# Struts2

[Struts2 2](#_Toc19879)

[第一章 2](#_Toc18349)

[第一节 引言 2](#_Toc2427)

[第二节 MVC设计思想 3](#_Toc20770)

[第三节 前端控制器 4](#_Toc15592)

[第二章 5](#_Toc13206)

[第一节 Struts2简介 5](#_Toc23293)

[第二节 第一个Struts2程序 6](#_Toc22052)

[第三节 Eclipse支持struts.xml语法提示 7](#_Toc5515)

[第四节 struts2的简单执行流程 8](#_Toc30198)

[第五节 Struts2框架和应用的配置文件以及常见的常量 9](#_Toc9539)

[第六节 package,action,result的相关配置 10](#_Toc24670)

[第七节 Action类的三种编写方式 12](#_Toc25589)

[第八节 Action中多方法调用 13](#_Toc19118)

[第三章 14](#_Toc20859)

[第一节 访问ServletApi三种方式 14](#_Toc7642)

[第二节 Action获取请求参数三种方式 16](#_Toc8540)

[第三节Struts2拦截器概述 19](#_Toc31787)

[第四节Struts2中Action的执行流程 20](#_Toc7724)

[第五节Struts2内置拦截器 21](#_Toc14733)

[第六节 自定义拦截器 23](#_Toc22828)

[第七节 Struts2国际化 27](#_Toc826)

[第四章 28](#_Toc4031)

[第一节 OGNL和ValueStack(值栈) 28](#_Toc6610)

[第二节 Struts2的输入校验 32](#_Toc15785)

[第三节 输入校验的原理 33](#_Toc22814)

[第四节 Struts2的文件上传 35](#_Toc19005)

[第五节 文件下载[[t 37](#_Toc25367)

[第六节 Struts2标签 38](#_Toc28630)

## 第一章

### 第一节 引言

一、什么是框架

框架从何而来,为什么使用框架?

1.是一系列jar包的集合,其本质是对JDK功能的拓展.

什么是jar,jar有什么作用: jar其实就是class文件的打包,而程序运行只需要有字节 码即可.

JDK/JRE的功能:JDK只能开发和运行JavaSE.

2.框架是一组程序的集合,**包含了一系列的最佳实践**,作用是**解决某一个领域**的问题.

不同类型的框架解决了不同领域的问题.

3.框架其实就是一个半成品,一般的,我们做开发是基于框架,在框架上继续做开发.

二、 什么是最佳实践

最佳实践(Best Practice):实际上是无数程序员经历过无数次尝试之后,总结出来的**处理特定问题的特定方法**.

如果把程序员的自由发挥看作是一条通往**成功的途径**,最佳实践就是其中的最短路径,能极大的解放生产力,提高了效率.

最佳实践三要素:可读性,可维护性,可拓展性.

简单就是美:

消除重复

化繁为简

简单必须可读,简单必须可拓展

**减少依赖,消除耦合**

三、 Web中的最佳实践

Web开发中的最佳实践:分层开发模式(技术层面根据功能职责的不同,"分而治之"):

JavaEE开发根据职责的纵向划分:**表现层,业务层,持久层**: (SSH)

表现层(Predentation Layer):负责处理与界面交互的相关操作 (Struts2/Spring MVC)

业务层(Business Layer):负责复杂的业务逻辑计算和判断 (Spring)

持久层(Persistent Layer):负责将业务逻辑数据进行持久化存储(DAO) Hibernate/MyBatis

我们习惯吧**表现层的框架,称之为MVC框架**.

### MVC设计思想

1. 原理

MVC框架的功能作用(WEB开发常见功能):

MVC令程序开发有章可循,撇开框架,但是表现层的困惑也就出来了.

表现层需要处理的功能(回顾之前Servlet做了哪些功能):

设置编码,接受请求参数,参数类型转换,把参数封装成对象,响应视图,

输入验证,文件上传,文件下载,国际化(i18n),令牌机制,自定义标签.

------------------------------------------------------------

MVC框架至少就必须具有上述的功能.

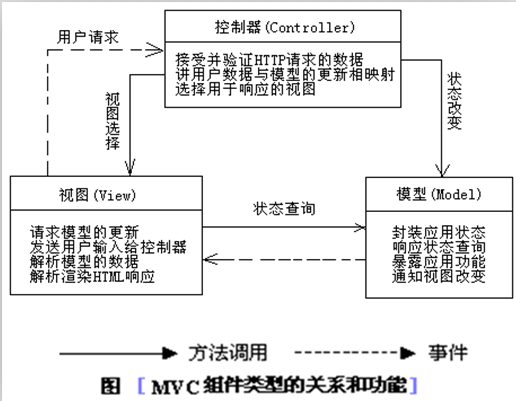
1. 功能职责

MVC架构型模式，它本身并不引入新的功能，只是用来指导我们改善应用程序的架构，使得应用的模型和视图相分离，从而得到更好的开发和维护效率。

数据模型（Model）：负责封装应用的状态，并实现应用的功能。通常又分为数据模型和业务逻辑模型，数据模型用来存放业务数据，比如订单信息、用户信息等；而业务逻辑模型包含应用的业务操作，比如订单的添加或者修改等。

视图展现（View）：界面，用来将模型的内容展现给用户，用户可以通过视图来请求模型进行更新。视图从模型获得要展示的数据，然后用自己的方式展现给用户，相当于提供界面来与用户进行人机交互；用户在界面上操作或者填写完成后，会点击提交按钮或是以其它触发事件的方式，来向控制器发出请求。

控制器（Controller）：用来控制应用程序的流程和处理视图所发出的请求。当控制器接收到用户的请求后，会将用户的数据和模型的更新相映射，也就是调用模型来实现用户请求的功能；然后控制器会选择用于响应的视图，把模型更新后的数据展示给用户。



### 第三节 前端控制器

Front Controller模式要求在WEB应用系统的前端（Front）设置一个入口控制器（Controller），所有的request请求都被发往该控制器统一处理。

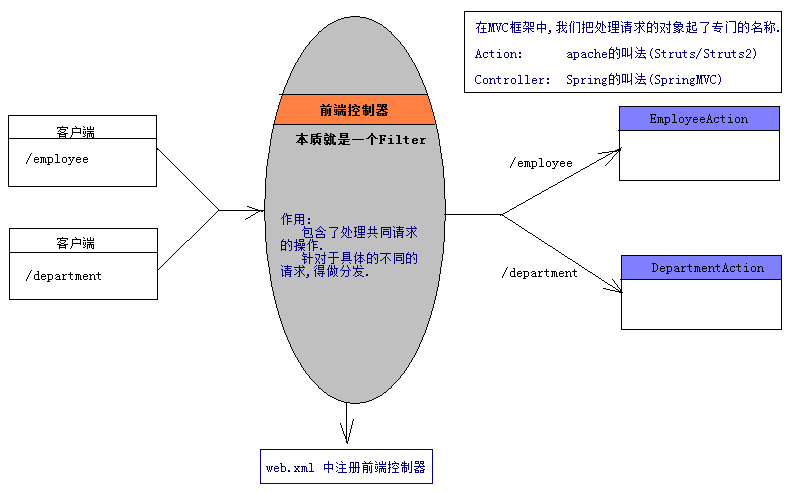
Front Controller一般可以用来做一个共通处理比如认证，页面导航，Session管理，国际化或本地化处理等。

优点：

1. 集中控制

2. 提高可管理性和安全控制能力

3. 提高可重用性可扩展性



## 第二章

### Struts2简介

Struts2的前世今生：

1.早期开发模型Servlet+JSP+JavaBean(Model2)显得力不从心：

流程凌乱、数据传递无序、缺乏辅助功能。

2.MVC模式的轻量级Web应用框架:**Apache Struts1很快风靡全球**。

代码结构划分合理，实用工具框架（如验证框架、国际化框架）等,搞J2EE必会Struts。

3.时间推移，Struts1的缺点：

线程不安全、灵活性低、和ServletAPI耦合、页面传值麻烦等。

4.异军突起，**SpringMVC**和OpenSymphony的**WebWork**等。

5.Apache Struts + OpenSymphony WebWork2 = Struts2

Struts2:基于MVC的轻量级的Web应用框架,

来源于Webwork2, 与Struts1.x完全不兼容

====================================================

Struts2 是一个非常优秀的MVC框架，基于Model2 设计模型.

由传统Struts1和WebWork两个经典框架发展而来:

Struts2框架=Struts2+XWork

Strust2 核心功能:

允许POJO（Plain Old Java Objects）对象 作为**Action**.

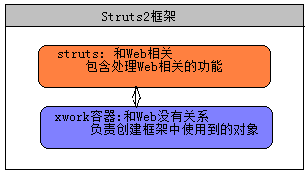
Action的execute 方法不再与Servlet API耦合，更易测试

支持更多视图技术（JSP、FreeMarker、Velocity）

基于**Spring AOP思想**的拦截器机制，更易扩展

更强大、更易用输入校验功能

整合Ajax支持



**struts2目录结构:**

apps:该文件夹包含了基于struts2 的示例应用，这些示例应用对于学习者是非常有用的

docs:该文件夹下包含了struts2 相关文档，包括struts2 快速入门、struts2的文档以及API文档等

lib:该文件夹下包含了Struts2框架和核心类库，以及struts2第三方插件类库

src: 该文件夹下包含了Struts2框架的全部源代码

### 第一个Struts2程序

**Struts2第一个程序的开发步骤:**

1. **拷贝/apps/struts2-blank/WEB-INF/lib中的jar到项目的lib中.**

注意:

开发Struts2依赖的jar,不需要挨个去挑选:

找到根下apps目录,找到struts2-blank(Struts2提供的空白项目),里面整合了Struts2的依赖环境和配置.

Struts运行必要jar包如下:

struts2-core-2.3.1.1.jar：Struts 2框架的核心类库

xwork-core-2.3.1.1.jar：Command模式框架,WebWork和Struts2都基于xwork

ognl-3.0.3.jar：对象图导航语言(Object Graph Navigation Language), struts2框架通过其读写对象的属性

freemarker-2.3.18.jar：Struts 2的UI标签的模板使用FreeMarker编写

commons-logging-1.1.x.jar：ASF出品的日志包，Struts 2框架使用这个日志包来支持Log4J和JDK 1.4+的日志记录。

commons-fileupload-1.2.2.jar： 文件上传组件，2.1.6版本后需要加入此文件

commons-io-2.0.1.jar：传文件依赖的jar包

commons-lang-2.5.jar：对java.lang包的增强

1. **在web.xml中配置Struts2的前端控制器-StrutsPrepareAndExecuteFilter.**

<filter>

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

<filter-class>

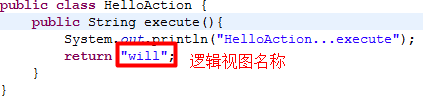
org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter

</filter-class>

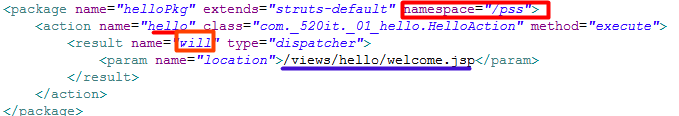
</filter>

**3.拷贝struts.xml文件到项目的source folder中.**

**4.定义一个POJO类:HelloAction,并提供一个公共无参数的execute方法.**



**5.在struts.xml文件中,配置HelloAction.(把HelloAction交给Struts2框架管理).**



**6.部署Web项目,访问Action.**

访问格式: http://ip:port/contextPath/namespace/actionName[.action]

上图的访问格式:http://localhost:80/pss/hello[.action]

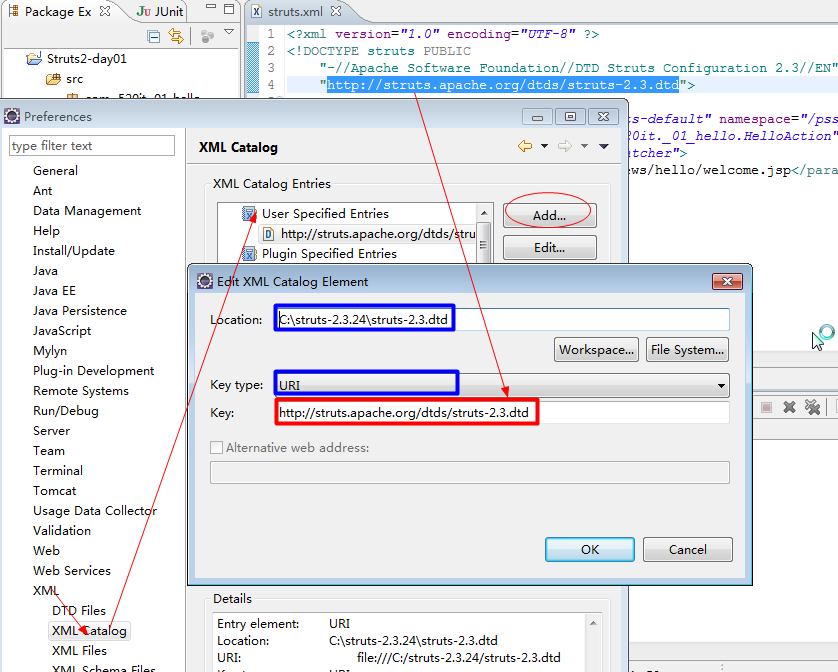
### Eclipse支持struts.xml语法提示

Eclipse支持struts.xml语法提示:

方式1:让电脑联网,去下载XML所依赖的dtd文件.

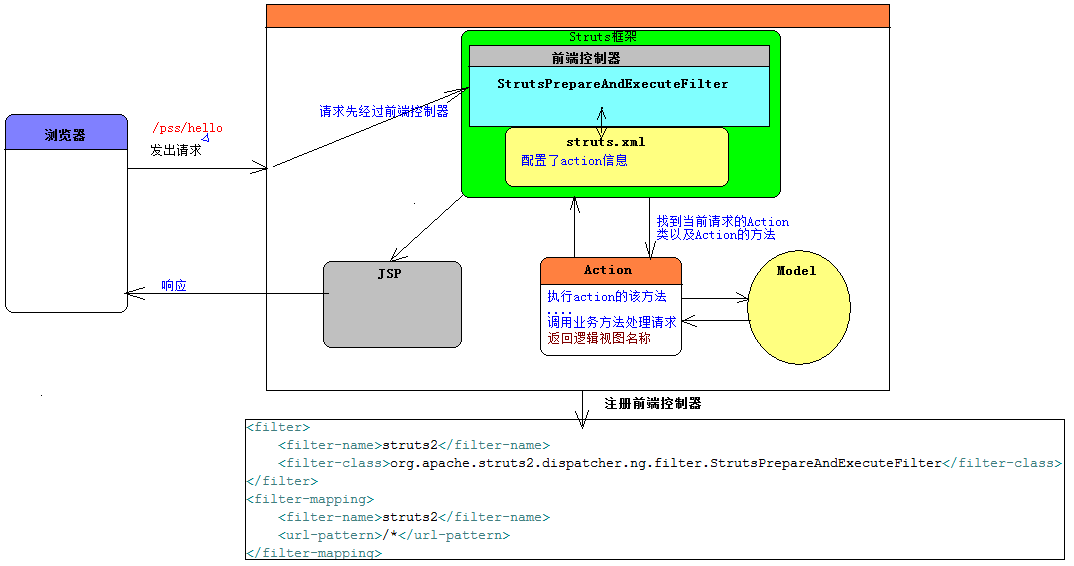
方式2:手动关联xml和dtd文件(推荐).

存在路径:struts-2.3.24\src\core\src\main\resources\struts-2.3.dtd.



### 第四节 struts2的简单执行流程

这里先简单的,片面的分析一下Struts2执行流程,在学习了拦截器之后,会重新来讲Action的完整执行流程.



### Struts2框架和应用的配置文件以及常见的常量

在大部分应用里，随着应用规模的增加，系统中Action的数量也会大量增加，导致struts.xml配置文件变得非常臃肿。为了避免struts.xml文件过于庞大、臃肿，提高struts.xml文件的可读性，我们可以将一个struts.xml配置文件分解成多个配置文件，然后在struts.xml文件中**包含**其他配置文件。下面的struts.xml通过<include>元素指定多个配置文件：

struts.xml:

<struts>

<include file="struts-part1.xml"/>

<include file="struts-part2.xml"/>

</struts>

**Struts2中的6大配置文件:**

-----------------------------------------

Struts2框架按照如下顺序加载struts2配置:

1.default.properties 该文件保存在 struts2-core-2.3.7.jar 中 org.apache.struts2包里面:包含了Struts2的默认常量配置

2.struts-default.xml 该文件保存在 struts2-core-2.3.7.jar:包含了框架依赖的对象配置和结果类型,拦截器等配置.

3.struts-plugin.xml 该文件保存在Struts2框架的插件中:struts-Xxx-plugin-2.3.7.jar.由插件提供

上述三个文件是框架自带的,我们不能修改,只能使用.

4.struts.xml 该文件是web应用默认的struts配置文件.重点.配置自定义的Action和其他信息.

5.struts.properties 该文件是Struts的默认配置文件-->可以修改default.properties 的常量配置.

6.web.xml 该文件是Web应用的配置文件

上述三个文件是我们可以修改操作的.

---------------------------------------------------------

如果多个文件配置了同一个struts2 常量，则后一个文件中配置的常量值会覆盖前面文件配置的常量值.

注意:一般的,我们只在struts.xml中做常量配置.



**Struts2中常见常量:**

---------------------------------------

1):指定默认编码集,作用于HttpServletRequest的setCharacterEncoding方法和 freemarker、velocity的输出

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

2):该属性指定需要Struts 2处理的请求后缀，该属性的默认值是action，即所有匹配 \*.action的请求都由Struts2处理。

如果用户需要指定多个请求后缀，则多个后缀之间以英文逗号（,）隔开

<constant name="struts.action.extension" value="action,,"/>

3):设置浏览器是否缓存静态内容,默认值为true(生产环境下使用),开发阶段最好关闭

<constant name="struts.serve.static.browserCache" value="false"/>

4):当struts的配置文件修改后,系统是否自动重新加载该文件,默认值为false(生产环境下 使用),开发阶段最好打开

<constant name="struts.configuration.xml.reload" value="true"/>

5):开发模式下使用,这样可以打印出更详细的错误信息

<constant name="struts.devMode" value="true" />:修改struts.xml之后,不要重启Tomcat.

6):默认的视图主题

<constant name="struts.ui.theme" value="simple" />

7):是否支持动态方法调用

<constant name="struts.enable.DynamicMethodInvocation" value="false"/>

### package,action,result的相关配置

**<package>元素**: 是<struts>根元素的子元素.

用来对多个<action>元素分类管理,和Java中的package没有关系.

常见的属性:

<package name="" extends="" namespace="" abstract=""></package>

name: 表示<package>的名字,但是要保证不同的<package>元素的name不同. 可以通过该名字被其他的包所指代.

extends: 表示当前<package>继承哪一个<package>,一般都是:struts-default.

而struts-default其实就是struts-default.xml中<package>元素的名字.

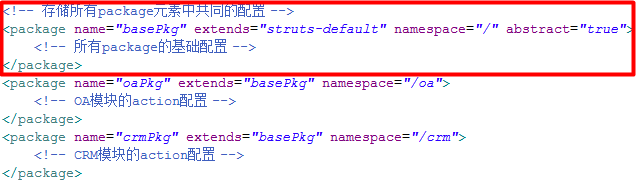
继承struts-default之后,就拥有了该<package>定义的所有资源.(结果返回类型,拦截器..)



namespace: 表示命名空间,一般的以"/"打头,命名一般以模块名.如: /crm, /oa. 和<action>的name决定了一个Action类的访问路径.

abstract: 抽象的,缺省值是false. 若一个<package>的abstract="true",那么该<package>中就不能再定义<action>元素,只能用来继承.

**企业中的最佳实践:**



------------------------------------------------------------------------------

**<action>元素**:是<package>元素的子元素.

专门用来配置Action对象的.

<action name="" class="" method=""/>

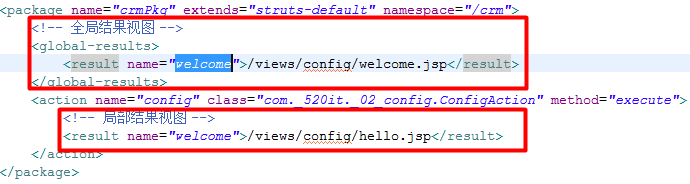
常见的属性:

name: action的名称,在同一个<package>中,action的名字必须唯一. 和<package>的namespace共同决定了一个Action类的访问路径.

注意:action的name值不能以"/"打头.

class:一个Action类的全限定名. **缺省值:ActionSupport类.**

method:当前Action动作访问的方法, **缺省值:execute.**



------------------------------------------------------------------------------

**<result>元素**:配置结果视图.

<result name="" type=""></result>

局部结果视图: <result>定义在<action>中.

全局结果视图: <result>定义在<global-results>中,而<global-results>在<package>中

如右图:

逻辑视图寻找顺序:先从当前的Action中去选择局部结果视图,找到就跳转,找不到,再找当前action所在package中找全局的结果视图,找到,跳转,找不到,报错.No result defined.

----------------------------------

常见的属性:

name:Action方法返回的逻辑视图名称. **缺省值:success**

type:结果的跳转类型.该类型的值在struts-default.xml中已经预定义好了. **缺省值:dispatcher**

**常见的type值(结果类型):**

**dispatcher:** 表示从Action请求转发到页面(JSP).**缺省值**

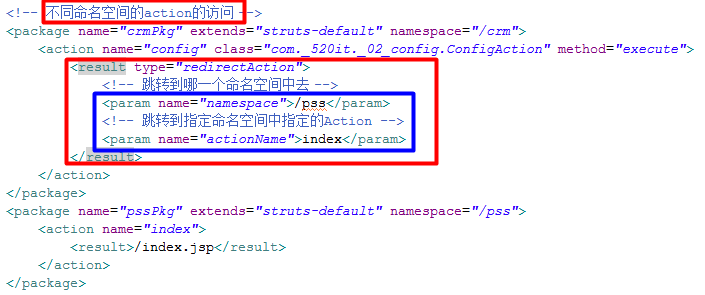
redirect: 表示从Action重定向到页面(JSP).

chain: 表示从Action请求转发到另一个Action.

redirectAction: 表示从Action重定向到另一个Action.

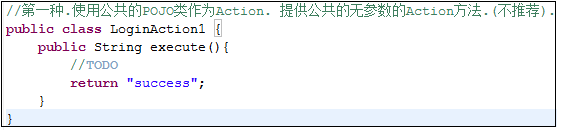
stream: 表示返回流. 文件下载时使用.

<param name="">表示参数:name缺省值:location(地址)



### Action类的三种编写方式

**第一种.使用公共的POJO类作为Action. 提供公共的无参数的Action方法.(不推荐).**



缺点:

没有一种方式约束Action方法必须是公共的无参数的.

Action方法的返回逻辑视图名可以自定指定. 有时起名不规范. 比如:"ooxx".

解决方案:第二种.

**第二种.定义一个类,实现于com.opensymphony.xwork2.Action接口.并覆写execute方法即可.(不推荐)**

Action接口中,不仅提供了Action方法的声明,也提供了常用的逻辑视图名称:

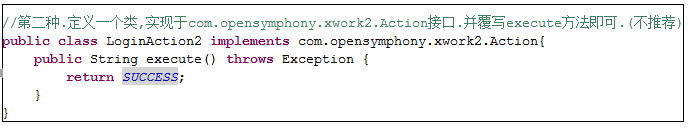
public static final String SUCCESS = "success";

public static final String NONE = "none";

public static final String ERROR = "error";

public static final String INPUT = "input";

public static final String LOGIN = "login";



缺点:

不支持国际化,数据校验,消息机制.

解决方案:第三种:

**第三种.定义一个类,继承于com.opensymphony.xwork2.ActionSupport类.(推荐)**

public class ActionSupport implements Action, Validateable, ValidationAware, TextProvider, LocaleProvider, Serializable {}

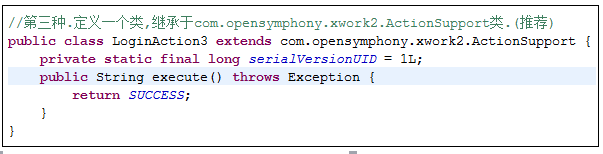
---------------------------------------------

真实开发中,我们却往往再提供一个BaseAction类.

--BaseAction extends ActionSupport

-----AAction extends BaseAction

-----BAction extends BaseAction



### Action中多方法调用

问题:在一个Action中可能包含多个Action方法,比如EmployeeAction就包含处理员工操作的CRUD等方法.

Action中多个Action方法会造成<action>配置的臃肿(右图).

解决方案:

**方案1: DMI:动态方法调用 :官方不推荐.**

格式: action名!方法名

比如: emp!edit emp!list

在Struts2新的版本中,默认的关闭了DMI.若我们需要使用DMI,就需要配置常量,启用动态方法调用.

此时:<action/>元素不需要指定method属性值.

<constant name="struts.enable.DynamicMethodInvocation" value="true"/>

---------------------------------------------------------------------------------

**方案2: 使用通配符的方式类配置: 通配符:\***

<action name="emp\_\*" class="com.\_520it.manymethod.EmployeeAction" method="{1}">

Action的名字: emp\_Action方法: 比如:emp\_list,那么{1}的值就是list

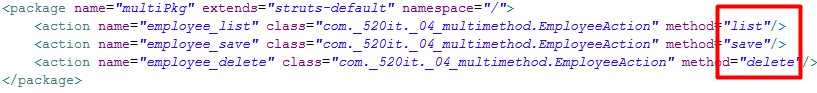
emp\_edit,那么{1}的值就是edit

两个通配符:

<action name="\*\_\*" class="com.\_520it.manymethod.{1}Action" method="{2}">

Action名字:Action类名\_Action方法. 比如:Employee\_list,表示调用的EmployeeAction中的list方法

比如:Department\_edit,表示调用DepartmentAction中的edit方法.



## 第三章

### 访问ServletApi三种方式

Action访问Servlet API:

在Action类中操作Servlet中相关的对象.

request,repsonse,session,cookie等.

获取请求参数: request.getParameter(String name)

设置/获取共享数据: 域对象.setAttribute(String name,Object value);

操作Cookie: response.addCookie(Cookie cookie);

依赖servlet-api.jar

-----------------------------------------------------------------------------

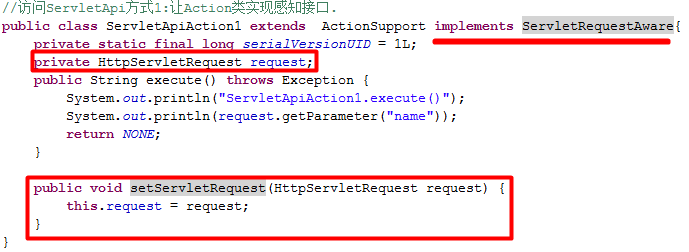
**解决上述问题的方式:**

**访问ServletApi方式1:让Action类实现感知接口.**

ServletRequestAware：感知HttpServletRequest对象；

ServletResponseAware：感知HttpServletResponse对象；

ServletSessionAware：感知HttpSession对象；

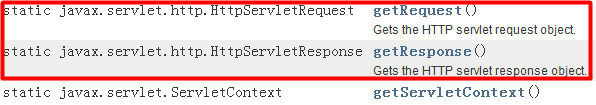


存在的问题，和ServletAPI藕合严重；

问题:Action中感知接口的方法是谁在调用?----->servletConfig拦截器(优先于Action执行)

**访问ServletApi方式2:通过ServletActionContext工具类.**

**通过ServletActionContext类中的静态方法，得到Servlet相关的Api；**



这种方式直接通过ServletActionContext的静态方法就可以获取Servlet的API对象,操作和理解非常简单.

在开发中很多人都喜欢使用,但是依然Action与ServletAPI有耦合.

**访问ServletApi方式3:通过ActionContext工具类,**

理解ActionContext:从字面上分析,表示Action的上下文对象.ActionContext封装每一次请求的相关信息.

获取ActionContext对象:

ActionContext ctx = ActionContext.getContext();

在ActionContext中把request,session,application三大作用域都作为Map对象.

---------------------------------------------------------------------------------------------------------------------

1):获取请求参数.

原始:Map<String,String[]> params = request.getParameterMap();

现在:Map<String,String[]> params = ctx.getParameters();

2):操作request作用域:

原始:request.setAttribute(String name, Object value); Object val = request.getAttribtue(String name);

现在:ctx.put(String key,Object value); Object val = ctx.get(String name);

3):操作session作用域:

原始:HttpSession session =request.getSession();

现在:Map<String,Object> session = ctx.getSession(); 接下来就是操作Map对象

4):操作servletContext(application)作用域:

原始:ServletContext servletContext = request.getServletContext();

现在:Map<String,Object> app = ctx.getApplication();

### Action获取请求参数三种方式

在Servlet中获取请求参数,需要做哪些工作:

如果参数只有一个值: String xx = request.getParameter("参数名");

如果参数存在多个值: String[] xx = request.getParameterValues("参数名"); //多选

无论是什么类型参数,传递到后台都是**String类型**,但是我们需要在Servlet中做类型转换,很麻烦.**并且需要封装成对象,很麻烦**.

-------------------------------------------------------------------

J2EE的最佳实践:按照功能职责,分层开发(经典的三层架构:表现层/业务层/持久层):

**表现层**:

职责:

1.接受请求数据

2.把数据封装Model对象

3.调用业务逻辑方法处理请求

4.控制界面跳转

-------------------------------------------------

**MVC思想:**

M:Model: 数据模型对象:封装了数据和功能.

V: View : JSP

C: Controller: 控制请求的转发: StrutsPrepareAndExecuteFilter(前端控制器.)

问题:Action到底充当什么角色?

因为Action会返回逻辑视图名称,间接的控制了界面跳转,所以Action充当了控制器的角色.

如果Action封装了数据,此时Action也可以充当数据模型对象,如果Action不封装数据,不是数据模型对象.

------------------------------------------------------------------

**Action获取请求参数三种方式,归功于拦截器(ParametersIntercepter):**

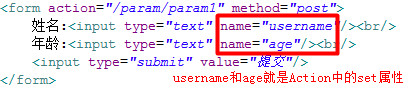
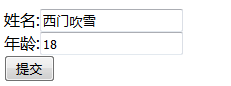
第一种:Action本身作为Model对象，通过setter方法封装(属性注入)

第二种:创建独立Model对象，页面通过ognl表达式封装(属性注入)

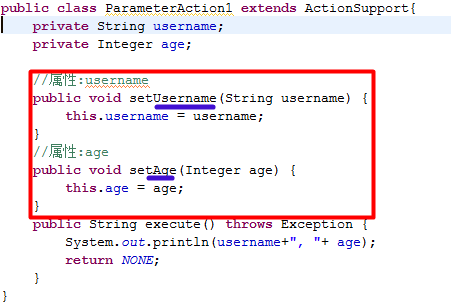
第三种:使用ModelDriven接口，对请求数据进行封装(模型驱动)

**方式1:Action本身作为Model对象，通过setter方法封装(属性注入).**

**JSP:**



**Action:**



Struts2内置有强大的类型转化器,帮我们把String类型的参数转换为指定的类型.

如果不够用,就得自定义类型转化器.

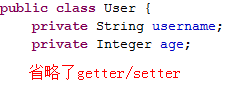
-------------------------------------------------------

方式1: 优点:简单,直观.

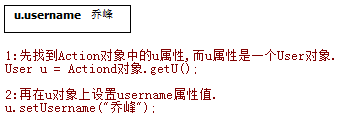
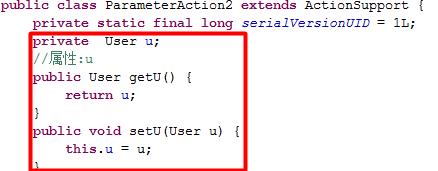
缺点:参数过多,Action比较臃肿.自己来封装成对象.

**方式2:创建独立Model对象，页面通过ognl表达式语法来封装对象.**

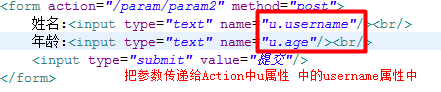
Model:



Action:

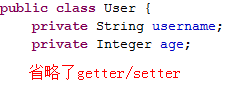


JSP:

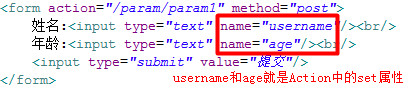


**方式3:使用ModelDriven接口，对请求数据进行封装.**

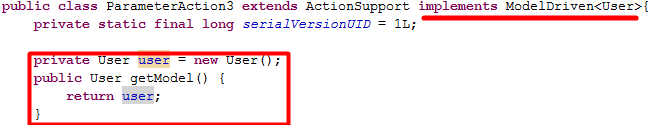
**Model:**



JSP:表单和方式1一样



Action:



Action获取请求参数三种方式如何选择???:

从操作便捷性来分析,首选使用方式2.

但是方式2必须保证请求参数名必须是Model中的一个属性.

举个例子.

在修改密码的时候:

oldpassword: 旧密码

newpassword: 新密码

confirmpassword: 确认密码

此时我们可以使用:方式1+方式2.或者方式1+方式3.

不推荐使用方式3.

### 第三节Struts2拦截器概述

**拦截器:Interceptor**

------------------------------

拦截器：Struts2拦截器是在访问某个Action或Action的某个方法之前或之后实施拦截，并且Struts2拦截器是可插拔的，拦截器是AOP的一种实现．

**AOP:面向切面编程**.其实现原理:动态代理模式--->留给Spring

WebWork中文文档解释：拦截器是动态拦截Action调用的对象。它提供了一种机制使开发者可以定义在一个Action执行的前后执行的代码，也可以在一个action执行前阻止其执行。同时也提供了一种可以提取Action中可重用的代码的方式。

拦截器栈（Interceptor Stack）:Struts2拦截器栈就是将拦截器按一定的顺序连接成一条链。在访问被拦截的方法或字段时，Struts2拦截器链中的拦截器就会按其之前定义的顺序被调用。

-------------------------------------------------------------------------

**拦截器的"美":**

---------------------------------------------------

DRY原则：Dont't Repeat Yourself.

拦截器在设计和程序结构上的优点:

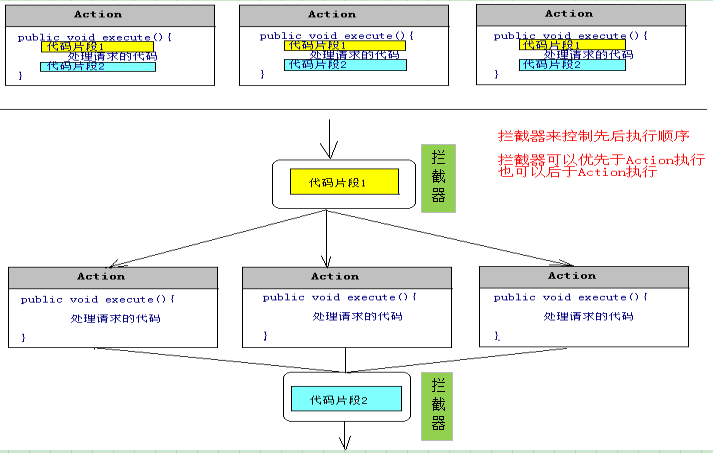
拦截器能把很多功能从Action中独立出来，分散到不同的拦截器里面，减少了Action的代码。如此，拦截器和Action本身的功能都更单一了。当通用的功能代码被封装在拦截器里面(代码模块化)，就可以对不同的Action，根据功能需要，来配置相应功能的拦截器了。提高了拦截器所实现的功能的重用性，也变相实现了装配式和可插拔式的体系结构，使得整个系统结构变得更灵活。

1.简化Action的实现

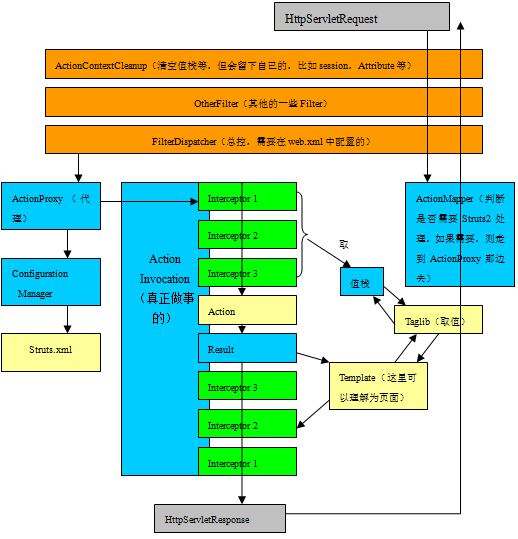
2.功能更单一

3.通用代码模块化

4.提高重用性



### 第四节Struts2中Action的执行流程



**Struts2中Action的执行流程**

----------------------------------------------------------

①.客户端发送请求；

②.该请求经过一系列的过滤器(Filter)：其中可选过滤器ActionContextCleanUp，帮助Struts2和其他框架集成。例如：SiteMesh Plugin。

③.接着FilterDispatcher被调用，FilterDispatcher询问ActionMapper，来决定该请求是否需要调用某个Action。

④.若ActionMapper决定需要调用某个Action，FilterDispatcher把请求的处理交给ActionProxy。

⑤.ActionProxy通过Configuration Manager询问框架的配置文件，找到需要调用的Action类。

⑥.ActionProxy创建一个ActionInvocation的实例。

⑦.ActionInvocation实例调用Action的前后，涉及到相关拦截器(Intercepter)的调用。

⑧.一旦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。返回结果是一个JSP或其他页面(也可以是其他的Action链)。 JSP页面展现可使用Struts2框架中的标签(该过程会涉及ActionMapper)。

-------------------------------------------------------------------

在上述整个过程中,Struts2框架中会产生很多对象,都是由**Struts2中的xwork容器负责创建的**.

===============================================================================

### 第五节Struts2内置拦截器

Struts2之所有强大,是因为它内置了很多拦截器,Struts2框架中自定义的,可以直接拿来使用的拦截器.

不同的拦截器栈(interceptor-stack)是不同拦截器的组合,用户根据不同的需求选择不同的拦截器栈.

内置拦截器在哪里,如何去看,如何去用,分别有什么作用?

找到:struts-default.xml文件:



**Struts2中内置的拦截器:**

在struts-core-2.3.x.jar--->struts-default.xml中

------------------------------------------

**常见的拦截器:**

1:params拦截器

这个拦截器**偷偷的**把请求参数设置到相应的**Action的属性去的，并自动进行类型转换**。

2.modelDriven拦截器

如果Action实现ModelDriven接口，它将getModel()取得的模型对象存入OgnlValueStack中。

3.execption拦截器

顾名思义，在抛出异常的时候，这个拦截器起作用。最好把它放在第一位，让它能捕获所有的异常。

4.validation拦截器

调用验证框架读取 \*-validation.xml文件，并且应用在这些文件中声明的校验。

5.token拦截器

核对当前Action请求（request）的有效标识，防止重复提交Action请求。

6.fileUpload拦截器

用来处理文件上传

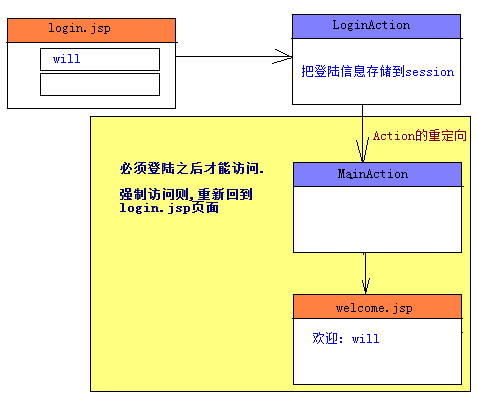
7.workflow拦截器

调用Action的validate方法，一旦有错误返回，重新定位到INPUT结果视图

8.servletConfig

通过感知接口,获取感应对象

### 第六节 自定义拦截器



自定义拦截器:CheckLoginInterceptor:

**步骤1.定义拦截器类**

方式1:实现com.opensymphony.xwork2.interceptor.Interceptor接口.

方式2:继承com.opensymphony.xwork2.interceptor.AbstractInterceptor类.



**步骤2.在struts.xml中注册拦截器,告诉Struts2框架来帮我管理CheckLoginInterceptor.**

第一步:先在<package>中声明拦截器CheckLoginInterceptor.

第二步:在<action>中来引用CheckLoginInterceptor拦截器.



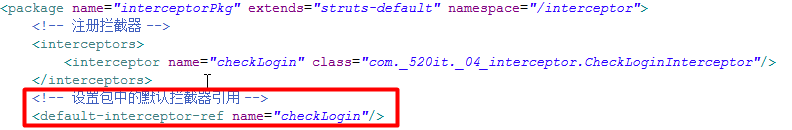
上述的拦截器配置没有问题.

我们说main这个action需要登陆之后才能访问,因此我们在main的action中,引用了checkLogin拦截器.

但是,如果该<package>中多个action都需要做登陆检查,此时每一个<action>元素中都得引用checkLogin拦截器-->重复.

解决方案:

解决方案,在当前<package>中引用拦截器,那么该<package>中所有的<action>都拥有该功能.如下.



但是如此一来,Action再也不能获取请求参数了!

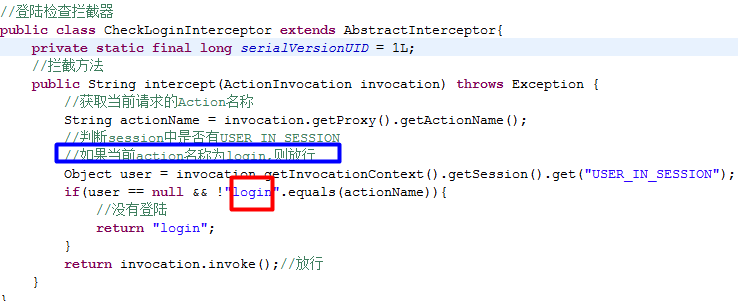
因为:<package>本身有默认的拦截器栈:defaultStack.

然后,我们修改了<package>的默认拦截器栈,因此就没有接受参数的拦截器了.

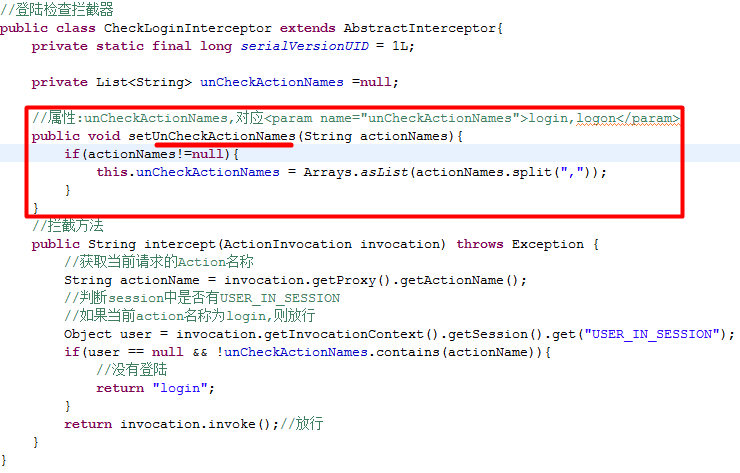
解决方案:我们可以定义自己的拦截器栈,包含defaultStack和checkLogin.



**给拦截器配置参数**







### 第七节 Struts2国际化

需求:国际化的登陆界面:

准备国际化资源文件:基本名词\_语言\_国家.properties

中文:app\_zh\_CN.properties

英文:app\_en\_US.properties

---------------------------------

告诉Struts2框架:国际化资源文件叫什么名字.

<constant name="struts.custom.i18n.resources" value="app"/>

---------------------------------

在jsp中引入Struts2标签:<%@ taglib uri="/struts-tags" prefix="s" %>

准备国际化的login.jsp

在JSP中,获取指定的国际化信息:(先引入Struts2的标签库)

<s:text name="资源文件中的key"/>

-----------------------------------

若要让Action支持国际化,那么Action必须继承ActionSupport或者ActionSupport的子类.

在Action中获取资源信息:

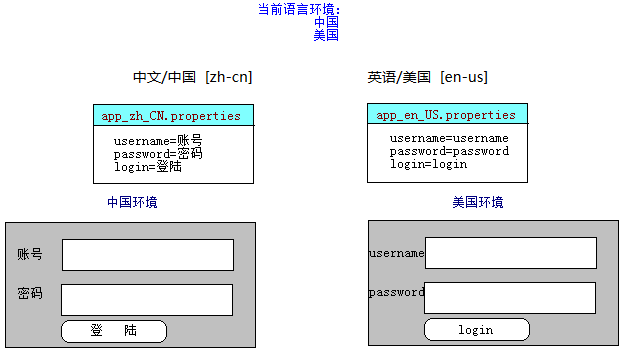
通过ActionSupport类中的:

String getText(String name,String[] args);

参数: name:表示资源文件中的key

args:资源文件中指定key占位符的真实值.

返回:拼接好的一个国际化信息.



## 第四章

### OGNL和ValueStack(值栈)

表现层中给页面(JSP)传递数据的方式:

1，通过HttpServletRequest对象传递；页面上使用EL表达式获取；

Servlet: request.setAttribute(String name,Object value);

JSP: ${name}

优点：易于理解；

缺点：过于依赖Servlet API；操作复杂，页面如果有大量数据，非常麻烦；页面取数据麻烦；

2， 通过Action传递，引入Struts2标签库；

1，通过HTTP对象传递数据，页面使用<s:property/>获取数据；

2，通过在Action使用ValueStack传递；页面使用<s:property/>获取数据；

引出OGNL和ValueStack

**1.什么是OGNL**

OGNL是Object Graphic Navigation Language（对象图导航语言）的缩写，它是一个开源项目。

Struts2框架使用**OGNL作为默认的表达式语言**。

EL(表达式语言),OGNL就是EL的升级版.

作用:Action和视图(JSP)之间**数据交互的桥梁**.

讲解OGNL之前,先得学习**ValueStack**.

--------------------------------------------------------------

**2.什么是ValueStack**

值栈是对应**每一个请求对象的轻量级**的**内存数据中心**。

每一次请求的时候,都会创建一个新的ValueStack对象,该ValueStack对象封装了这一次请求相关的数据信息.

1).ValueStack实际是一个接口,在Struts2中利用OGNL时，实际上使用的是实现了该接口的**OgnlValueStack类**,这个类是Struts2利用OGNL的基础。

2).ValueStack贯穿整个Action的生命周期(一次请求)：每个Action类的实例都拥有一个ValueStack对象。 ValueStack相当于数据的中转站，在其中保存该次请求和当前Action对象和其他相关对象信息。

3).Struts2框架把ValueStack对象保存在名为“struts.valueStack”的request属性中。

新的请求发出:

ValueStack vs = Struts2框架创建ValueStack对象;

request.setAttribute("struts.valueStack",vs);

**3,如何获取ValueStack.**

方式1: 因为ValueStack在请求中,属性名为:struts.valueStack.

ValueStack vs = request.getAttribute("struts.valueStack");

ValueStack vs = ServletActionContext.getRequest().getAttribute("struts.valueStack");

ValueStack vs = ServletActionContext.getRequest().getAttribute(ServletActionContext.STRUTS\_VALUESTACK\_KEY);

方式2: 通过ActionContext对象获取

ValueStack vs = ActionContext.getContext().getValueStack();

以上两种方式获取的是同一个ValueStack对象.

**4.ValueStack内部结构**

ValueStack对象中有两个很重要的属性,这两个属性就是ValueStack的主要内部结构:

------------------------------------

root: 类型: CompoundRoot extends ArrayList : 表示栈的数据结构(后进先出)

context: 类型: Map: : 上下文

root:主要存储Action对象相关的数据信息.

context:主要存储映射关系数据(key-value).

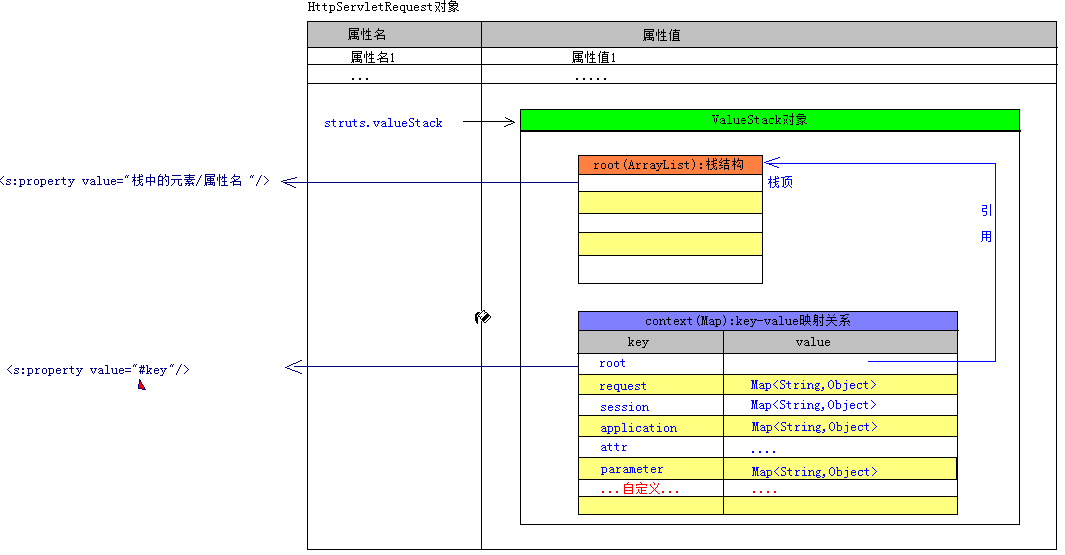
context存在用root对象的引用(只要拿到context对象就可以获取到root对象):

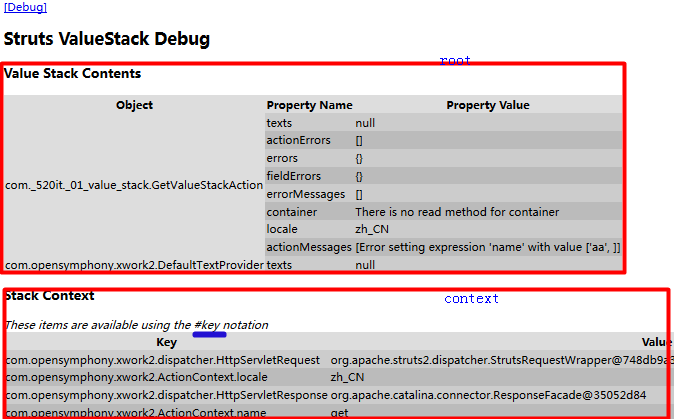
context中还存在request、session、application、attr、parameters对象的引用。

获取到context,就可以获取ValueStack中所有的数据.

从root中获取数据: 直接使用属性名获取. ---><s:property value="属性名"/>

从context中获取数据: #key ---><s:property value="#key"/>





5.把数据放入ValueStack.

1).把数据放入root中:(栈,ArrayList.每次都是压在栈顶)

方式1:ValueStack对象.getRoot().add(0, Obejct val);//把数据压入栈顶

方式2:ValueStack对象.getRoot().push(Object val):等价于valueStack对象.getRoot().add(0, Obejct val);

方式3:ValueStack对象.set(String propertyName,Object value);

方式4.在Action中提供一个可访问的属性(getter方法).

一般的,把单个对象可放入root中,就在栈顶.

-------------------------------------------------------------------

2).把数据放入context中:

方式1:ValueStack对象.getContext().put(String key,Object value); 太长了.

方式2:ActionContext对象.put(String key,Object value);

一般的:把集合中的数据放入context中.

---------------------------------------------------------------------------------------------------------

6.如何从JSP中取出ValueStack中的数据:

此时必须使用Struts2的标签.--->先引入Struts2标签

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

Struts2的调试标签:<s:debug/> :主要用来观察数据在哪一个空间(root,context).

访问方式:<s:property value="OGNL表达式"/>

1).获取root中数据:

若:放入root中的数据,没有属性名:<s:property value="[0].top"/>

若:放入root中的数据有属性名: <s:property value="属性名"/>

2).把context中数据:

<s:property value="#key"/>

--------------------------------------

EL可以访问ValueStack中的数据:

不建议这么做:

为什么可以呢: Struts2重新包装而来请求对象.${msg}---><%=pageContext.findAttribute("msg") %>

StrutsRequestWrapper:先从ValueStack中取出数据,再放入request中.

### Struts2的输入校验

输入校验:

一.客户端校验: 过滤正常用户的误操作，通过JS代码完成(可以绕开).

二.服务端校验: 整个应用阻止非法数据的**最后防线(必须)**,使用Java代码.

三.数据库校验: 在数据库中通过约束来限制列的存储.

------------------------------------------------------------------

Struts2校验方式:

1.编写代码方式校验:开发中常用；

2.配置校验:

2.1.基于XML配置校验(推荐使用)。

2.2.基于Annotation配置校验。



Struts2中做校验:

Action中

\* 要继承ActionSupport

\* 重写Validateable接口中的validate()方法 ,在该方法中完成验证

\* 步骤如下:

\* validate()方法在其他的业务方法之前执行

\* 验证出错转向的页面

struts.xml配置<result name="input">/validate/login.jsp</result>

其中input转向是在action中已经定义好的.

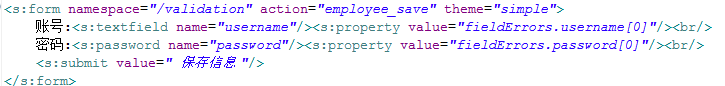
public static final String INPUT = "input";

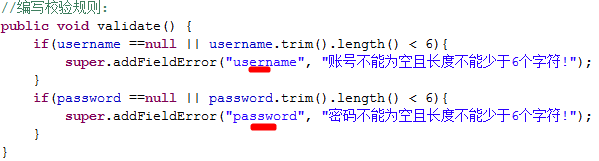
\* 什么时候表示验证出错(转向input所指向的页面)

\* super.addFieldError("字段名称", "错误信息");

\* 当集合不为空时,转向错误页面.

\* 若校验失败,不会再执行Action中的业务方法.





### 输入校验的原理

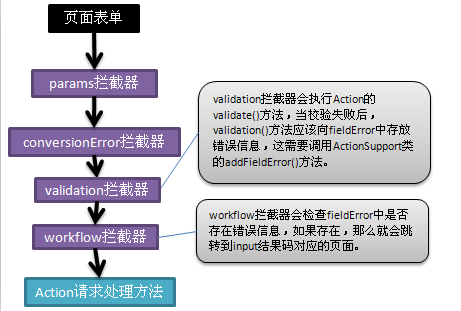
通过5个问题,搞明白输入校验的原理以及问题.

问题1:为什么在执行input方法之前,没有执行validate方法.

在validation拦截器中,标识需要排除的方法:

问题2:为什么校验失败,默认跳转到input结果视图.

在workflow拦截器中,如果有错误,默认的结果视图就是input.



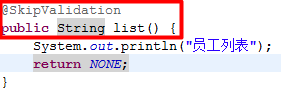


问题3:Action中某些方法不需要校验,却也执行了validate方法,如何避免问题.

比如,我们在做员工列表,或者员工删除的时候,没必要执行校验的功能.

只有表单的提交,才有必要执行校验.

在不需要被校验的方法上,使用标签:@SkipValidation.



问题4.开发中在一个Action中,被校验的方法是少数,大多数方法都不需要校验.

难道大多数方法都要使用@SkipValidation标签吗?--->麻烦.

解决方案: 就只针对于需要被校验的方法,编写校验规则的方法.

public void validate(){}:默认对Action中所有方法做校验.

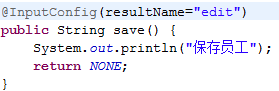
public void validateXxx():Xxx表示需要被校验的方法名称,那么该方法就只对xxx方法做校验.

比如针对save方法: public void validateSave(){};

问题5:从workflow的拦截可以看出,校验失败会跳到默认的结果视图("input").

比如,要求校验失败之后不要挑战到input视图,而是edit视图.

解决方案:在被校验的方法上,使用@InputConfig标签.



### Struts2的文件上传

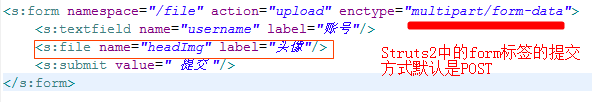
文件上传准备:

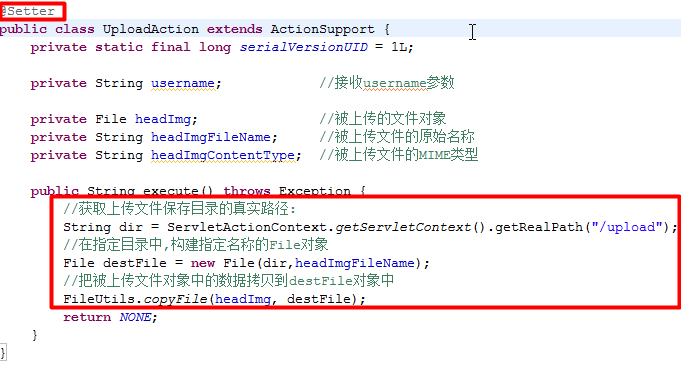
注意:

1):表单必须使用POST方式提交;

2):使用二进制编码:multipart/form-data;

3):<input type="file" name=""/>;

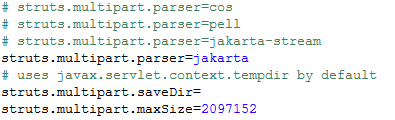




Struts2文件上传的细节:

Struts2中的FileUploadInterceptor拦截器完成了文件上传的操作.

Struts2中支持哪些上传的组件:

 Struts2支持cos,pell,jakarta文件上传组件,默认使用的jakarta(Apache).

struts.multipart.saveDir= 上传文件的临时目录,默认是在Tomcat下面.

struts.multipart.maxSize=2097152 一次请求允许上传文件的最大字节数,默认是2M.

--------------------------

控制上传文件的大小和文件类型.

1):控制单个文件的最大字节.

2):控制上传文件的类型,比如png,jpg,jpeg,bmp.

---------------------------------------------------------

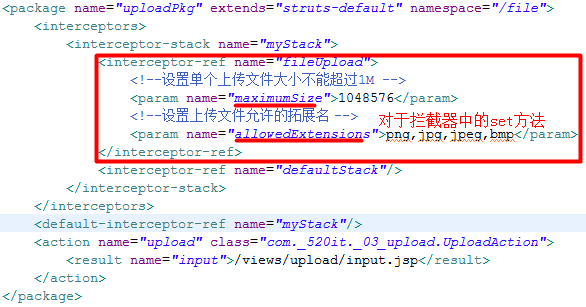
问题:上传失败之后,提示信息看不懂.

解决方案:把提示信息作出国际化.支持中文.

提示信息都在struts-messages.properties 文件里预定义. (org.apache.struts2包下)

在struts.xml中引用资源文件:

<constant name="struts.custom.i18n.resources" value="fileuploadMsg"/>



### 文件下载

Struts2提供了**stream结果类型**，该结果类型就是专门用于支持文件下载功能的。

(阅读stream结果集代码)

指定stream结果类型 需要指定一个 inputName参数，该参数指定一个输入流，提供被下载文件的入口

----------------------------------------------------------------------------------------------

<action name="" class="">

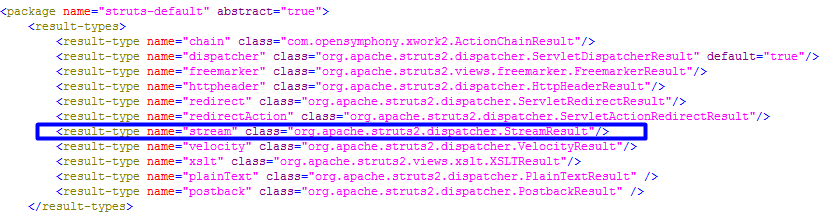
<result name="success" type="stream">

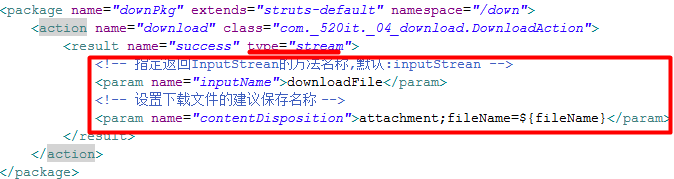
<param name="contentDisposition">attachment;fileName="${fileName}"</param>

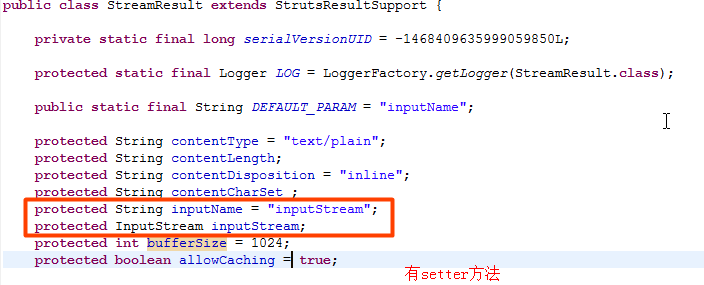
<param name="inputName">downloadFile</param>

</result>

</action>







### Struts2标签

Struts2的标签库:

Struts2中表单标签将在 HTML 里input元素不同的type被呈现为一个表单元素.

使用表单标签的优点:

表单数据回显

对页面进行布局和排版

-----------------------------------------------------

**Struts2标签:**

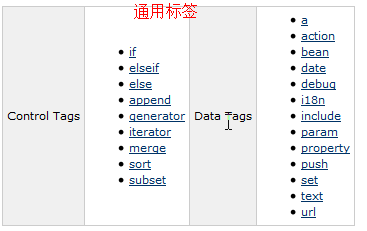
通用标签

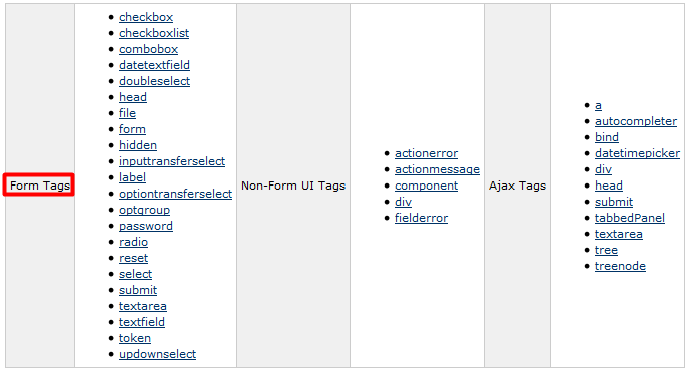
控制标签/数据标签

UI标签

引用标签库:

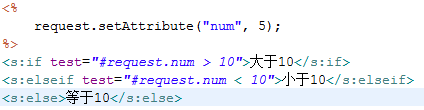
<%@ taglib uri="/struts-tags" prefix="s"%>





UI标签

1:控制流程的标签:if-else if-else.



2:迭代标签:<s:iterator/>:

