# 叩丁狼教育

课程:Struts2

讲师:钟昕灵

更多的教程,可以通过扫描屏幕下方的二维码轻松获取

# Struts2

## 第一章 概述

### 第一节 基本概念

**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.

优秀的程序需要做到尽量简单,而简单的程序需要使用上面的三要素来决定

**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

**四、 MVC思想**

在web开发中,我们需要将视图和业务逻辑代码分离开,所以提出了MVC思想提高代码的可读性和可维护性

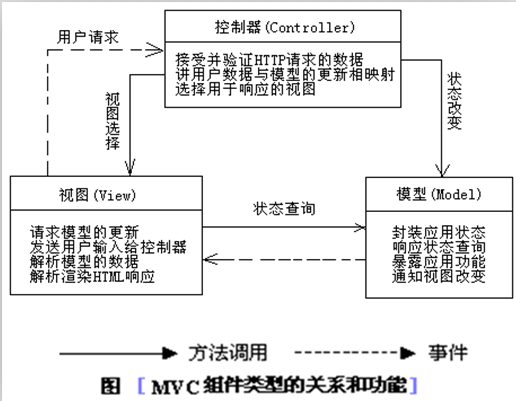
1. 功能职责

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。(Service/DAO/Domain)

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。(JSP)

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。(Servlet)



通常,我们习惯将表现层的框架称之为MVC框架.

### 第二节 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。

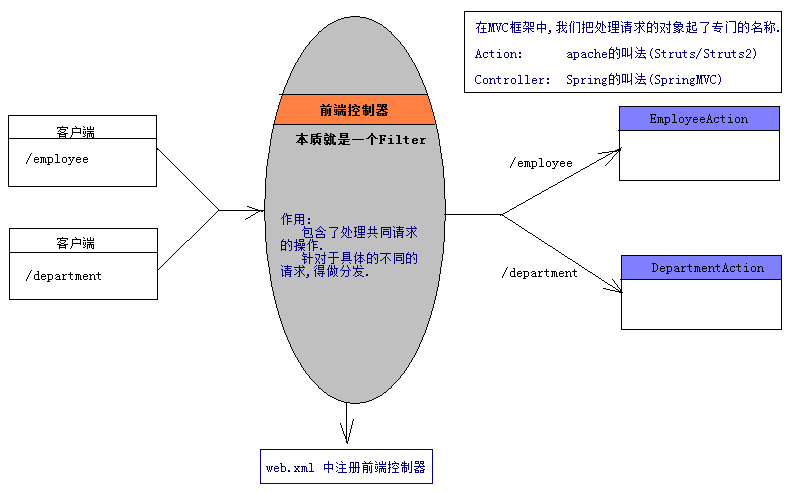
Front Controller一般可以用来做一个共通处理比如认证，页面导航，Session管理，国际化或本地化处理等。

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 基础入门

### 第一节 Struts2简介

Struts2的前世今生：

1. 2000年5月发展至今，Struts1已经成为了一个高度成熟的框架，不管是稳定性还是可靠性都得到了广泛的证明。市场占有率超过20%，拥有丰富的开发人群，几乎已经成为了事实上的工业标准。但是随着时间的流逝，技术的进步，Struts1的局限性也越来越多地暴露出来，并且制约了Struts1的继续发展。
2. 对于Struts1框架而言，由于与JSP/Servlet耦合非常紧密，因而导致了一些严重的问题。首先，Struts1支持的表现层技术单一。由于Struts1出现的年代比较早，那个时候没有[FreeMarker](https://baike.baidu.com/item/FreeMarker" \t "https://baike.baidu.com/item/Struts%202/_blank)、[Velocity](https://baike.baidu.com/item/Velocity" \t "https://baike.baidu.com/item/Struts%202/_blank)等技术，因此它不可能与这些视图层的模版技术进行整合。其次，Struts1与Servlet API的严重耦合，使应用难于测试。最后，Struts1代码严重依赖于Struts1 API，属于侵入性框架。
3. 从目前的技术层面上看，出现了许多与Struts1竞争的视图层框架，比如[JSF](https://baike.baidu.com/item/JSF/442519" \t "https://baike.baidu.com/item/Struts%202/_blank)、[Tapestry](https://baike.baidu.com/item/Tapestry" \t "https://baike.baidu.com/item/Struts%202/_blank)和[spring MVC](https://baike.baidu.com/item/spring MVC" \t "https://baike.baidu.com/item/Struts%202/_blank)等。这些框架由于出现的年代比较近，应用了最新的设计理念，同时也从Struts1中吸取了经验，克服了很多不足。这些框架的出现也促进了Struts的发展。
4. Struts1已经分化成了两个框架：第一个是在传统的Struts1的基础上，融合了另外的一个优秀的Web框架[WebWork](https://baike.baidu.com/item/WebWork" \t "https://baike.baidu.com/item/Struts%202/_blank)的Struts2。[Struts 2](https://baike.baidu.com/item/Struts 2" \t "https://baike.baidu.com/item/Struts%202/_blank)虽然是在Struts1的基础上发展起来的，但是实质上是以WebWork为核心的。Struts2为传统的Struts1注入了WebWork的先进的设计理念，统一了Struts1和WebWork两个框架。

Struts1分化出来的另外一个框架是[Shale](https://baike.baidu.com/item/Shale/2717198" \t "https://baike.baidu.com/item/Struts%202/_blank)。这个框架远远超出了Struts1原有的设计思想，与原有的Struts1关联很少，使用了全新的设计思想。Shale更像一个新的框架而不是Struts1的升级。

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.

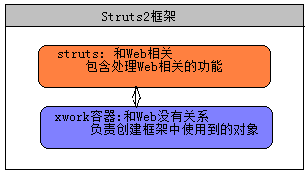
由传统Struts1和WebWork两个经典框架发展而来:

Struts2框架=Struts1+XWork

**Struts1和Struts2的区别:**

1. Stuts1有个核心控制器，但是只提供了一个接口，也就是execute，还要配置actionform之类的，很麻烦，所以依赖性比较强

2. Stuts2是针对拦截器开发的，也就是所谓的AOP思想，可以配置多个action，用起来比较方便，但是因为请求之前的拦截器有一些注入的操作，速度相对Stuts1来说稍微慢一点



**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

### 第二节 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-2.3.24.jar：Struts 2框架的核心类库

xwork-core-2.3.24.jar：Command模式框架,WebWork和Struts2都基于xwork

ognl-3.0.6.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-2.3.22.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-1.3.1.jar： 文件上传组件，2.1.6版本后需要加入此文件

commons-io-2.2.jar：传文件依赖的jar包

commons-lang3-3.2.jar：对java.lang包的增强

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

**前端控制器是框架的入口**

<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.**

**package** cn.wolfcode.\_01\_hello;

**public** **class** HelloAction {

**public** String execute() {

System.***out***.println("Hello Struts2!");

**return** "hello";

}

}

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

<struts>

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

<!-- 将当前action交给Struts2框架管理,

同时为其起一个共外界访问的名称:hello -->

<action name=*"hello"* class=*"cn.wolfcode.\_01\_hello.HelloAction"*>

<!--

请求处理完成之后需要执行跳转,在result中配置跳转的信息

跳转信息:type-跳转方式,location-跳转的页面

-->

<result name=*"hello"* type=*"dispatcher"*>

<param name=*"location"*>/views/hello/hello.jsp</param>

</result>

</action>

</package>

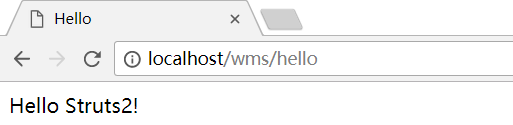
</struts>

**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/crm/hello[.action]

访问执行结果:



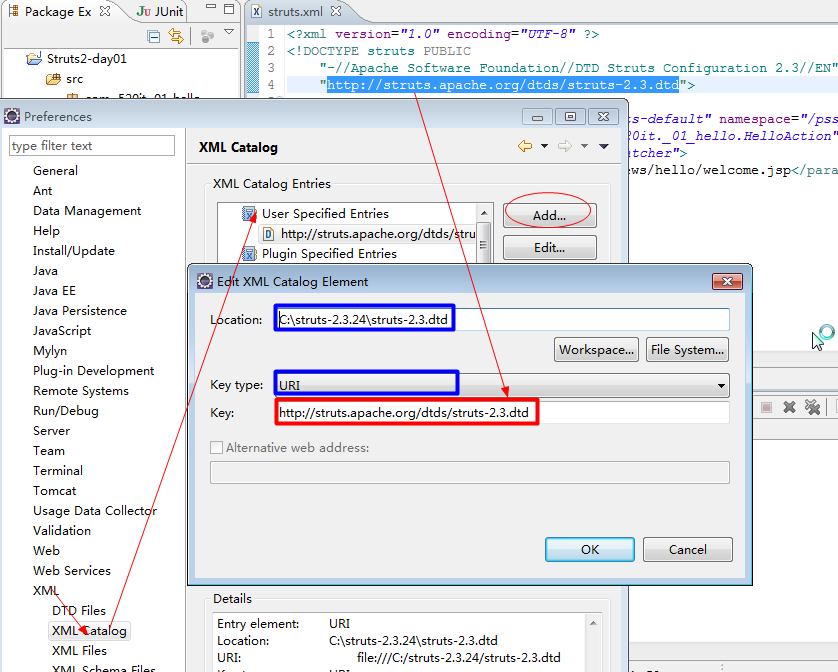
### 第三节 Eclipse支持struts.xml语法提示

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.

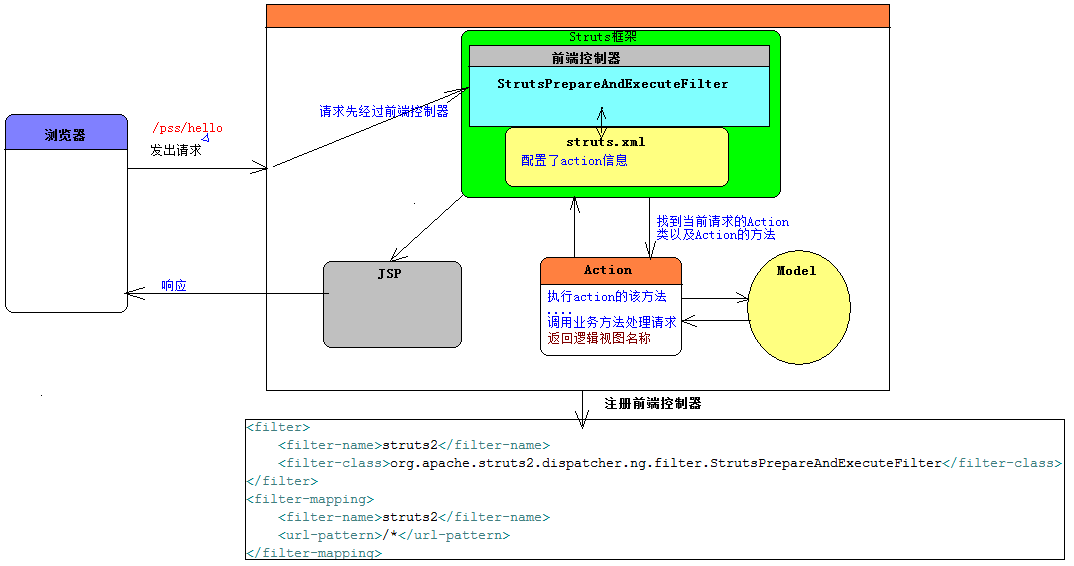
方式2:手动关联xml和dtd文件(推荐).

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.



### 第五节 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-2.3.24.jar 中 org.apache.struts2包里面:包含了Struts2的默认常量配置 req.setCharacterEncoding(“UTF-8”)

2.**struts-default.xml**  该文件保存在 struts2-core-2.3.24.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-2.3.24.jar.由插件提供

上述三个文件是框架自带的,我们不能直接修改

**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

注意:在Struts2中post请求参数的编码方式为UTF-8

<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"/>

### 第六节 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继承.

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.

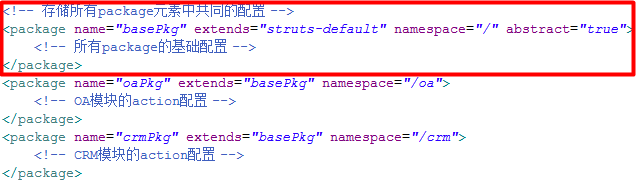
继承struts-default之后,就拥有了该<package>定义的所有资源.(结果返回类型,拦截器..)



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.

需要Struts2框架帮我们来管理action 的生命周期,

1. 对象的创建,使用反射的方式来创建action对象

需要使用到action类的全限定名

1. 帮我们调用对应的方法来处理请求

需要我们告诉框架需要调用哪一个方法来处理请求

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

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.

**当需要跳转到不同的package中的Action的时候,需要使用<param>元素来配置Action的路径**

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

### 第七节 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

### 第八节 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="cn.wolfcode.manymethod.EmployeeAction" method="{1}">

{1}对应表示当前action配置中的第1个\*

{2}对应表示当前action配置中的第2个\*

{n}对应表示当前action配置中的第n个\*

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="cn.wolfcode.manymethod.{1}Action" method="{2}">**

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.

### 第九节 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.如: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:**

**方式1:让Action类实现感知接口.**

ServletRequestAware：感知HttpServletRequest对象；

ServletResponseAware：感知HttpServletResponse对象；

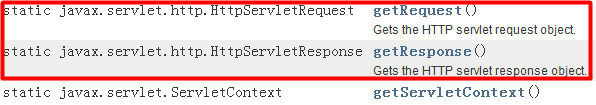
ServletSessionAware：感知HttpSession对象；

存在的问题，和ServletAPI耦合严重；

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)

**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,但是依然是Action与ServletAPI有耦合.

**方式3:通过ActionContext工具类**

**Struts2将作用域对象重新使用Map集合进行了封装,所以现在操作作用域中的共享数据就是直接操作对应的Map集合**

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.

此时要获取请求对象实际上就是在获取ActionContext对象(因为ActionContext是对请求数据的封装,相当于就是一个请求对象)

**可以使用ActionContext中的非静态方法来获取Servlet相关的API**

**所以需要获取到ActionContext的对象**

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();

4):操作servletContext(application)作用域:

原始:ServletContext ctx = request.getServletContext();

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

### 第十节 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)

**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对应Model对象中的一个属性.

而如果是下面这个修改密码的需求的话,我们通常是使用组合的方式来解决

在修改密码的时候,需要传入三个参数

oldpassword: 旧密码

newpassword: 新密码

confirmpassword: 确认密码

通常newpassword我们是封装在用户对象中,而其他两个单独接收

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.

数组类型的参数的接收:

\* List类型参数的封装

页面中的OGNL表达式的书写:

users[0].name = xxx

users[0].age = 19

users[1].name = ooo

users[1].age = 20

Action中的属性的定义:

private List<User> users = new ArrayList<>();

Getter/Setter方法

\* Map类型参数的封装

页面中的OGNL表达式的书写:

map [‘a’].name = “xxx”

map [‘a’].age = 19

map [‘b’].name = “ooo”

map [‘b’].age = 20

Action中的属性的定义:

private Map<String, User> map = new HashMap<>();

Getter/Setter方法

## 第三章 高级进阶

### 第一节 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 Spring AOP的一种实现．

AOP:面向切面编程.其实现原理:动态代理模式

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。

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

拦截器的作用

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

DRY原则：Dont't Repeat Yourself.

拦截器在设计和程序结构上的优点:

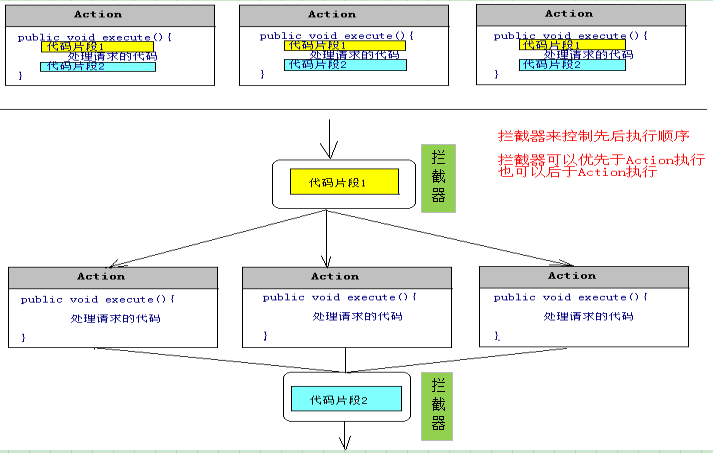
拦截器能把很多功能从Action中独立出来，分散到不同的拦截器里面，减少了Action的代码。如此，拦截器和Action本身的功能都更单一了。当通用的功能代码被封装在拦截器里面(代码模块化)，就可以对不同的Action，根据功能需要，来配置相应功能的拦截器了。提高了拦截器所实现的功能的重用性，也变相实现了装配式和可插拔式的体系结构，使得整个系统结构变得更灵活。

1.简化Action的实现

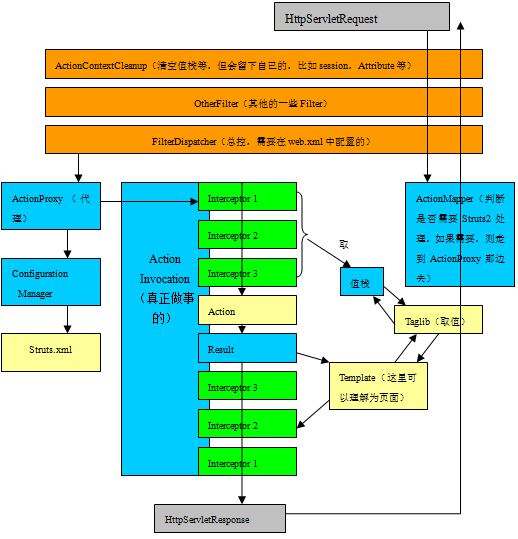
2.功能更单一

3.通用代码模块化

4.提高重用性



### 第二节 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(前端控制器)被调用，FilterDispatcher询问ActionMapper，来决定该请求是否需要调用某个Action。拦截器只对Action作拦截

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struts.xml，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result。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-ObjectFactory

### 第三节 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

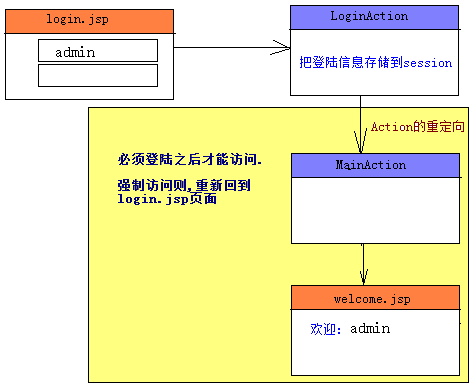
在校验的时候,如果有错误,将将错误信息放到一个List 集合中

此时workflow拦截器来判断List集合中是否有数据,如果有就返回一个逻辑试图名称为input

8.servletConfig

通过感知接口,获取感应对象

### 第四节 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.

**细节问题的处理:**

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>中修改默认拦截器的引用,那么该<package>中所有的<action>都拥有该功能.

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.

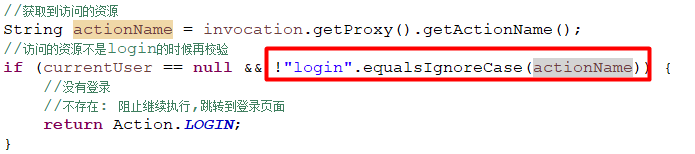
然后,我们修改了<package>的默认拦截器栈,因此就没有接收参数的拦截器了.

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

然后将默认拦截器的引用配置为当前的拦截器栈,特殊的action做特殊的处理

**给拦截器配置参数(解决硬编码)**

**在访问资源的时候,可能会有多个资源是不需要进行登录校验的,所以需要进行很多的判断**



使用上面的方式,我们是将代码写死,这些称之为硬编码(通常使用配置的方式来解决)

### 第五节 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中获取资源信息可以通过ActionSupport类中的:

String getText(String name);

参数: name:表示资源文件中的key

返回:拼接好的一个国际化信息.

语言切换方式:

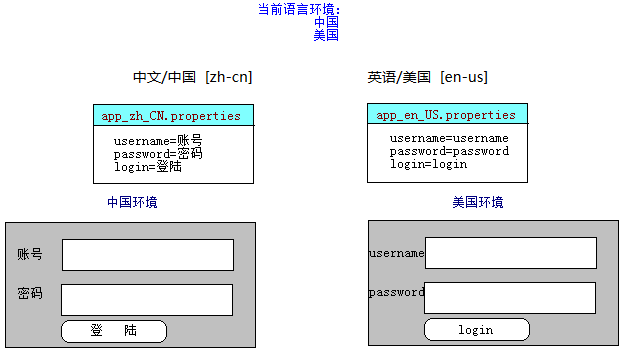
1. 自动切换:程序会根据当前的语言环境自动切换语言
2. 手动切换:可以通过发起请求到服务端执行以下代码进行切换:

//创建本地的语言环境

Locale locale = **new** Locale("zh","CN");

//将创建好的环境对象使用”WW\_TRANS\_I18N\_LOCALE”名称共享到session中

ServletActionContext.*getRequest*().getSession().setAttribute("WW\_TRANS\_I18N\_LOCALE", locale);



### 第六节 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；

在Action中如果有共享数据,将放到ValueStack值栈中,

然后到JSP页面中使用OGNL表达式来获取

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)**

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.****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).Map List

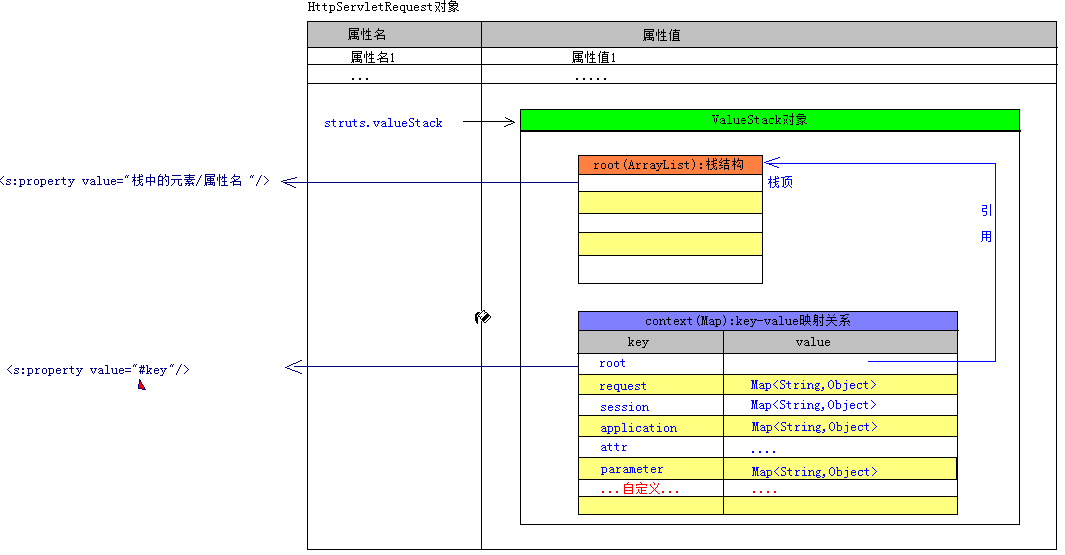
context存在用root对象的引用(只要拿到context对象就可以获取到root对象):

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"/>



**3,如何获取ValueStack.**

**Struts2框架接收到请求之后,立即创建一个新的ValueStack对象**

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);

所以,可以直接从request中获取到ValueStack对象

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

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

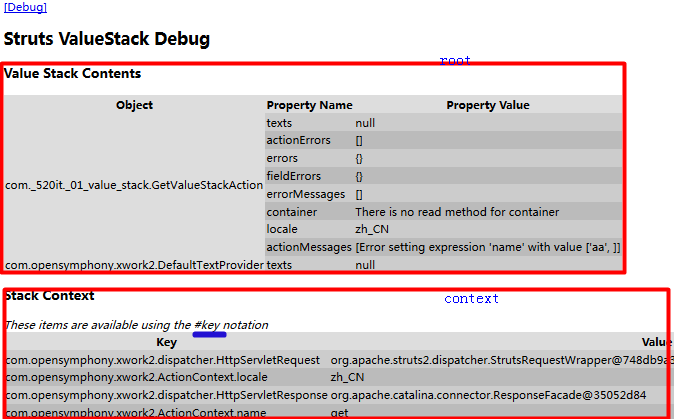
ValueStack vs = ServletActionContext.getRequest().getAttribute("struts.valueStack");

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

方式2: 通过ActionContext对象获取--相当于一个请求对象

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

以上两种方式获取的是同一个ValueStack对象.



**5.将数据存储到ValueStack中.**

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.在JSP页面中使用Struts2标签从ValueStack中获取数据:**

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"/>

取数据的时候,不加#表示先从Root中查找,找不到再到Context中查找

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

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.

### 第七节 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.

### 第八节 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.

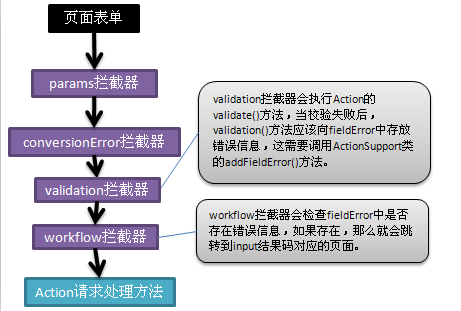
在validation拦截器中,标注需要排除的方法:

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.

执行数据校验,如果校验失败,将错误添加到Map集合中---validation拦截器

再执行判断Map集合中是否有错误信息,如果有,就跳转到input视图中--workflow拦截器



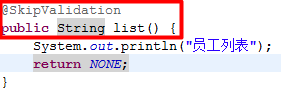


问题3:Action中某些方法不需要校验,却也执行了validate方法,如何避免问题.

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.

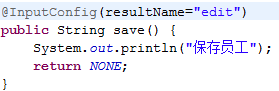
public void validateXxx():Xxx表示需要被校验的方法名称,那么该方法就只对xxx方法做校验.

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跳转到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.



### 第九节 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;

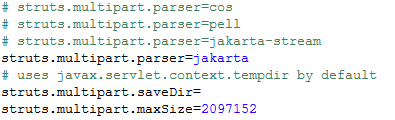
3):<input type="file" name=""/>;

Struts2底层使用的是Apache的上传组件实现文件的上传

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"/>

### 第十节 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)

在Action中定义一个inputStream方法处理下载文件的请求

指定当前方法的结果类型为:stream

可以通过修改inputName属性的值 修改处理下载请求的方法名称

设置结果类型的contentDisposition的值修改下载文件的名称

contentType:  指定被下载文件的文件类型;

inputName:  指定被下载文件的入口输入流;

contentDisposition：  指定下载的文件名;

bufferSize：  指定下载文件时的缓冲大小。

### 第十一节 Struts2标签

Struts2的标签库:

对于一个MVC框架而言，重点是实现两部分：业务逻辑控制器部分和视图页面部分。Struts2作为一个优秀的MVC框架，也把重点放在了这两部分上。控制器主要由Action来提供支持，而视图则是由大量的标签来提供支持。

  在JavaWeb中，Struts2标签库是一个比较完善，而且功能强大的标签库，它将所有标签都统一到一个标签库中，从而简化了标签的使用，它还提供主题和模板的支持，极大地简化了视图页面代码的编写，同时它还提供对ajax的支持，大大的丰富了视图的表现效果。与JSTL(JSP Standard Library，JSP 标准标签库)相比，Struts2标签库更加易用和强大。

PS：使用标签，需要引入Struts2核心jar包，在jsp头部，加入

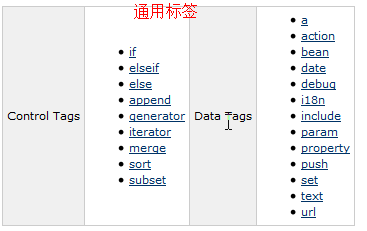
<%@ taglib uri="/struts-tags" prefix="s" %>

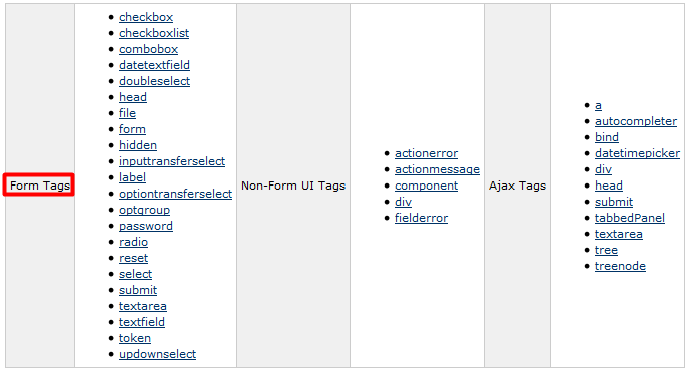
**Struts2标签:**

通用标签

控制标签/数据标签

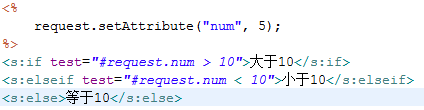
UI标签





UI标签

1:控制流程的标签:if-else if-else.



2:迭代标签:<s:iterator/>:

