1. Struts框架整体感知：
2. 客户端发送请求   
   2、请求先通过ActionContextCleanUp-->FilterDispatcher   
   3、FilterDispatcher通过ActionMapper来决定这个Request需要调用哪个Action   
   4、如果ActionMapper决定调用某个Action，FilterDispatcher把请求的处理交给ActionProxy，这儿已经转到它的Delegate--Dispatcher来执行   
   5、ActionProxy根据ActionMapping和ConfigurationManager找到需要调用的Action类   
   6、ActionProxy创建一个ActionInvocation的实例   
   7、ActionInvocation调用真正的Action，当然这涉及到相关拦截器的调用   
   8、Action执行完毕，ActionInvocation创建Result并返回，当然，如果要在返回之前做些什么，可以实现PreResultListener。添加PreResultListener可以在Interceptor中实现。
3. 链接器的执行过程
4. 链接器是配置在Struts2.xml文件中的，Struts2初始化的时候就会将其读入内存中
5. 每接收到一个请求，上第六步的时候，会创建一个filterchain的迭代调用链,依次执行每个拦截方法，因为所有的都是线程局部的，------》共享的

(2)Struts的常用类和接口

* 1. Action接口：为了规范action的编写制定的接口，主要逻辑放在execute()中，这个方法是默认执行的方法
  2. ActionSupport类：为了简化开发，部分实现了Action接口：API听丰富
  3. ActionContex类：可以访问底层的httpServletRequest等对象的模拟：参考API

HttpSession: Map session = ActionContext.getContext().getSession(); (map模拟的Session对象，操作方便)

* 1. ServletActionContext类：直接访问底层的ServeltRequest等对象

(1)取得HttpServletRequest对象: HttpServletRequest request = ServletActionContext. getRequest();

(2)取得HttpSession对象: HttpSession session = ServletActionContext. getRequest().getSession();

以上方法都不要再在Action的构造方法中进行get操作，因为这个时候有些值为null

* 1. cleanUp过滤器:用于在所有的操作完成之后进行ActionContext的清理工作，(换句话来说，Struts的过滤器本身就是自带顺序的)

在Struts中大量使用了线程局部变量，以保证线程的安全

1. Struts.xml文件的简单配置

(2-1)

<package name=”eportal” extends= “Struts-default”>

<action name=”login” class=”xxxx.xxx..xx” method=”xxxxx”>---🡪现实在一个acton中多个方法

<result name=”xxxxx” type=”xxx”>/xxxx.xxxx</result>

</action>

</package>

通配符的做法

<package name=”eportal” extends= “Struts-default” namespace=“xxxx”>-----🡪name是个区分的，不能有两个相同名字的package namespace属性是用来划分url的命名空间的

/ /sys 等这些是不同的

<action name=”\*” class=”xxxx.xxx..xx” method=”{1}”>---🡪现实在一个acton中多个方法

<result type=”xxx”>/{1}.xxxx</result>------------🡪如果没有name，则默认为success

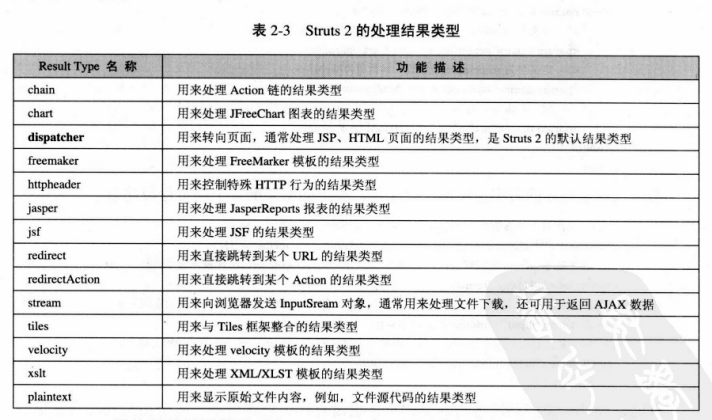
<result name=”xxxxx” type=”xxx”>/xxxx.xxxx</result>

</action>

</package>

对于上面的\* 还可以是\*.\*.\* 相对的 在action找哦你则为{1}{2}{3}

(2-2)在result中的type属性



1. action详解

(3-1)全局result

<global>

<result name=”xxxx”>/xxx.jsp</result>

</global >

局部的result在actioon中配置

(3-2)

1. OGNL详解 -------- >>>一种解析型语言 有一个解析器的

ActionContext是OGNL的上下文 ValueStack是其中的根元素 访问方法${name}

Action中的值是保存在ValueStack中的，所以访问方法同上

在上下文中，有些之是不再ValueStack中的，所以，通过#来访问这部分对象的值

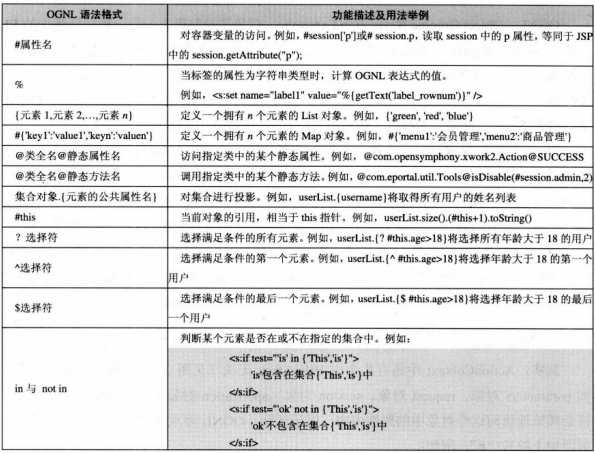
注：$访问ValueStack #访问上下文

例：${loginname}

#request[‘xxx’]=request.xxx=session.getAttribute(“xxx”)

使用OGNL表达式可以调用方法%{方法名}

OGNL的基本语法



1. Struts的常量配置说明

<!-- 指定请求url与action映射器，默认为org.apache.struts2.dispatcher.mapper.DefaultActionMapper

-->

<constant name="struts.mapper.class"

value="org.apache.struts2.dispatcher.mapper.DefaultActionMapper" />

<!-- 是否允许使用表达式语法，默认为true. -->

<constant name="struts.tag.altSyntax" value="true" />

<!-- 模板目录 -->

<cosntant name="struts.ui.templateDir" value="template" />

<!-- 设置模板类型. 可以为 ftl, vm, or jsp -->

<cosntant name="struts.ui.templateSuffix" value="ftl" />

<!-- 定位velocity.properties 文件. 默认velocity.properties -->

<cosntant name="struts.velocity.configfile" value="velocity.properties" />

<!-- 设置velocity的context. -->

<cosntant name="struts.velocity.contexts" value="...." />

<!-- 定位toolbox -->

<cosntant name="struts.velocity.toolboxlocation" value="...." />

<!-- 指定web应用的端口 -->

<cosntant name="struts.url.http.port" value="80" />

<!--

该属性指定Struts2应用加载用户自定义的属性文件，该自定义属性文件指定的属性不会覆盖

struts.properties文件中指定的属性。如果需要加载多个自定义属性文件，多个自定义属性文

件的文件名以英文逗号（,）隔开。(也就是说不要改写struts.properties!)

-->

<constant name="struts.custom.properties"

value="application,org/apache/struts2/extension/custom" />

<!-- 指定加密端口 -->

<cosntant name="struts.url.https.port" value="443" />

<!-- 设置生成url时，是否包含参数.值可以为: none,get or all -->

<cosntant name="struts.url.includeParams" value="get" />

<!-- 设置要加载的国际化资源文件，以逗号分隔. -->

<cosntant name="struts.custom.i18n.resources" value="application" />

<!--

对于一些web应用服务器不能处理HttpServletRequest.getParameterMap(), 像

WebLogic,Orion, and OC4J等，须设置成true,默认为false.

-->

<cosntant name="struts.dispatcher.parametersWorkaround" value="false" />

<!-- 指定freemarker管理器 -->

<cosntant name="struts.freemarker.manager.classname"

value="org.apache.struts2.views.freemarker.FreemarkerManager" />

<!-- 设置是否对freemarker的模板设置缓存,效果相当于把template拷贝到 WEB\_APP/templates. -->

<cosntant name="struts.freemarker.templatesCache" value="false" />

<!-- 通常不需要修改此属性. -->

<cosntant name="struts.freemarker.wrapper.altMap" value="true" />

<!-- 指定xslt result是否使用样式表缓存.开发阶段设为true,发布阶段设为false. -->

<cosntant name="struts.xslt.nocache" value="false" />

<!-- 设置struts自动加载的文件列表. -->

<cosntant name="struts.configuration.files"

value="struts-default.xml,struts-plugin.xml,struts.xml" />

<!-- 设定是否一直在最后一个slash之前的任何位置选定namespace. -->

<cosntant name="struts.mapper.alwaysSelectFullNamespace"

value="false" />

<!-- spring 托管 与spring集成时需要用到 -->

<constant name="struts.objectFactory" value="spring" />

<!-- 指定加载struts2配置文件管理器，默认为org.apache.struts2.config.DefaultConfiguration ,开发者可以自定义配置文件管理器，该类要实现Configuration接口，可以自动加载struts2配置文件。

-->

<constant name="struts.configuration"

value="org.apache.struts2.config.DefaultConfiguration" />

<!-- 设置默认的locale和字符编码 国际化使用 -->

<constant name="struts.locale" value="zh\_CN" />

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

<!-- 指定Struts的工厂类 -->

<constant name="struts.objectFactory" value="spring"></constant>

<!--

指定spring框架的装配模式，装配方式有: name, type, auto, and constructor (name

是默认装配模式)

-->

<constant name="struts.objectFactory.spring.autoWire" value="name" />

<!-- 该属性指定整合spring时，是否对bean进行缓存，值为true or false,默认为true -->

<cosntant name="struts.objectFactory.spring.useClassCache" />

<!-- 指定类型检查,包含tiger和notiger -->

<cosntant name="struts.objectTypeDeterminer" value="tiger" />

1. Struts2异常处理

全局的异常处理：

<global-execption-mapping>

<execption-mapping execption=”java.lang.Execption” result=”execption” > execption是个全局的跳转

在xml文件的配置的异常处理，要与在Java类中的异常声明相一致

</gloable-execption-mapping>

1. 在url中制定要调用的方法：

<http://localhost:1000/struts2actionMethod/user/user.action!add>

在url中带参数

<http://localhost:1000/struts2actionMethod/user/user.action!add>?repositoryid=0&flag=1