Structs2作为框架对很多功能(类型转换，验证，异常处理，国际化等)进行了封装(框架的作用)

MVC：

MVC设计模式是基于面向对象程序设计，C层从M层中获取数据，V层展示

2，调用Model方法

JavaBean

1，请求

Servlet

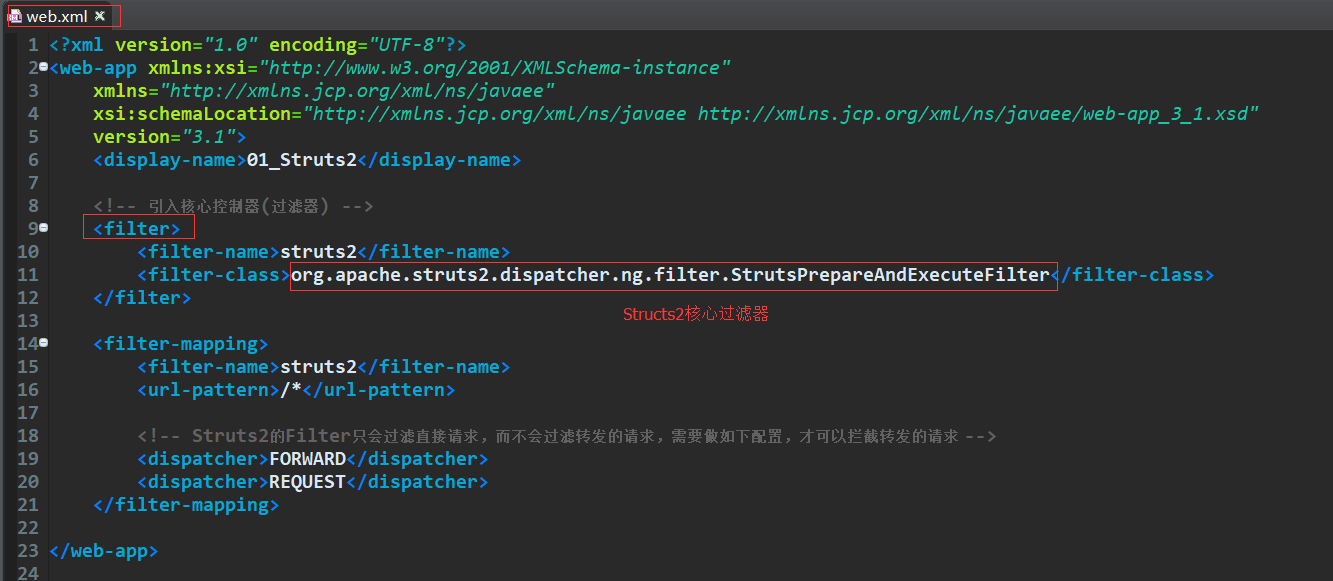
浏览器

4，响应

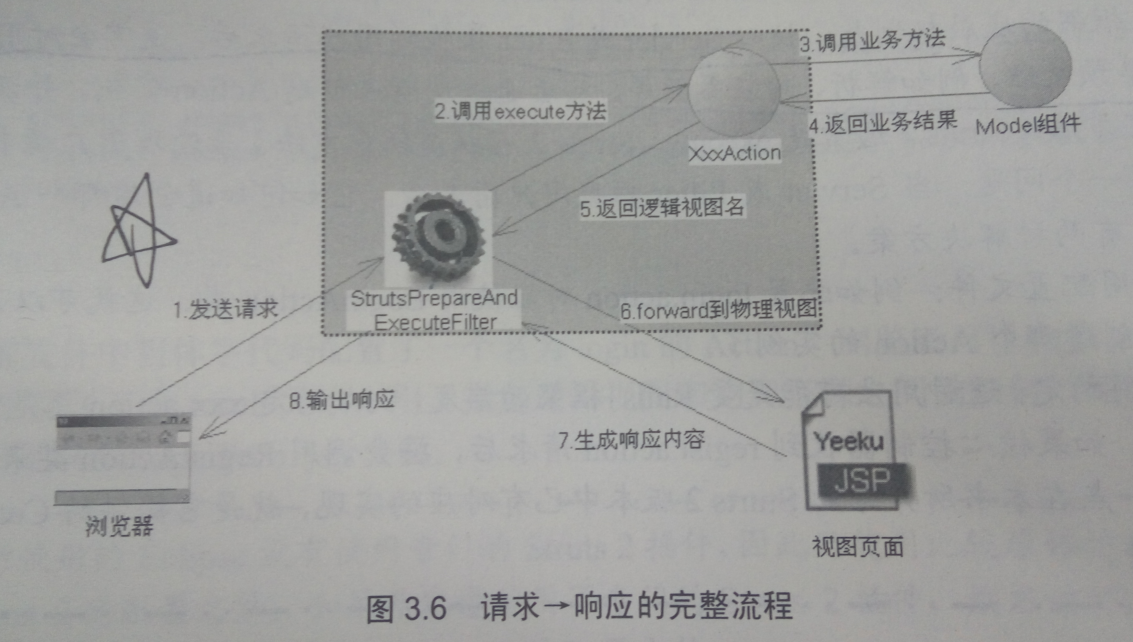
3，forward请求

JSP

Structs2是基于Filter，请求提交后，创建HttpServletRequest/HttpServletResponse，经过listener，经过Filter，此时会经过Structs2的核心过滤器StructsPrepareAndExecuteFilter，然后请求被Structs2处理。(这样Structs2框架就介入WEB应用中，框架只是简化了开发，有模板，框架任然是基于Servlet实现的，在Servlet中，核心的是Filter，Servlet，而Structs2就是基于Filter的，SpringMVC基于Servlet)



框架的核心Servlet/Filter接收到请求后，会对用户的请求进行简单的预处理(解析，封装参数)



StructsPrepareAndExecuteFilter(核心控制器)和Action(业务控制器)公共构成C层，Action执行完方法后，返回一个字符串(逻辑视图)，核心控制器再请求forward物理视图(jsp)

Eclipse在生成，部署WEB项目的时候，会自动将src目录的资源复制到WEB-INF/classes路径下(classpath)，Structs2基本上将所有的jsp页面都放在WEB-INF下，通过请求的方式访问，而不是直接访问。

Structs2会自动加载struct.xml(开发者自定义)，structs-default.xml(Struct2自带的配置文件)

structs-plugin.xml(插件的配置文件)配置文件

Structs2加载常量的顺序：

Structs-default.xml：structs2-core.jar包中的

Structs-plugin.xml：Structs-Xxx.jar插件包中的(struts2-json-plugin-2.3.31.jar)

Structs.xml：用户自定义

Structs.properties：Structs2的默认配置文件(常量配置)

Web.xml：web应用的配置文件

Structs2的Action不需要继承任何Structs2的接口，类就可以被Structs2管理(在struts.xml中作了相关配置)，但是一般都会继承ActionSupport类(模板，固定的套路，更好的被Struts2管理)

Struts2获取4个域对象的4种方式：

① 获取Map类型

Static ActionContext getContext()：

获取ActionContext() (**ActionContext**.*getContext*())

Map getApplication()：

类似获取Servlet的ServletContext(**ActionContext**.*getContext*().getApplication())

Map getParameters()：获取请求参数，类似HttpServletRequest的getParameterMap()

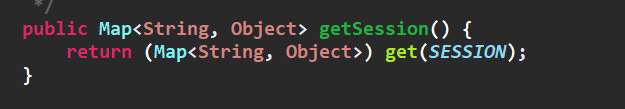
Map getSession()：HttpSession

获取原理：

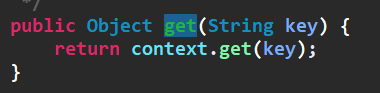


在创建ActionContext之前，会为ActionContext创建一个参数context(类型为OgnlContext，而OgnlContext又继承至Map，所以context是一个Map类型)，在创建context的时候，会创建4个Map类型的域，并且将这4个Map和request，response放入context(Map类型)中，在创建ActionContext的时候，就会将此context传入到ActionContext中

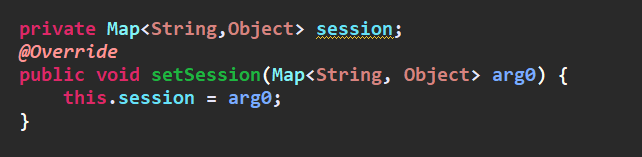
在获取4个域的时候，通过ActionContext的get方法获取，然后get方法会到context中获取Map

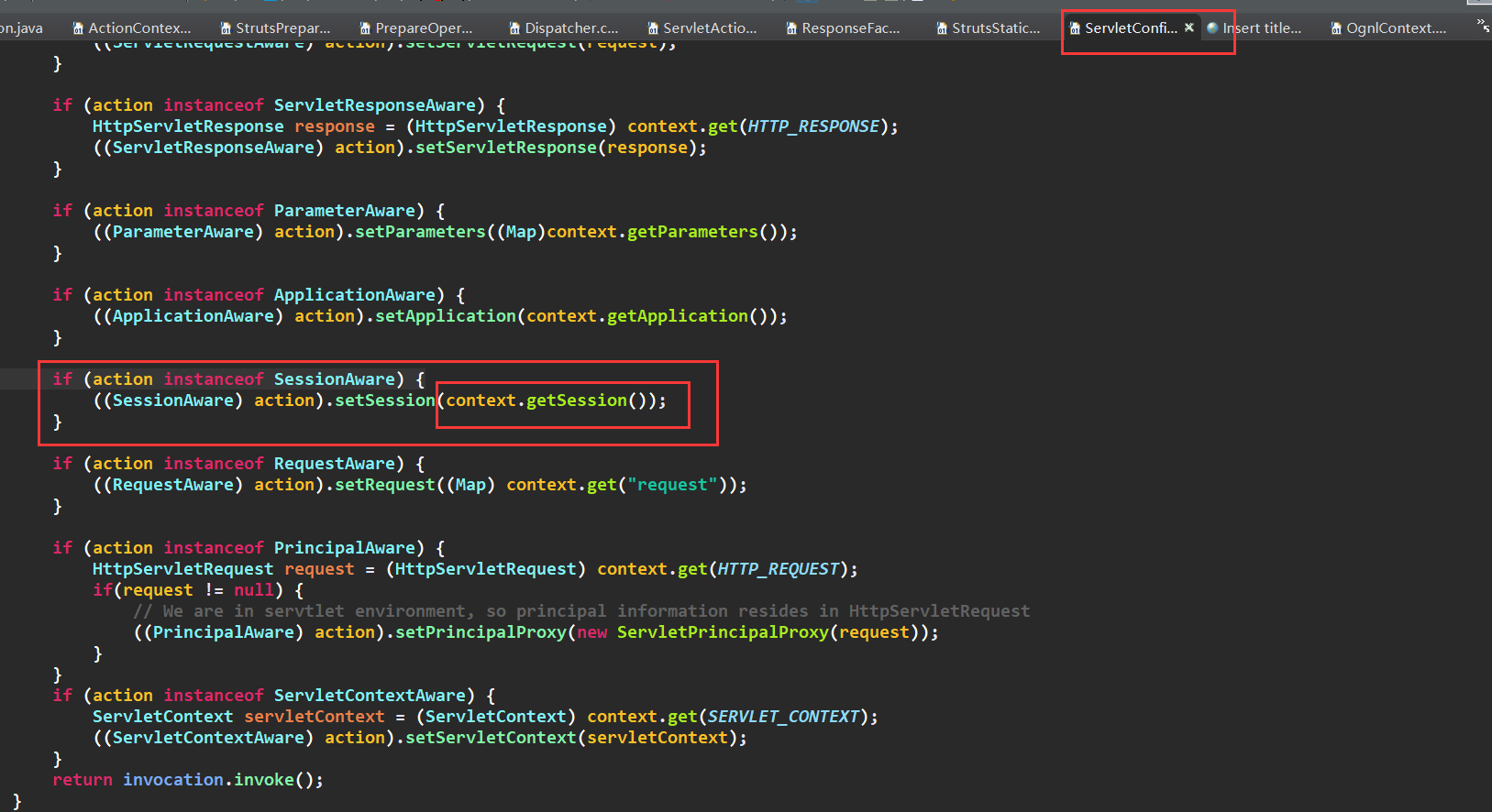


Session：com.opensymphony.xwork2.ActionContext.session



② ApplicationAware,SessionAware, RequestAware, ParameterAware接口用户获取Struts2中的域对象

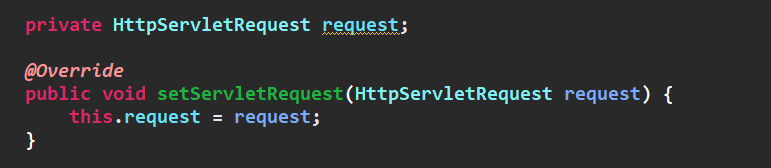




当执行到ServletConfigInterceptor拦截器时，会去判断Action是否实现接口，如果实现会将获取到的Map传入Action中

③ Action实现ServletContextAware/ServletRequestAware/ServletResponseAware接口(不建议)

实现原理和②一样



④ ServletActionContext访问(不建议，和原生Servlet API耦合)

ServletActionContext.getRequest();

Static PageContext getPageContext()

Static HttpServletRequest getRequest()

Static HttpServletResponse getResponse()

Static ServletContext getServletContext()

获取原理：

ServletActionContext继承与ActionContext

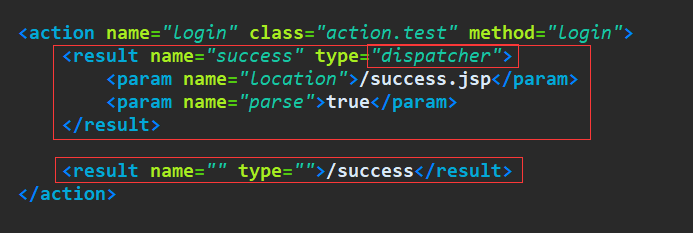
实现原理和①类似，因为在context中保存了原生的request和response，所以我们可以获取到任意的原生的Servlet，通过键到context中查找()



HTTP\_REQUEST：com.opensymphony.xwork2.dispatcher.HttpServletRequest

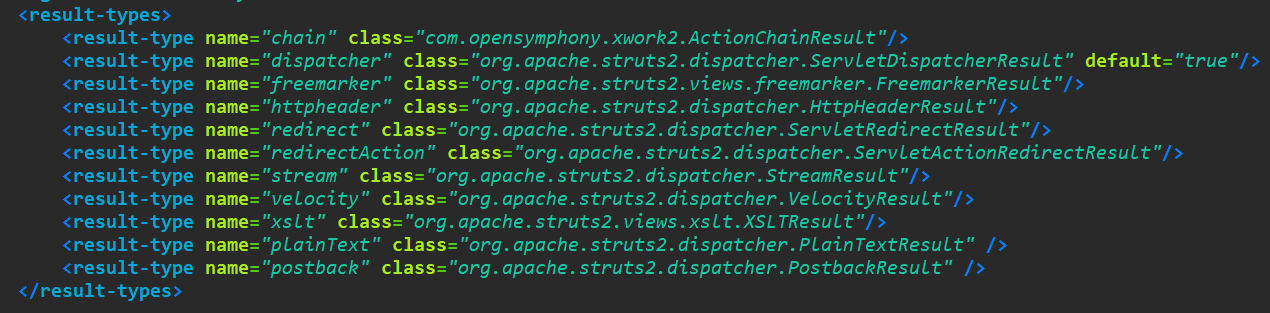
**动态访问Action的方法：**在浏览器中可以输入<http://127.0.0.1:8080/test/testAction!login>会自动访问testAction下的login方法，但是这种方式不安全，Struts2默认禁用

Action方法执行完后，返回一个逻辑视图，在跳转(forward)到物理视图时可以在指定方式：



在跳转到物理视图时，可以通过设置type来设置跳转方式(默认dispatcher)，location表示跳转的地址，parse表示是否支持OGNL表达式，默认为true，所以一般使用下面那种方式

Struts2支持的跳转方式：



Chain：将请求**转发**给另一个Action处理

Dispatcher：转发(默认)

Freemaker：

Httpheader：

Redirect：重定向到JSP，Struts2将调用HttpServletResponse的sendRedirect(String)重定向(底层)

RedirectAction：重定向到Action，使用ActionMapperFactory的ActionMapper重定向

Stream：

Velocity：

Xslt：

Plaintext：将JSP的源代码显示给用户

Postback：



PreResultListener：

在Struts2中有一个PreResultListener监听器，在Action执行完执行方法后，在跳转到物理视图之前调用，做一些操作(日志)

Struts2的异常处理：

一般有异常时可以通过try catch来捕获，在Action中出错时，是通过throws将异常信息往上层抛，直接抛给Struts2框架处理，这样我们就不用处理异常，Struts2的conversionError拦截器会自动处理异常

throws是表明方法抛出异常，需要调用者来处理，如果不想处理就一直向外抛，最后会有jvm来处理；

try catch 是自己来捕获别人抛出的异常，然后在catch里面去处理；

可以在Struts.xml的配置文件声明当出错时，跳转到指定的页面(必须指定错误的类型，可以在Struts.xml中声明局部/全局的错误映射类型)



Struts2标签：

ognl的很多标签在处理数据的时候是先将数据压入ValueStack的栈顶，当标签执行完毕后，再将数据移出。

Subset标签用于根据指定方式取得集合的子集，这种方式可以自定义(实现SubsetIteratorFilter.Decider)，很多标签都可以自定义截取方式

Token标签用于防重复提交

**主题：**

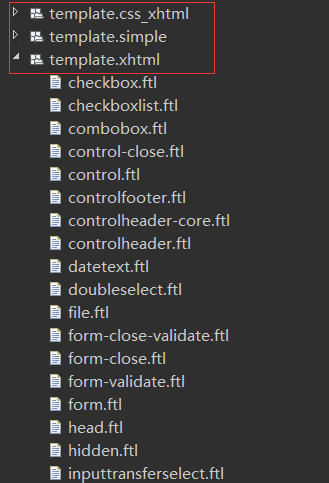
Struts2标签在被解析为HTML语法时，会根据自己的主题解析

Simple：只会将Struts标签转化普通HTML元素

Xhtml：对simple主题进行扩展，添加一些CSS样式

Css\_Xhtml：对Xhtml主题进行扩展，添加更多的CSS样式

每个主题对应着一个转换的模板(struts2-core-2.3.31.jar下)



也可以通过修改这些模板的标签来修改转化后的样式

每次请求时都会重新实例化Action，毕竟要压入值栈，所以Action是线程安全的，一个线程有一个唯一的ActionContext

异常信息会被Struct2放到值栈的属性栈的栈顶中

在Struts.xml中可以使用OGNL表达式

<**result** name=*""* type=*"redirect"*>/login?name=${username}</**result**>

OGNL标签和EL标签作用完全类似，OGNL还能对数组，对象，集合进行处理，EL表达式作用单一

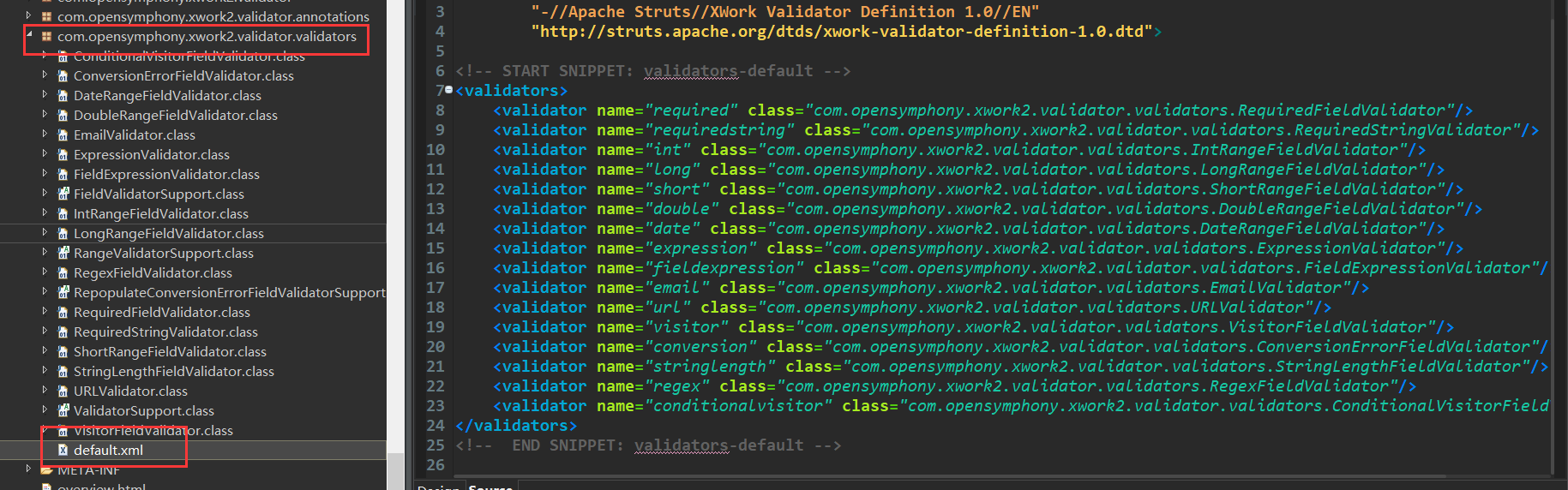
在每次执行Action之前都会创建新的ActionContext，ActionContext是线程安全的，也就是说在同一个线程里ActionContext里的属性是唯一的

Struts2如何保持session中的数据不变的：

每次请求都会创建新的ActionContext，在创建ActionContext的时候也会创建request，session，param，application的Map，在创建session的Map时，是通过新的HttpServletRequest在web容器中去寻找HttpSession的，这样保证了session中的数据不丢失。Struts2只是一种方便获取，管理属性的工具，而底层任然采用的是Servlet

Struts2在返回数据为JSON的时候，会将整个Action的对象转化为Json字符串，并且可以在struts.xml中指定要返回的属性和不返回的属性

Struts2验证框架：



ValueStack，CompoundRoot，ActionContext：

CompoundRoot中保存的是Struts2中的对象集合，ActionContext中保存的是域集合(Map类型的Request，Session，Application，Params)，ValueStack包含CompoundRoot(ArrayList)和ActionContext，而ActionContext又可以获取ValueStack，两者可以相互获取

可以通过getText()获取到国际化中的资源文件

ValueStack、OgnlValueStack、ActionContext、CompoundRoot

ValueStack是一个接口，OgnlValueStack是其实现类，ActionContext中的context是ognlContext(Map类)，

CompoundRoot保存的是对象，是ArrayList类，ValueStack中即包含ActionContext，又包含CompoundRoot，而ActionContext的context(Map)中又包含ValueStack

Ognl：

获取值栈(CompoundRoot)中属性：obj.propertyName/[0].propertyName

获取ActionContext中属性：#session.sessionName

如果未指明是在值栈还是属性栈中(<**s:property** value="username"/>)：

先从request域中查找，然后再到值栈中查找，然后在到ActionContext的context中查找(注意，这里context本身就是Map，它还保存了属性栈的Map，但是在查找的时候，只在context的Map中查找，而不会到属性栈中查找)，这也是包装后的StrutsRequestWrapper的get方法查找顺序

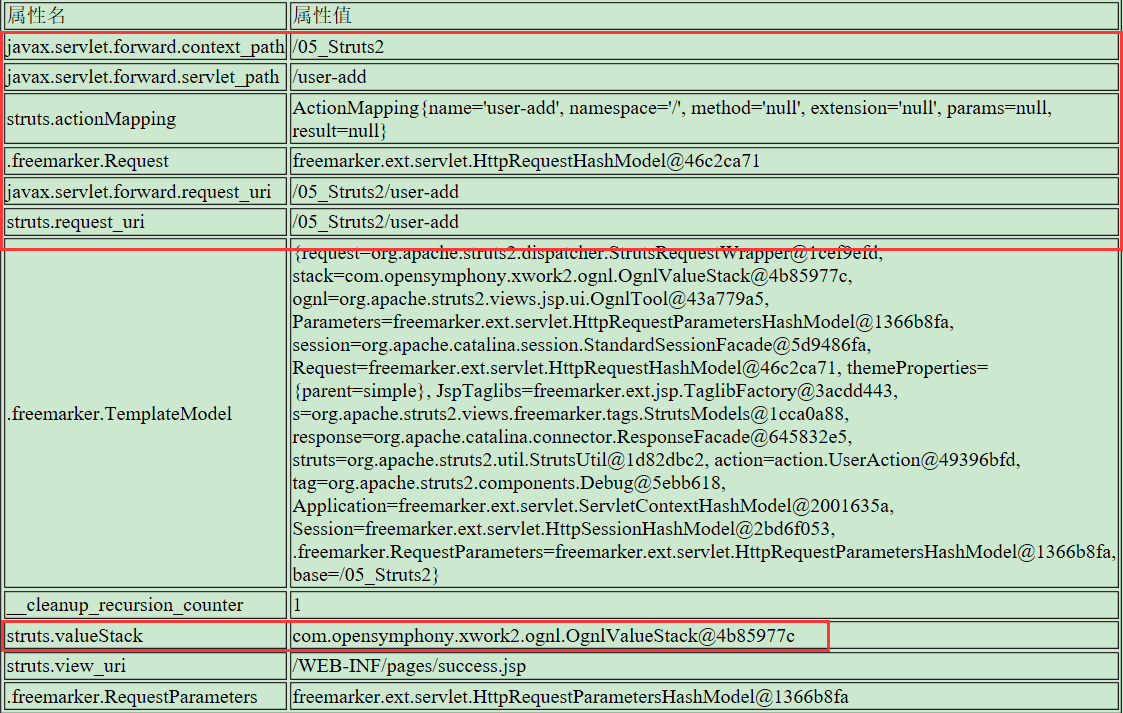
为啥${username }能获取到属性栈中值？ 注意，这里不是获取属性栈中的值，EL本身就是直接获取底层Servlet中的属性，而不是到Struts2的属性栈中查找，EL获取顺序：pageScope、requestScope、sessionScope、applicationScop(从小到大)

框架是脱离WEB容器的

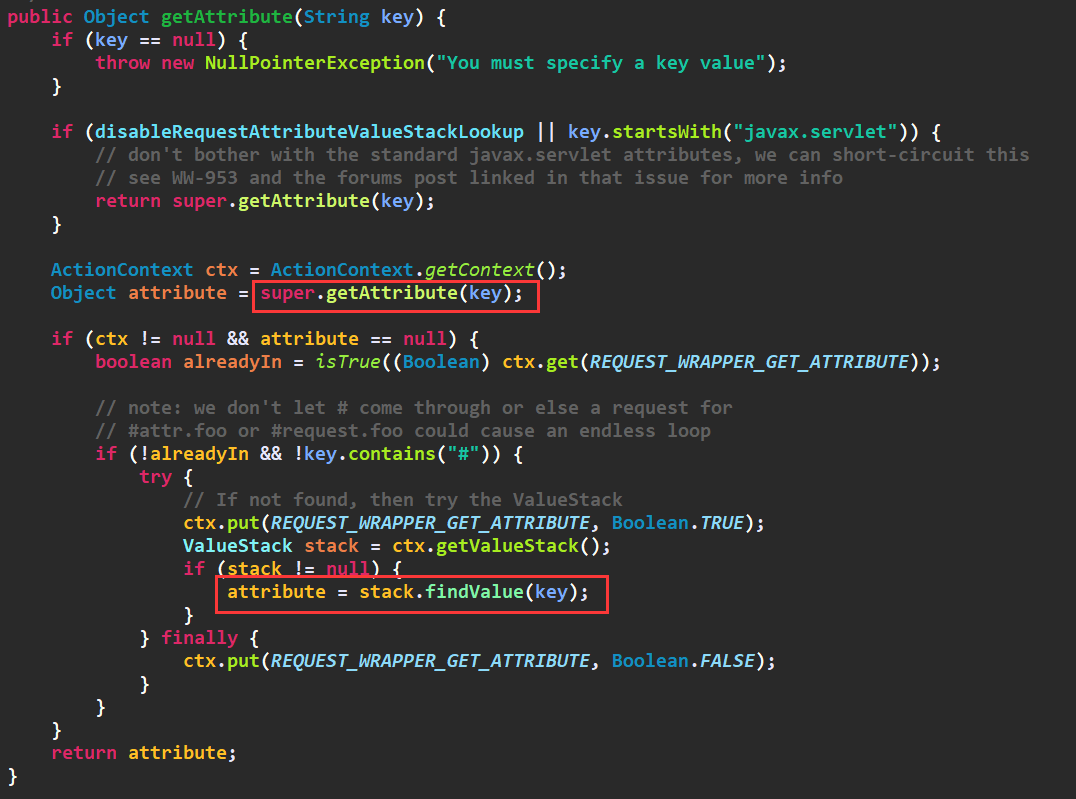
在Struts2中使用request的相关实际上都是操作org.apache.struts2.dispatcher.StrutsRequestWrapper，因为在Struts2的核心拦截器的doFilter方法中将HttpServletRequest进行了包装

Servlet中使用的HttpSession，HttpServletRequest等都是接口(多态)，具体的实现方法都是其子类，比如HttpSession实际上是org.apache.catalina.session.StandardSessionFacade

Struts2中，请求在经过拦截器时，会向request(以被包装)中添加其他属性，在Struts2的很多地方都会用到这些属性



Struts2中在使用标签获取属性时：如果未指定获取属性的域，Struts2首先会去request域中查询，如果找到了，就返回，如果没找到，Struts2会到OgnlValueStack的对象栈中找，如果找到返回，没找到而继续到Struts2的ActionContext的context中查找，但不会到属性栈中查找



被包装后的request(StrutsRequestWrapper)的getAttribute方法，首先会到request域中查找(super.getAttribute)，如果没找到，则到值栈(ValueStack)中查找，先从值栈的root(对象栈)中查找，然后再到ActionContext中的context查找

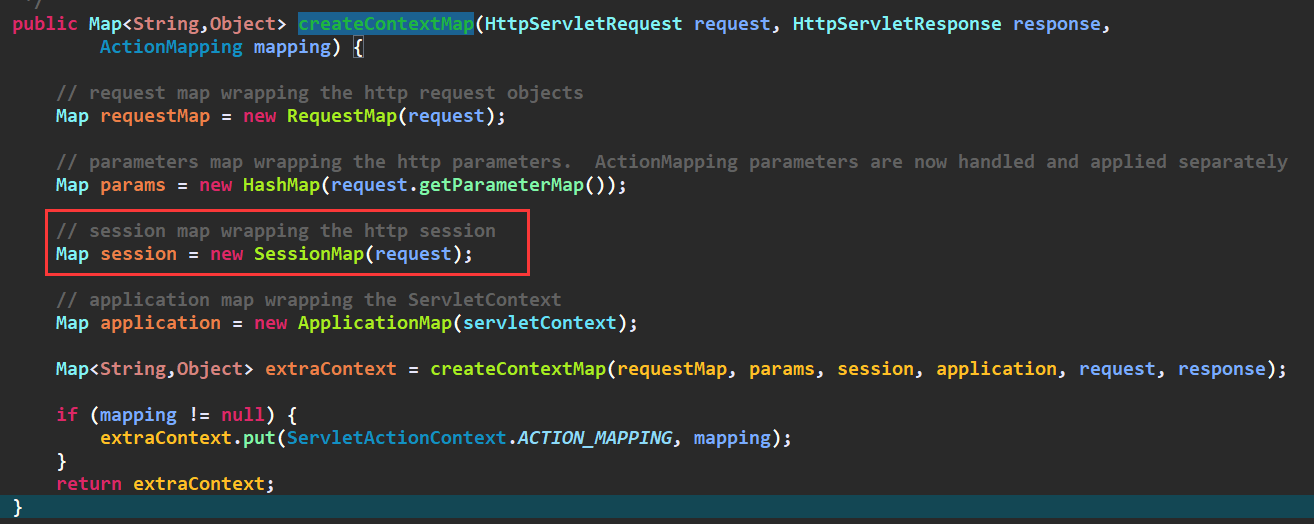
为什么Struts2和Servlet之间可以相互获取属性(Struts2是Map类型)?

在Struts2的核心FilterStrutsPrepareAndExecuteFilter的doFilter方法中，有一个创建ActionContext的操作



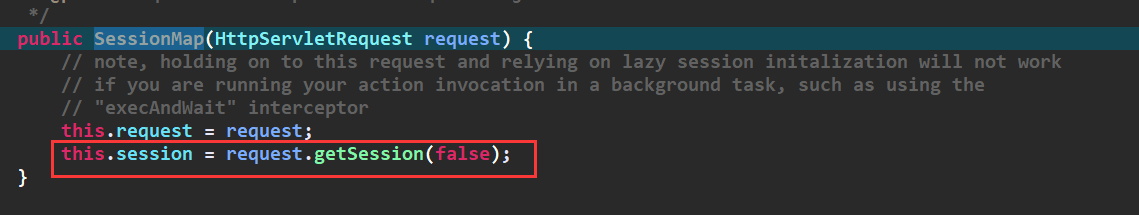


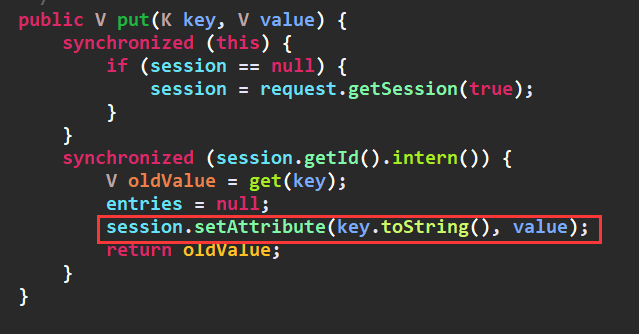
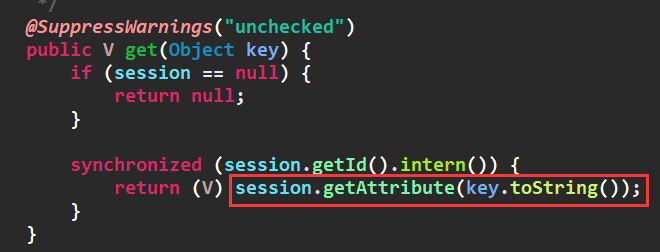
在创建ActionContext之前先要创建一个Map集合作为ActionContext的参数，这个Map集合是关键，它封装了Struts2的Request，Session，Param，Application共4个域的Map





分别将Request，session，ServletContext封装为其对应的Map，这里的SessionMap是Map的子类，将4个子类封装好后，调用createContextMap将4个Map封装为Map作为ActionContext的参数

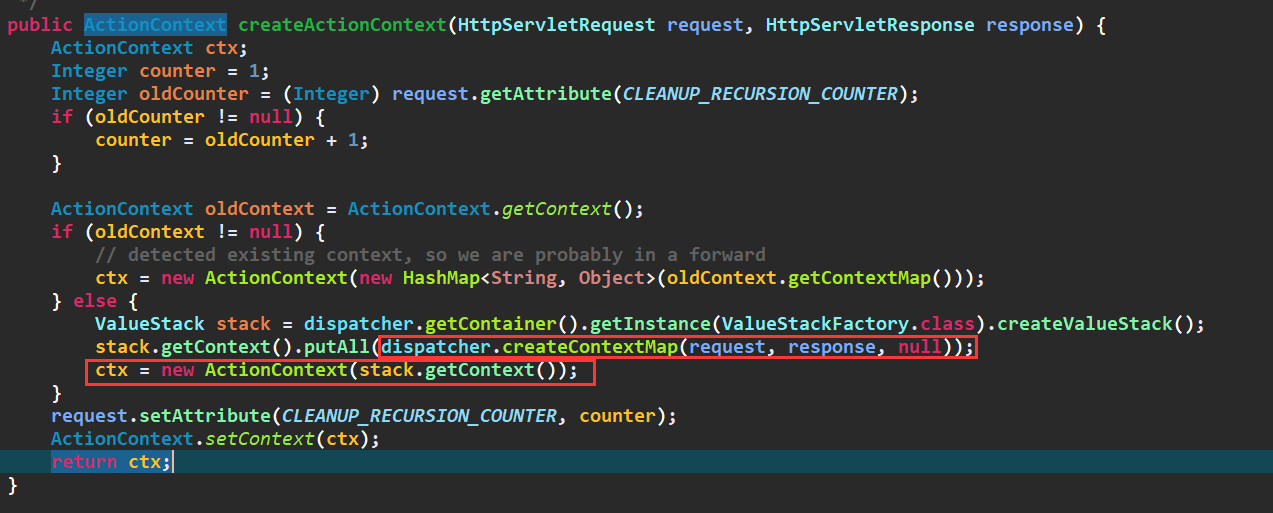




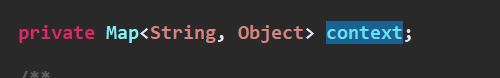
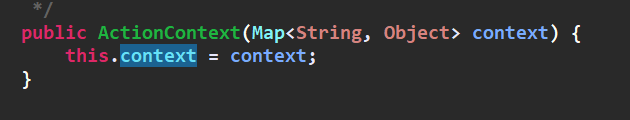
这两个是SessionMap的get和put方法，在操作Map的session的get/put时，实际上是操作的SessionMap

的get/put，而在SessionMap有实际上操作的是session(HttpSession)的方法

我们在操作Map时，底层实际常操作的任然是其对应的Servlet的API



在创建好ActionContext的参数后(Map)，实例化ActionContext



这里的context任然是Map

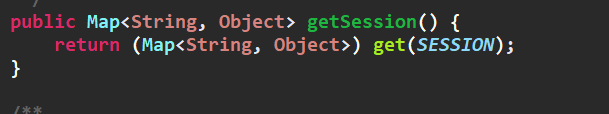
在ActionContext内存在一个Map类型的context，context的元素是4个Map+request和response集合

在后续的操作中，任然可以向context中添加属性

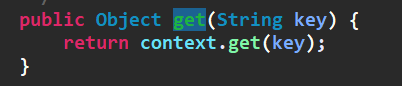
获取域对象时：



获取到ActionContext后，调用getSession方法



这里的SESSION作为常量，作为在context中获取SessionMap的键



通过ActionContext.getContext().getSession()获取到的实际上是SessionMap，父类Map指向SessionMap，

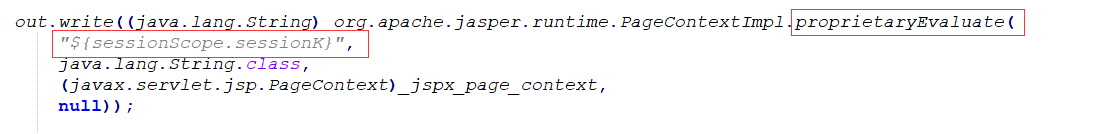
在Map操作的实际上是对SessionMap操作，而操作SessionMap实际就是操作HttpSession。

这也说明了为啥在Struts2中可以使用EL表达式：

在JSP使用EL表达式，将JSP转化为Servlet时，EL表达式任然会被转化

EL：${sessionScope.sessionK}

Servlet：



在通过EL表达式获取属性时，底层是先通过sessionScope获取到Session域中的所有属性值，封装为一个

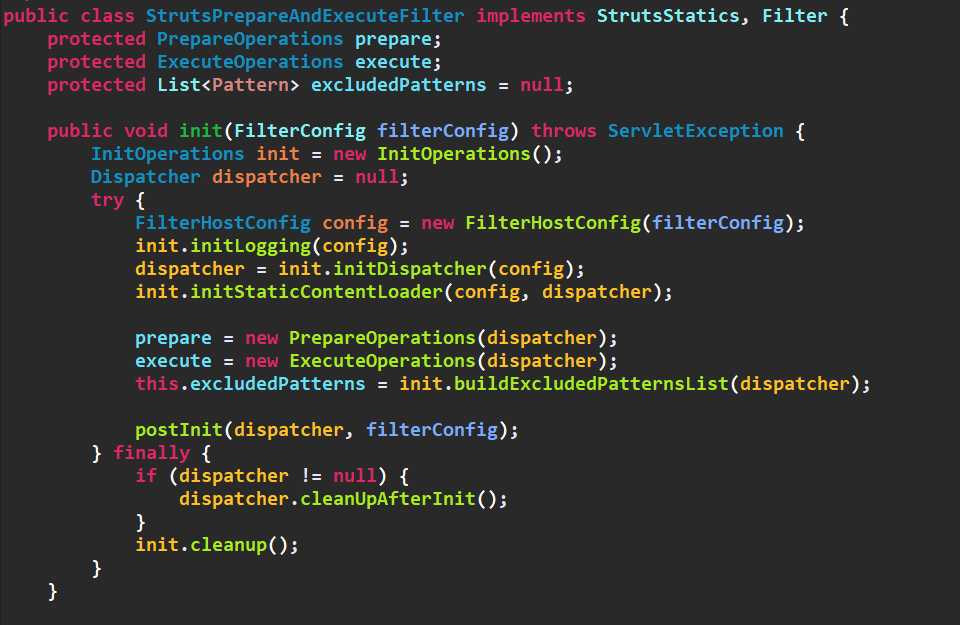
集合，然后根据sessionK到集合中查询找到指定属性(可以通过org.apache.jasper.runtime.PageContextImpl类

proprietaryEvaluate方法查看运行过程)

**Struts2运行流程：**

**StrutsPrepareAndExecuteFilter核心拦截器**

**Init：**

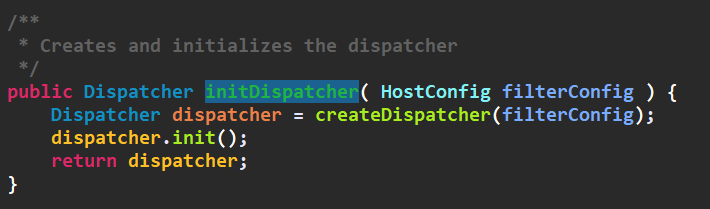


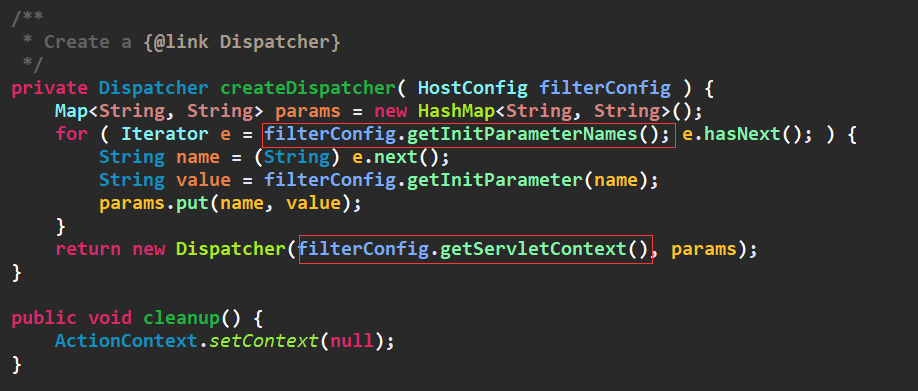
**FilterHostConfig：对FilterConfig进行封装**



init.initLogging(config)：初始化Struts2的日志

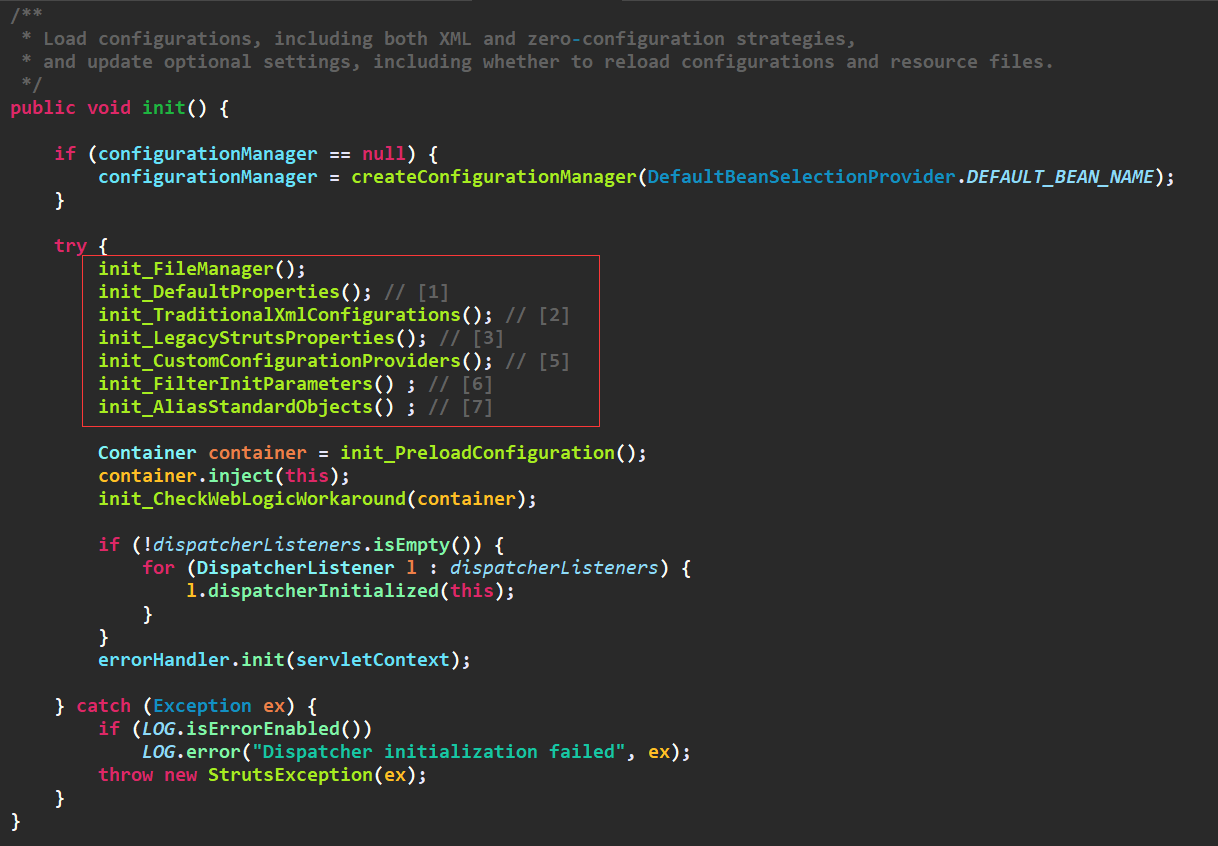
initDispatcher：创建dispatcher





将filter的初始化参数转化为Map，并将ServletContext放在Dispatcher中

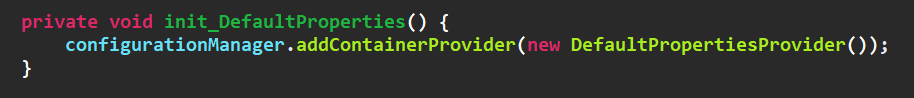
dispatcher的初始化



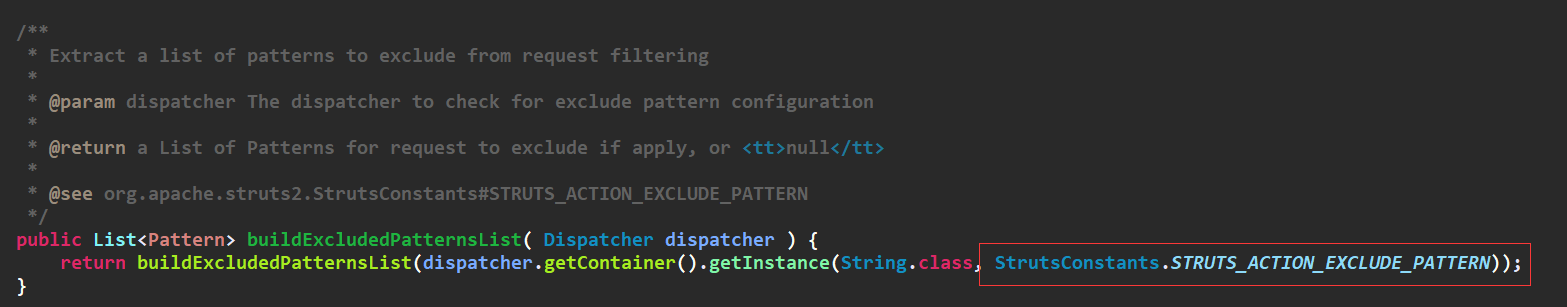
对Dispatcher作初始化操作：

 init\_DefaultProperties()：加载org/apache/struts2/default.properties             init\_TraditionalXmlConfigurations()：加载struts-default.xml,struts-plugin.xml,struts.xml

并将加载对的结果放入到**ConfigurationManager**中



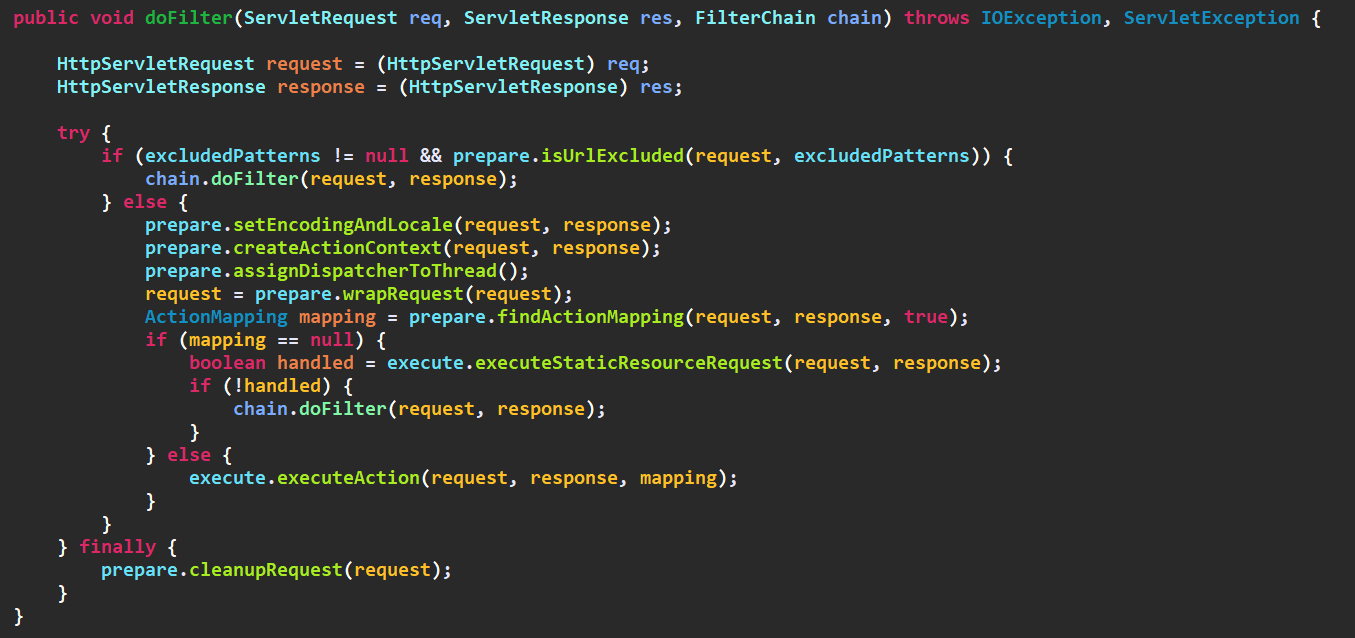
init.buildExcludedPatternsList(dispatcher)：哪些请求Struts2不拦截(exclude，排除)





StrutsConstants类和Struts.properties文件的值对应

**doFilter：**

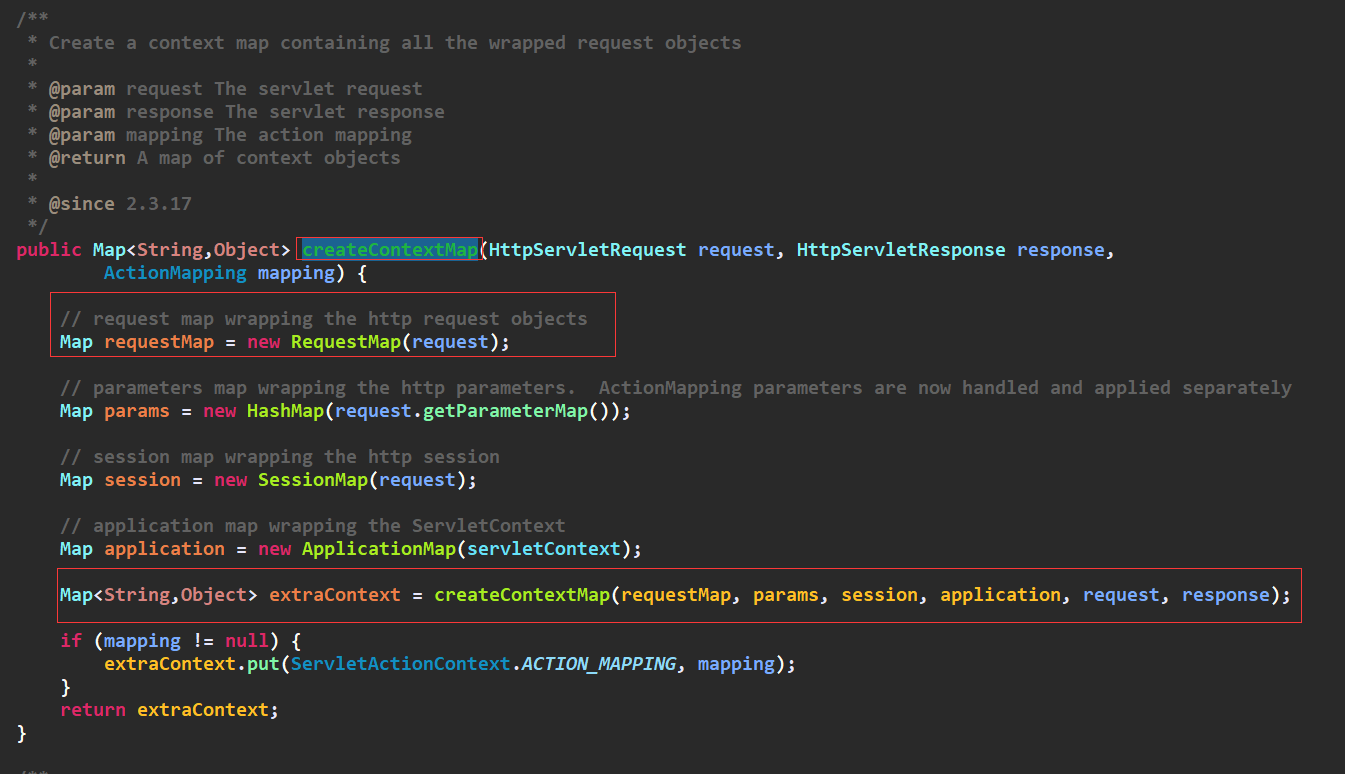


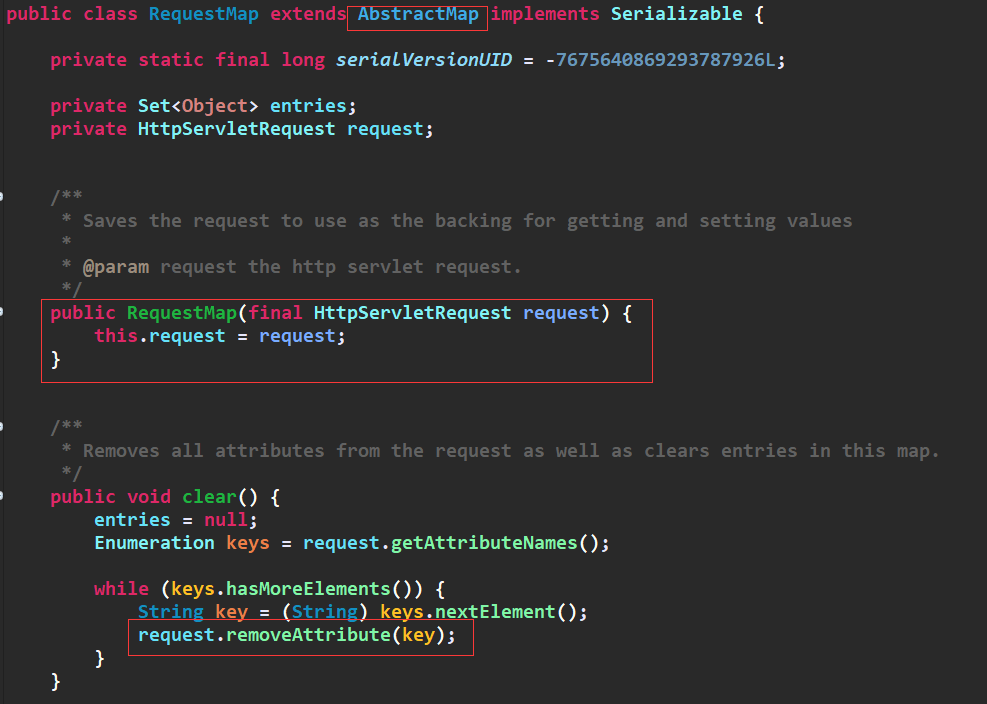
prepare.setEncodingAndLocale(request, response)：设置编码和国际化

prepare.createActionContext(request, response)：创建ActionContext



当用户第一次访问时，会创建ActionContext，并且创建Map Request，Response，Application，Session。





Struts2的Map类型的Application，Request，Session和原生的Servlet的HttpRequest之间的区别？

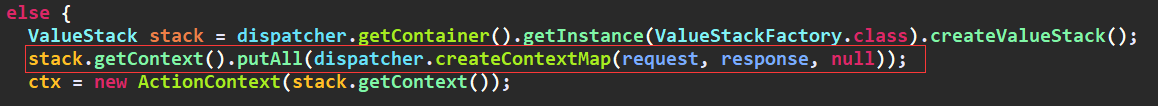
Struts2只是对原生的Servlet进行了封装，RequestMap和Struts2中的request(Map)都是都源于同一个父类，而在RequestMap对原生的Servlet进行了封装，在操作Struts2的Map的时候，实际上还是操作是Servlet的API(Map-----SessionMap------HttpSession)

在操作Request(Map)的时候，实际操作的是RequestMap，而RequestMap实际操作的又是原生的Servlet API(在RequestMap中有一个HttpServletRequest属性)





将所有的Map都封装到一个Map中，作为返回参数，返回的参数会被放入ActionContext中



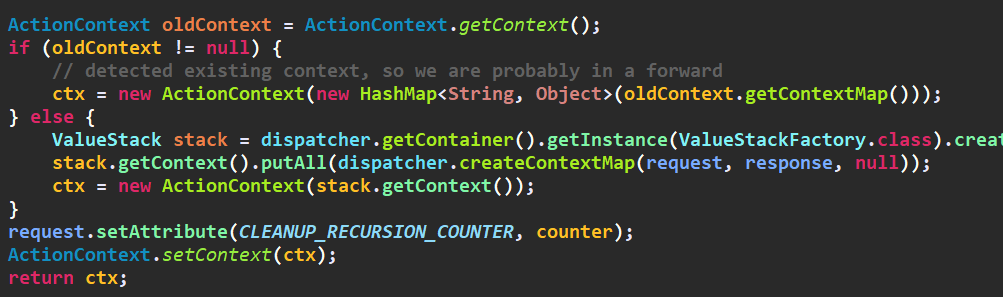
先创建了一个ValueStack(OgnlValueStack),ValueStack获取getContext(ActionContext)，然后将返回的Map放入到ValueStack的ActionContext中，然后创建ActionContext时再从ValueStack中取，这里创建ValueStack就是这个作用(创建的ValueStack只在当前else语句块有效)

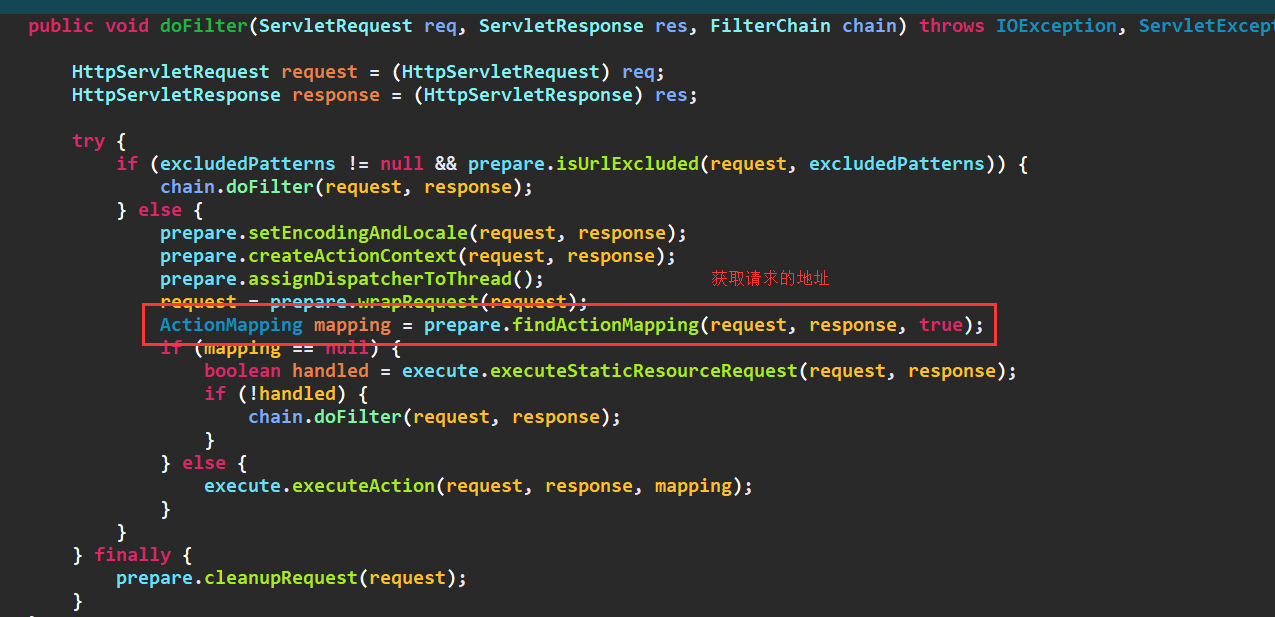
整个创建ActionContext的流程：

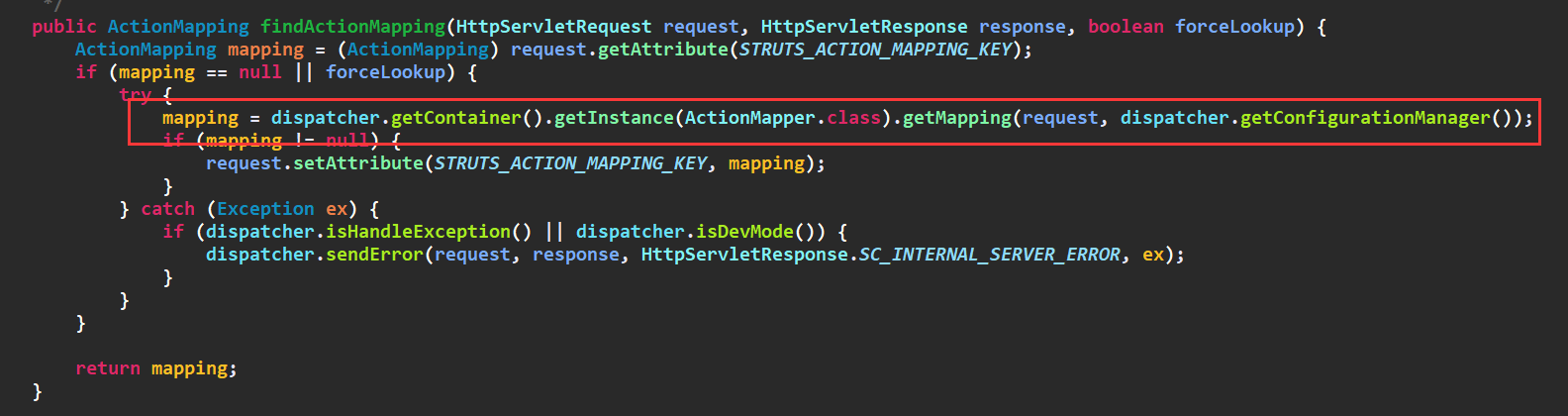
首先从从ThreadLocal中获取此ActionContext(ActionContext.getContext()底层实际调用的是ThreadLocal)

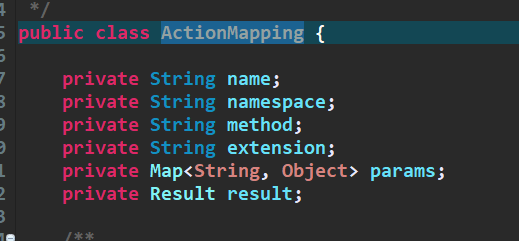
将创建好的Map放入到ActionContext中(ctx = **new** ActionContext(stack.getContext());)

将新创建的ActionContext放入到ActionContext中(**ActionContext**.*setContext*(ctx);)









通过getInstance(ActionMapper.class)获取到ActionMapper，调用getMapping方法获取请求的地址，并将请求地址分离为namespace + 请求方法的形式保存到ActionMapping中(这里传入ConfigManager的目的是获取struts.xml中的namespace，判断请求地址是否合法，这里还没有找执行方法和类的路径)

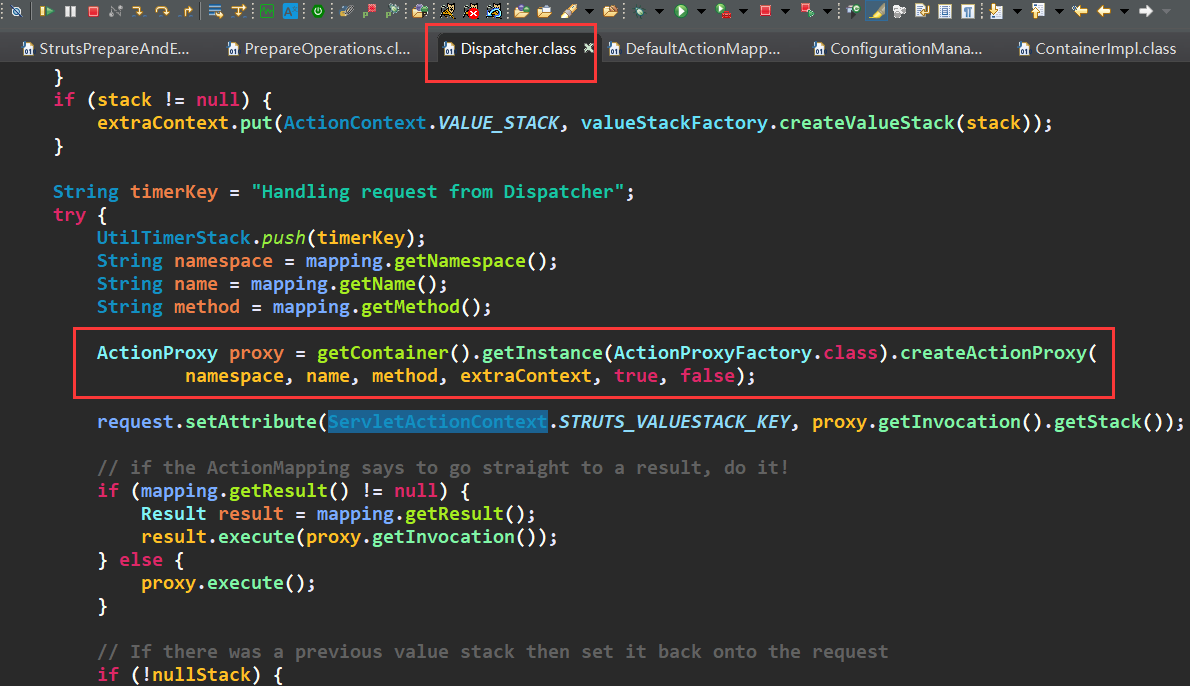
这里判断出了是否是action请求，但并未将请求类的类路径和请求方法封装到ActionMapping中(在核心拦截器的init方法中，在创建Dispatcher的时候就将配置文件初始化到ConfigManager中)



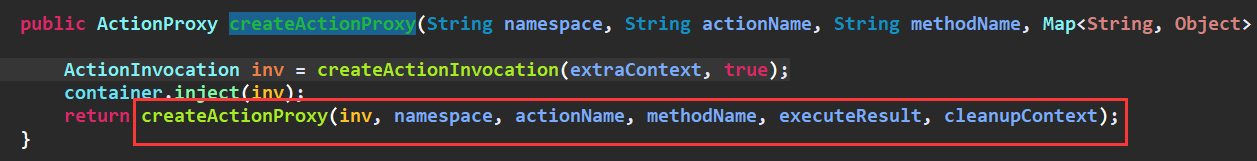


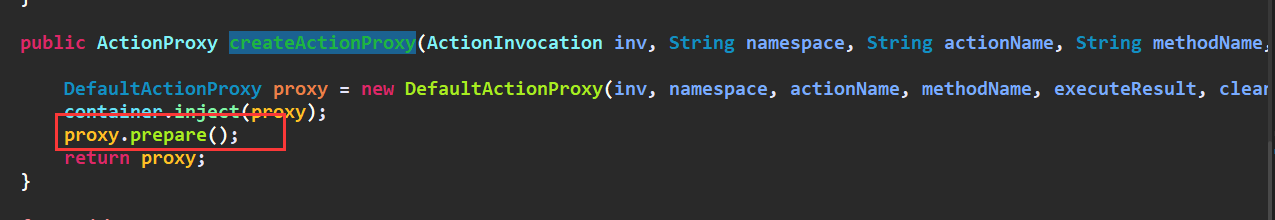
如果Mapping为空，说明是一个.jsp请求，直接执行doFilter

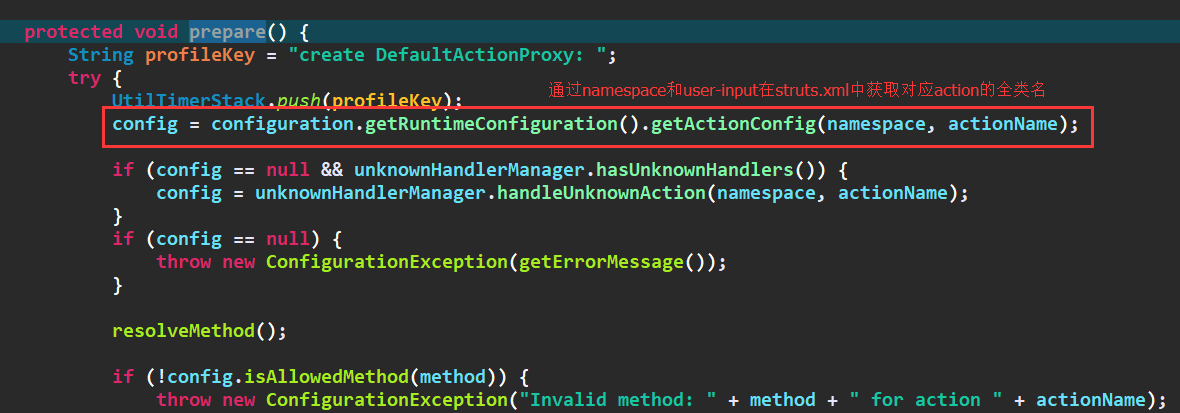
否则执行Action



创建Action的代理对象

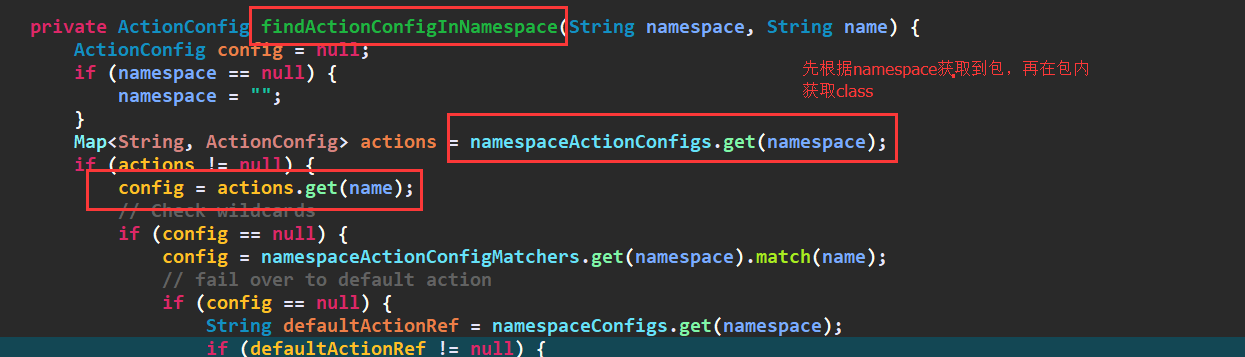




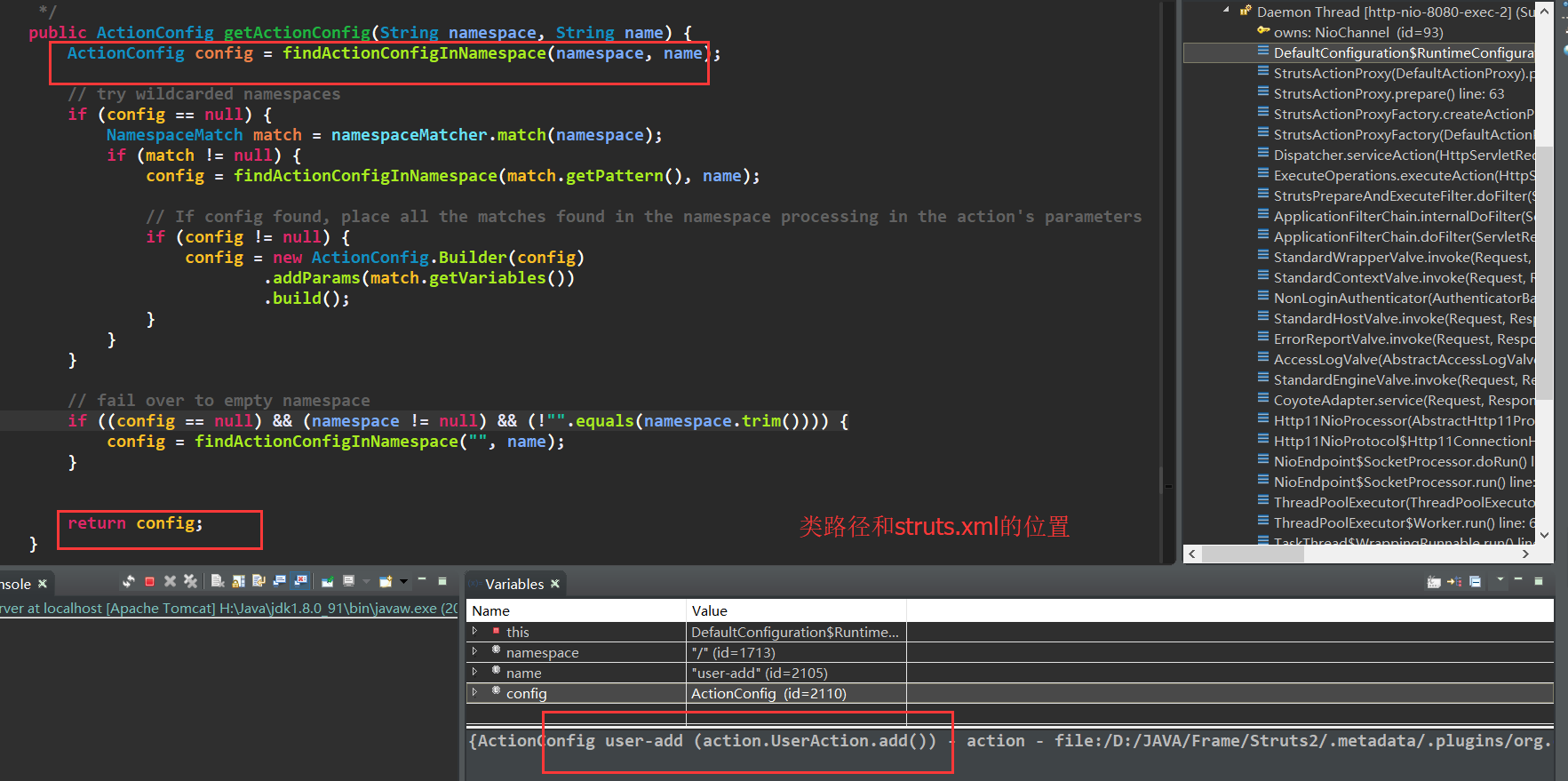


通过传入namespace和请求方法名给ActionConfig，在Struts.xml中找到对应的action的类路径(在Struts2的核心Filter的init方法中将所有的配置文件加载到ConfigurationManager中，现在又从中获取到ActionConfig对象)





先根据namespace获取所有的请求路径，封装到Map中，然后再到Map中根据请求路径找到对应的类路，请求方法，返回值，逻辑视图等信息封装到ActionConfig中

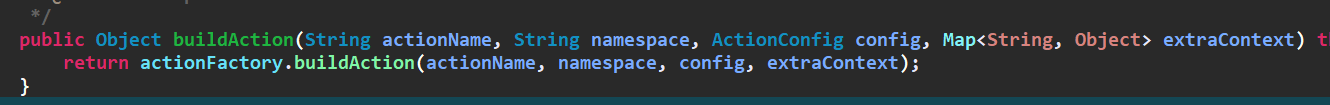




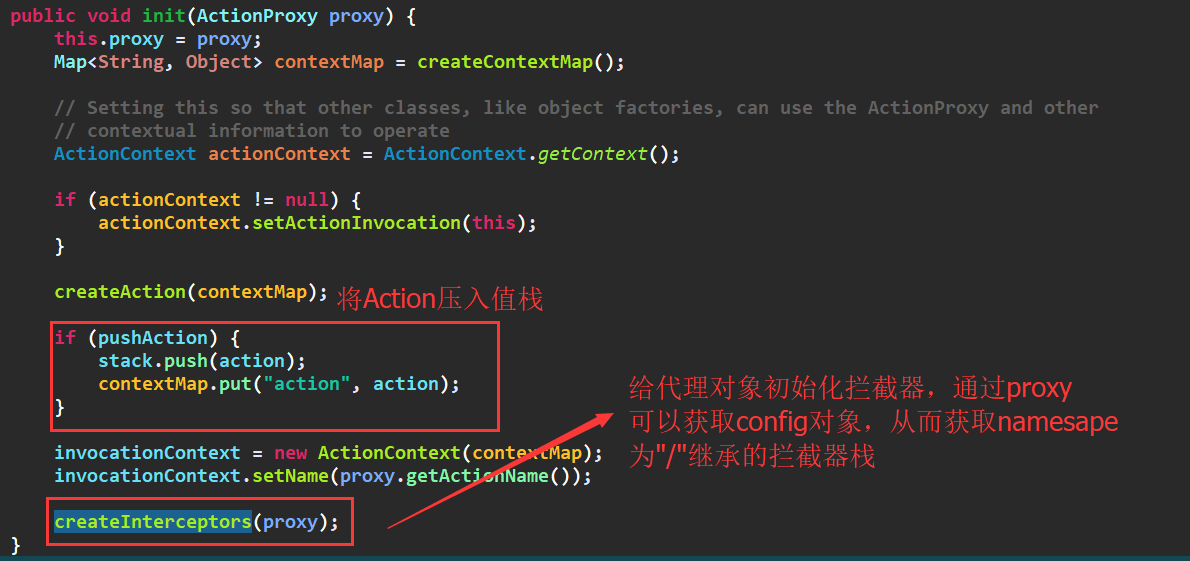
获取到config后，就可以执行创建代理对象的操作

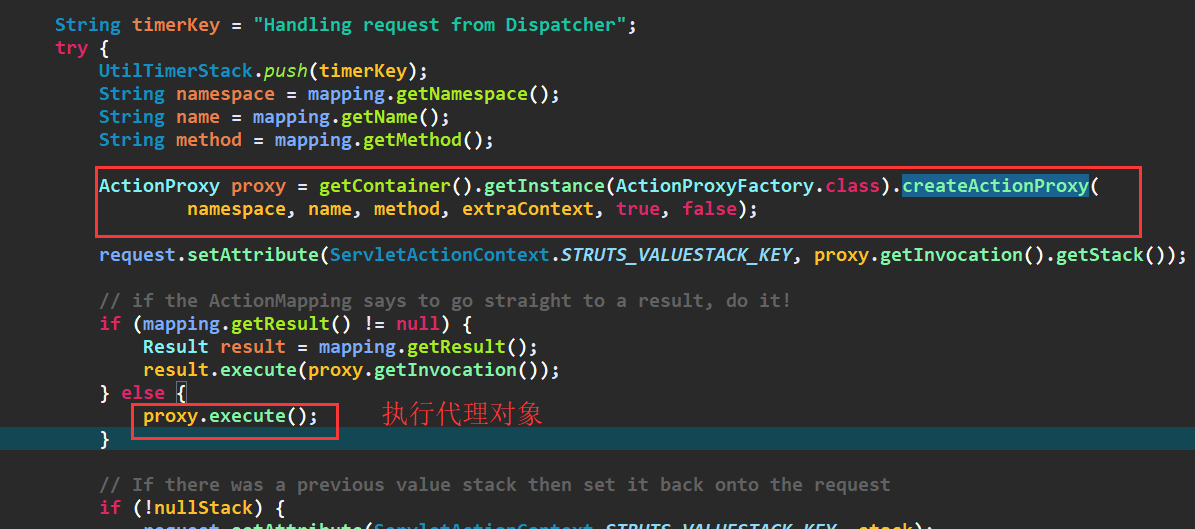




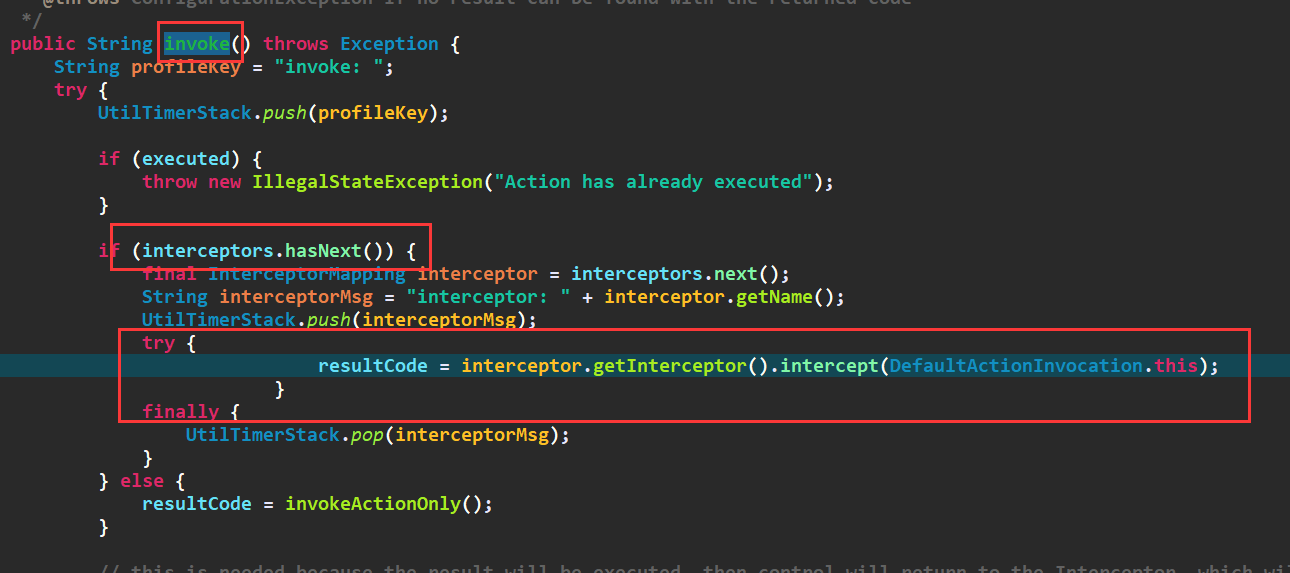




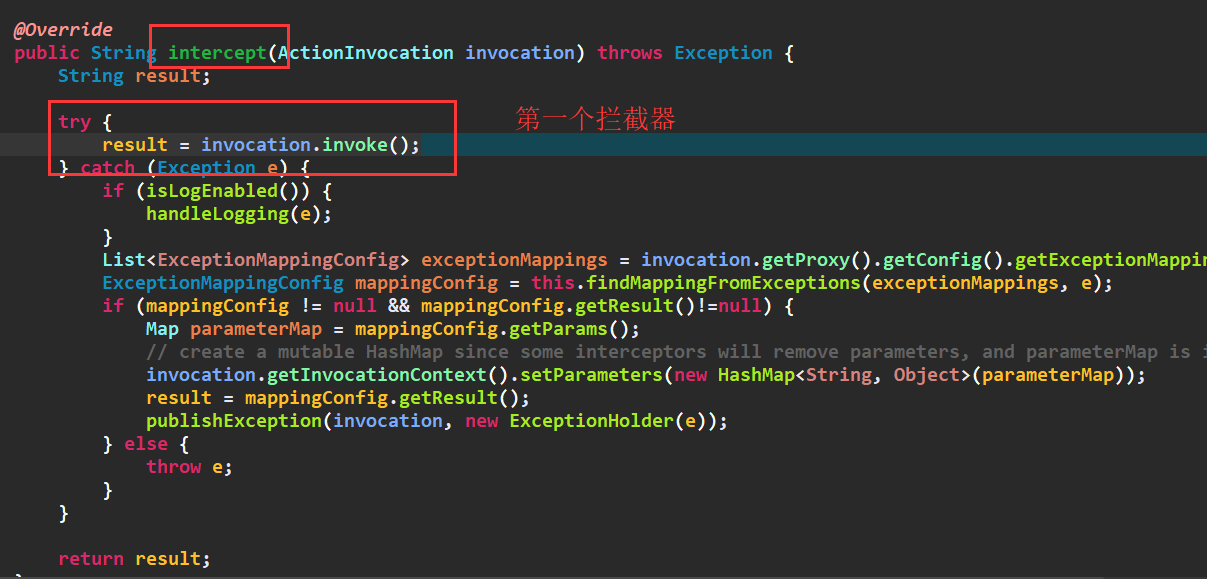


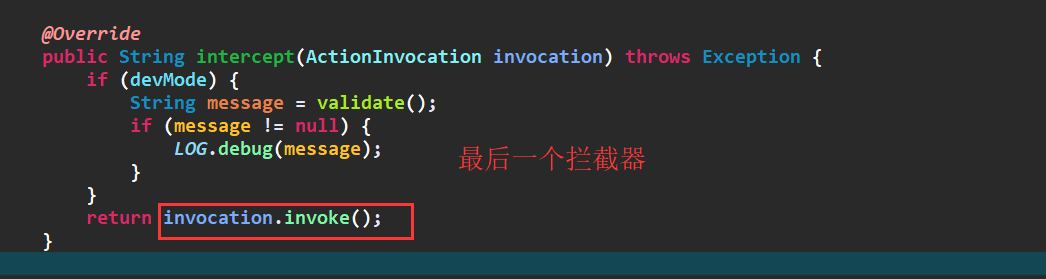






依次遍历执行拦截器栈



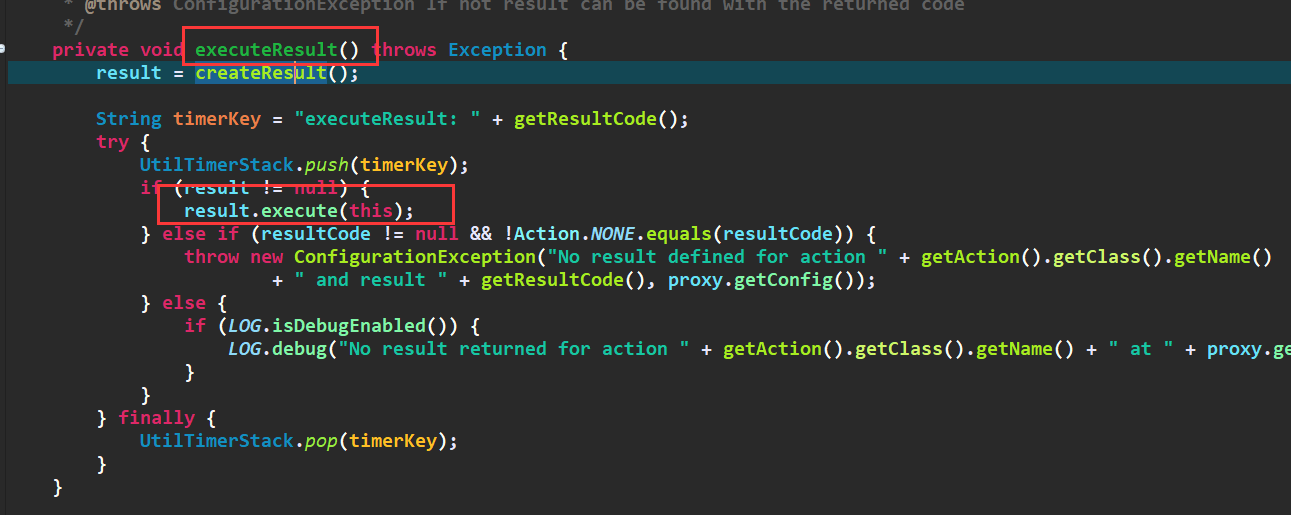


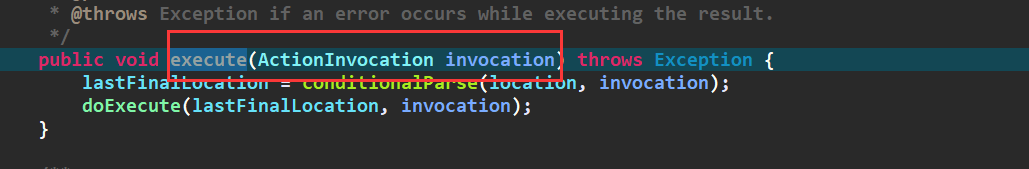






当遍历完所有拦截器后，最后执行Acting方法后，返回执行结果(resultCode)，然后执行executeResult方法







Struts2在将页面发送浏览器之后，还有进行其他操作

拦截器和过滤器的区别？

拦截器Interceptor，过滤器Filter

Filter是基于Servlet的

Struts2拦截器只能对Action的请求起作用，而过滤器则可以拦截所有请求

拦截器是Struts2的，而过滤器是Servlet的

Action为什么要创建代理对象：因为要执行拦截器栈

