileName** = UUID.*randomUUID*() + **"."** + fileExt;  
 }  
  
 **return this**.**fileName**;  
 }  
  
 */\*\*  
 \* 删除文件  
 \*  
 \** ***@param filePath*** *\** ***@return*** *\*/* **public boolean** deleteFile(String filePath) {  
 File file = **new** File(filePath);  
 **if** (file.exists()) {  
 **return** file.delete();  
 }  
 **return false**;  
 }  
  
 */\*\*  
 \* 保存文件  
 \*  
 \** ***@param outFilePath*** *\** ***@return*** *\*/* **public boolean** saveFile(String outFilePath) **throws** IOException {  
 File file = **new** File(outFilePath);  
 OutputStream out = **null**;  
 InputStream ins = **null**;  
  
 **try** {  
 *//文件路径不存在，则创建* **if** (!file.getParentFile().exists()) {  
 file.getParentFile().mkdirs();  
 }  
 out = **new** FileOutputStream(file);  
 ins = **new** FileInputStream(**this**.**uploadFile**);  
 *//每次读取内容* **byte**[] data = **new byte**[2048];  
 **int** len = 0;  
 **while** ((len = ins.read(data)) != -1) {  
 out.write(data, 0, len);  
 }  
 **return true**;  
 } **catch** (Exception e) {  
 **throw** e;  
 } **finally** {  
 **if** (out != **null**) {  
 out.close();  
 }  
 **if** (ins != **null**) {  
 ins.close();  
 }  
 }  
 }  
}

7.2、创建Action类处理上传

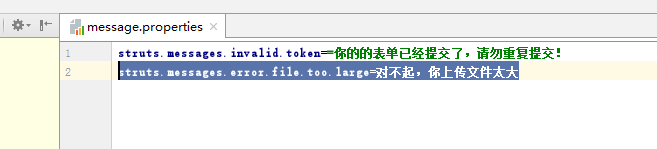
**public class** UploadAction **extends** ActionSupport {  
 **private** String **name**;  
  
 **private** File **photo**;  
  
 **private** String **photoFileName** ;  
  
 **private** String **photoContentType** ;  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
 **public void** setPhoto(File photo) {  
 **this**.**photo** = photo;  
 }  
  
 **public void** setPhotoFileName(String photoFileName) {  
 **this**.**photoFileName** = photoFileName;  
 }  
  
 **public void** setPhotoContentType(String photoContentType) {  
 **this**.**photoContentType** = photoContentType;  
 }  
  
 **public void** insert() {  
 UploadFileUtil ufu = **new** UploadFileUtil(**this**.**photo**,**this**.**photoContentType**) ;  
 String fileName = ufu.getFileName() ;  
 System.***out***.println(**"新的文件名称："** + fileName);  
 String filePath = ServletActionContext.*getServletContext*().getRealPath(**"/upload/"**) + fileName ;  
 **try** {  
 System.***out***.println(ufu.saveFile(filePath));  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

7.3、配置上传限制：



<**action name="upload" class="cn.mldn.action.UploadAction"**>  
 <**result name="input"**>/errors.jsp</**result**>  
 <**interceptor-ref name="fileUpload"**>  
 <**param name="allowedTypes"**>  
 image/bmp,image/jpg,image/jpeg,image/gif,image/png  
 </**param**>  
 </**interceptor-ref**>  
 <**interceptor-ref name="defaultStack"**/>  
</**action**>

修改错误提示：



1. 自定义转换器

8.1 创建Action类

**public class** LocaleAction **extends** ActionSupport {  
 **private** Locale **loc**;  
  
 **public void** setLoc(Locale loc) {  
 **this**.**loc** = loc;  
 }  
  
 **public** Locale getLoc() {  
 **return loc**;  
 }  
  
 **public void** insert() {  
 *//String str = LocalizedTextUtil.findDefaultText("welcome", this.loc);  
 // System.out.println(str);* System.***out***.println(**"\*\*\*\*\*"** + **this**.**loc**);  
 }

8.2 定义转换器

**import** com.opensymphony.xwork2.conversion.impl.DefaultTypeConverter;  
  
**import** java.util.Locale;  
**import** java.util.Map;  
  
**public class** StringToLocaleConverter **extends** DefaultTypeConverter {  
 @Override  
 **public** Object convertValue(Map<String, Object> context, Object value, Class toType) {  
 String result = ((String[])value)[0];  
 **if**(Locale.**class**.equals(toType)){  
 String[] str =result.split(**"\_"**);  
 Locale loc = **new** Locale(str[0],str[1]);  
 **return** loc;  
 }  
 **return null**;  
 }  
}

8.3 创建页面

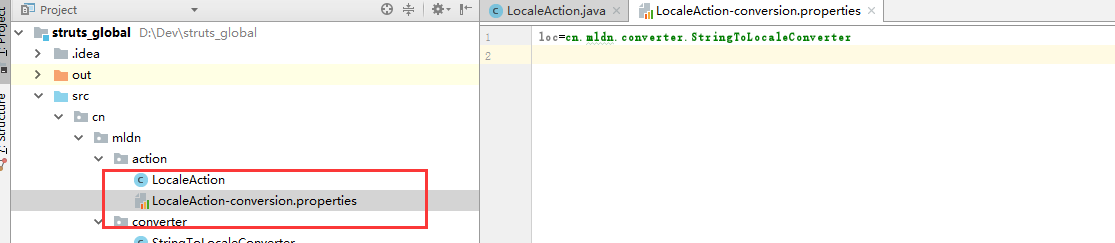
<%@ **page language**="**java**" **import**="**java.util.\***" **pageEncoding**="**UTF-8**"%>  
**<%** String path = request.getContextPath();  
 String basePath = request.getScheme() + **"://"** + request.getServerName() + **":"** + request.getServerPort()  
 + path + **"/"**;  
**%>**<!DOCTYPE **HTML** PUBLIC **"-//W3C//DTD HTML 4.01 Transitional//EN"**>  
<**html**>  
<**head**>  
<**base href="<%=**basePath**%>"**>  
<**title**>My JSP 'index.jsp' starting page</**title**>  
</**head**>  
  
<**body**>  
 <**div**>  
 <**form action="LocaleAction!insert.action" method="post"**>  
 选择信息语言： <**select name="loc"**>  
 <**option value="zh\_CN"**>中文显示</**option**>  
 <**option value="en\_US"**>英文显示</**option**>  
 </**select**>   
 <**input type="submit" value="提交"**>  
 </**form**>  
 </**div**>  
</**body**>  
</**html**>

完成以上步奏，要想让转换器起作用，还必须配置：

配置形式一：定义全局配置，在类路径下新建资源文件“xwork-conversion.properties”，配置内容:java.util.Locale=cn.mldn.converter.StringToLocaleConverter

此种配置会造成所有Action类的字符串转换皆被影响。所以不推荐使用。

配置形式二：针对某个Action类配置，可以在类所在目录下，新建资源文件

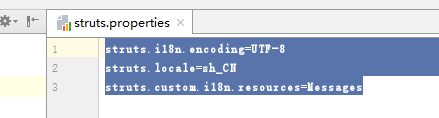


8.4 国际化支持

**import** java.util.Locale;  
  
**import** com.opensymphony.xwork2.ActionSupport;  
**import** com.opensymphony.xwork2.util.LocalizedTextUtil;  
  
**public class** LocaleAction **extends** ActionSupport {  
 **private** Locale **loc**;  
  
 **public void** setLoc(Locale loc) {  
 **this**.**loc** = loc;  
 }  
  
 **public void** insert() {

//此方法在2.5版本找不到了  
 **String str = LocalizedTextUtil.*findDefaultText*("welcome", this.loc);**  
 System.***out***.println(str);  
 }  
}

配置资源文件：



而后分别针对不同的语言环境，配置资源文件：

Messages\_en\_US.properties：welcome=WELCOME !

Messages\_zh\_CN.properties：welcome=欢迎光临 !



8.5 扩展转换器

8.5.1 Set 转换器(Struts2 无法确定Set类型，无法自动转换)

import com.opensymphony.xwork2.conversion.impl.DefaultTypeConverter;

public class IntegerToSetConverter extends DefaultTypeConverter {

@Override

public Object convertValue(Map<String, Object> context, Object value,

Class toType) {

if (Set.class.equals(toType)) { // 目标类型是否是Set集合

Set<Integer> set = new **HashSet**<Integer>() ;

String val = ((String[]) value)[0];

String result [] = val.split("\\|") ;

for (int x = 0 ; x < result.length ; x ++) {

set.add(Integer.parseInt(result[x])) ;

}

return set ;

}

return null ;

}

}

1. 注解（struts2-convention-plugin-2.3.34.jar）

Struts2.x默认情况下打开了注解支持，若果有特殊需要，也可进行单独打开配置：

<**filter**>  
 <**filter-name**>struts2</**filter-name**>  
 <**filter-class**>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</**filter-class**>  
 <**init-param**>  
 <**param-name**>actionPages</**param-name**>  
 <**param-value**>cn.xxx.xxx</**param-value**>  
 </**init-param**>  
</**filter**>

以上配置默认支持annotation扫描（好像不加也是可以的）。

9.1 核心配置

在整个项目中最为重要的就是action，但是任何项目里面都可能有一些特殊需求的配置，例如全局跳转、拦截器定义。

9.1.1 全局跳转

*<?***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.enable.DynamicMethodInvocation" value="true"**/>  
 <**package name="root" namespace="/" extends="struts-default"**>

<!--路径跳转-->  
 **<global-results>  
 <result name="forward.page">/forward.jsp</result>  
 <result name="error.page">/error.jsp</result>  
 </global-results>**  
 <**action name="LocaleAction" class="cn.mldn.action.LocaleAction"** >  
 <**result name="input"**>/index.jsp</**result**>  
 <**interceptor-ref name="defaultStack"**/>  
 </**action**>  
 </**package**>  
</**struts**>

设置父路径：

@ParentPackage(**"back"**)

@Action(value=**"message"**)

<**package name="root" namespace="/" extends="struts-default"**>  
 <**global-results**>  
 <**result name="forward.page"**>/forward.jsp</**result**>  
 <**result name="error"**>/error.jsp</**result**>  
 </**global-results**>  
</**package**>  
<**package name="back" extends="root" namespace="/pages/back"**></**package**>  
<**package name="front" extends="root" namespace="/pages/front"**></**package**>

设置子目录：

@Namespace(**"/pages/back/msg"**)

<http://localhost:8080/pages/back/msg/message!list.action>

设置返回路径：

@ParentPackage(**"back"**)  
@Namespace(**"/pages/back/msg"**)  
@Action(value = **"message"**)  
@Results(value = {  
 @Result(name = **"show.page"**, location = **"/pages/back/msg/show.jsp"**)  
})

<http://localhost:8080/pages/back/msg/message!list.action>

设置方法访问路径：

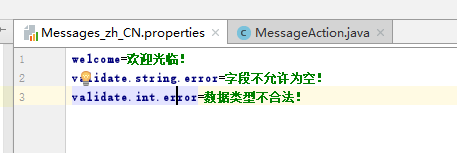
@ParentPackage(**"back"**)  
@Namespace(**"/pages/back/msg"**)  
@Action(value = **"message"**)  
@Results(value = {  
 @Result(name = **"show.page"**, location = **"/pages/back/msg/show.jsp"**)  
})  
**public class** MessageAction **extends** ActionSupport {  
 @Actions(value={@Action(**"list1"**),@Action(**"list2"**)})  
 **public** String list() {  
 System.***out***.println(**"\*\*\*\*\*\*\*\*\*\*\*\*消息列表\*\*\*\*\*\*\*\*\*\*\*"**);  
 **return "show.page"**;  
 }  
}

拦截器配置：

@InterceptorRefs(  
 value={  
 @InterceptorRef(**"myStack"**)  
 }  
)

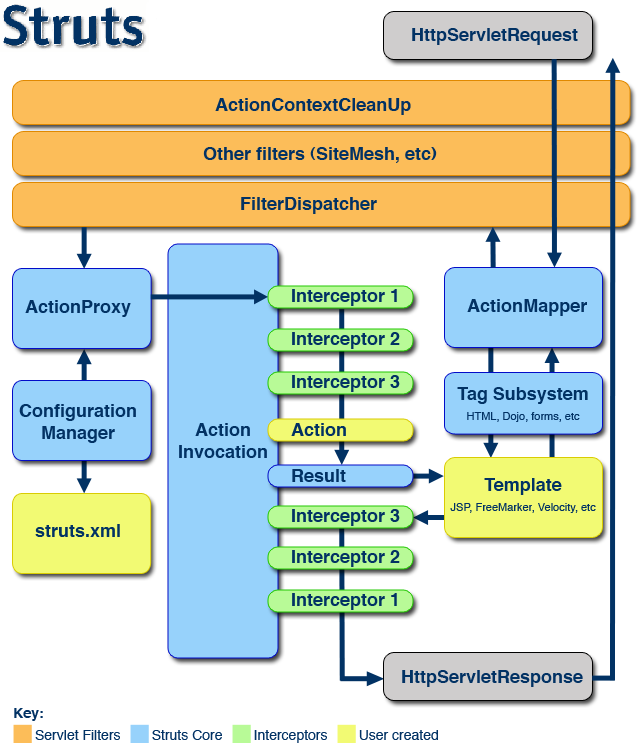
注解数据验证：

@ParentPackage(**"back"**)  
@Namespace(**"/pages/back/msg"**)  
@Action(value = **"message"**)  
@Results(value = {  
 @Result(name = **"show.page"**, location = **"/pages/back/msg/show.jsp"**)  
})  
@InterceptorRefs(  
 value = {  
 @InterceptorRef(**"myStack"**)  
 }  
)  
@Validations(  
 requiredStrings = {  
 @RequiredStringValidator(fieldName = **"member.mid"**, key = **"validate.string.error"**),  
 @RequiredStringValidator(fieldName = **"member.name"**, key = **"姓名不能为空"**),  
 @RequiredStringValidator(fieldName = **"member.email"**, key = **"邮箱不能为空"**),  
 },  
 regexFields = {  
 *//正则验证 STruts2.3以前caseSensitiveExpression替换为expression* @RegexFieldValidator(fieldName = **"member.mid"**, caseSensitiveExpression = **"\\d+"**, key = **"validate.int.error"**),  
 @RegexFieldValidator(fieldName = **"member.email"**, caseSensitiveExpression = **"\\w+@\\w+\\.\\w+"**, message = **"邮箱格式不正确"**)  
 }  
)  
**public class** MessageAction **extends** ActionSupport {  
  
 *//Struts2.x 对象转换之前，必需先实例化* **private** Member **member** = **new** Member();  
  
 **public** Member getMember() {  
 **return member**;  
 }  
  
 **public** String show() {  
 System.***out***.println(**this**.**member**);  
 System.***out***.println(**"消息显示!"**);  
 **return "show.page"**;  
 }  
  
 @Actions(value = {@Action(**"li"**), @Action(**"lis"**)})  
 **public** String list() {  
 System.***out***.println(**"\*\*\*\*\*\*\*\*\*\*\*\*消息列表\*\*\*\*\*\*\*\*\*\*\*"**);  
 **return "show.page"**;  
 }  
}



**Struts2工作原理和执行流程图**

在struts2的应用中，从用户请求到服务器返回相应响应给用户端的过程中，包含了许多组件如：Controller、ActionProxy、ActionMapping、Configuration Manager、ActionInvocation、Inerceptor、Action、Result等。下面根据 struts2.0体系架构图来解析这些组件之间的联系。

****

上图是Struts2 官网提供的Struts 体系架构图，依照此图，可以看出一个请求在Struts2.x 中处理步奏大概如下：

* 首先web容器一旦启动，就会初始化核心过滤器StrutsPrepareAndExecuteFilter，并执行初始化方法，完成Struts2初始化配置。
* 客户端初始化一个指向一个Servlet容器的请求(如Tomcat)，即客户端提交一个HttpServletRequest请求。
* 请求被提交到一系列的过滤器（Filter）。如ActionContextCleanUp（可选）、其他过滤器（SiteMesh等）、 FilterDispatcher。注意：这里是有顺序的，先ActionContext CleanUp，再其他过滤器（Othter Filters、SiteMesh等），最后到FilterDispatcher（FilterDispatcher是控制器的核心，就是MVC的Struts 2实现中控制层（Controller）的核心。）。
* 接着FilterDispatcher被调用，FilterDispatcher询问ActionMapper(根据请求uri,解析action名称和命名空间等)来决定这个请是否需要调用某个Action。
* 如果ActionMapper决定需要调用某个Action，FilterDispatcher把请求的处理交给ActionProxy 。
* ActionProxy通过Configuration Manager询问框架的配置文件（struts.xml），找到需要调用的Action类。
* ActionProxy创建一个ActionInvocation实例，同时ActionInvocation通过代理模式调用Action。但在调用之前，ActionInvocation会根据配置加载Action相关的所有Interceptor（拦截器）。
* 一旦Action执行完毕，ActionInvocation负责根据struts.xml中的配置找到对应的返回结果。返回结果通常是（但不总是，也可 能是另外的一个Action链）一个需要被表示的JSP或者FreeMarker的模版。