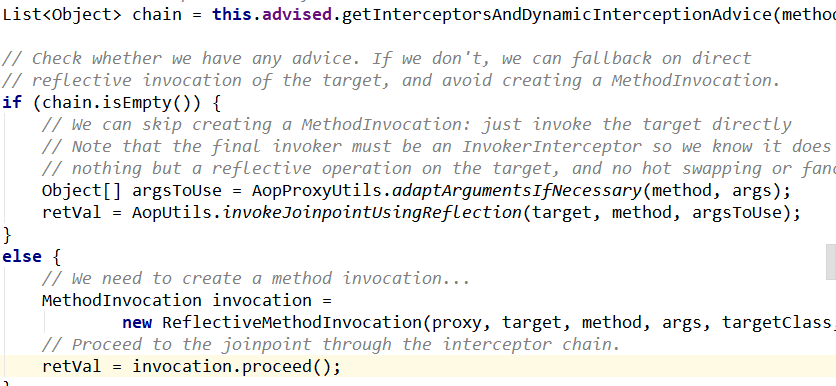
### SpringAOP:

AOP是在完成依赖注入之后也就是populate方法之后通过initializeBean方法应用BeanPostProcessor后置处理器，其中abstractAutoProxyCreator类实现了BeanPostProcessor类并重写了PostProcessafterInitialization方法，在里面创建了代理对象，对象为单例时，把代理对象注册进容器中。

以JDK动态代理方式为例，JdkDynamicAopProxy实现了InvokeHandler类，重写了invoke()方法，在invoke()方法中其获得了一个MethodInterceptor[ ]的数组调用链。



debug之后其顺序为：

0:ExposeInvocationInterceptor

1: AspectjAfterThrowingAdvice

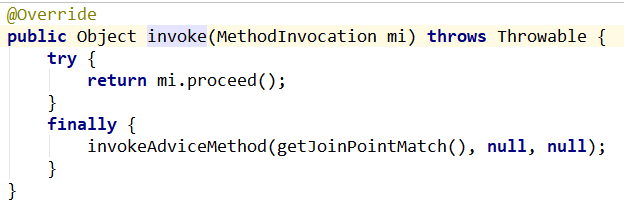
2: AfterReturnAdviceInterceptor

3:AspectjAfterAdvice其自己的调用放在finally中，所以其一定会被调用。

4:AspectjAroundAdvice

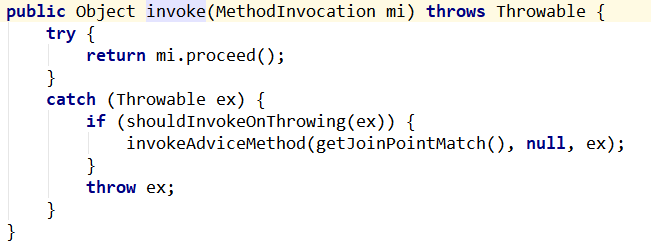
5:MethodBeforeAdviceInterceptor

其调用顺序是链式调用：先调用前一个再调用自己，所以其调用顺序是倒序调用，以AspectjAfterAdvice为例：

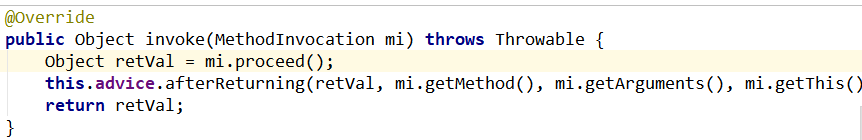


其他类型的调用链如下：

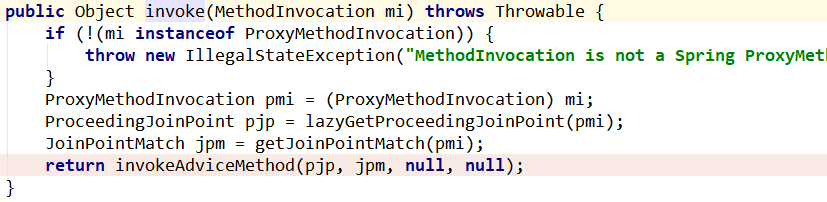
AspectjAfterThrowingAdvice：

****

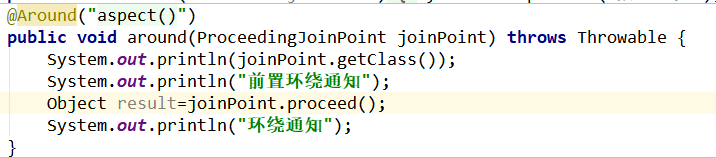
AfterReturnAdviceInterceptor

****

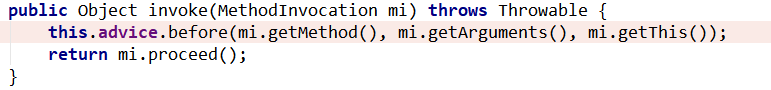
AspectjAroundAdvice：这个比较特殊，在invoke方法中，将下一个调用链即MethodBeforeAdviceInterceptor包装成ProceedingJoinPoint传入切面类加了@Around注解的方法的参数中，在该方法中可以调用ProceedingJoinPoint的proceed方法，在proceed方法中最终会调用MethodBeforeAdviceInterceptor的invoke方法。也就是说如果不调用ProceedingJoinPoint的proceed方法，则前置通知不能生效。

****

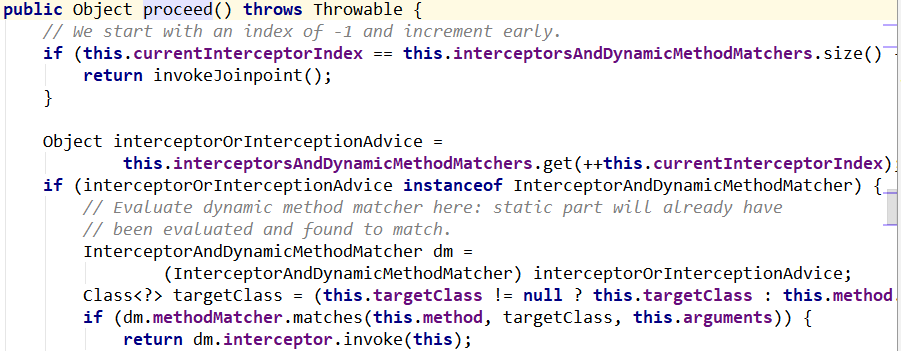
以自己写的切面类为例：

****

MethodBeforeAdviceInterceptor

****

继承自MethodInvocation的ReflectiveMethodInvocation的proceed方法：

****

如果最后一个MethodInvocation，则调用被代理类的方法。

### Springboot：

小细节：

在查看某个自动配置时可以通过在AutoConfigurationImportSelector中的process方法中的autoConfigurationEntries属性来查看，里面有哪些自动装配类从而进行相应的配置。

**备注：Springboot的自动配置的信息从**AutoConfigurationImportSelector这个类中读取的。后续再复习一下原理。

### Spring与springMVC整合：

xml整合：spring-mvc.xml，applicationContext.xml，web.xml。

在web.xml中：

ContextLoaderListener实现了ServletContextListener，在tomcat容器启动完成后会调用contextInitialized方法来初始化父容器。

<!-- 配置spring ioc容器 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:config/applicationContext.xml</param-value>

</context-param>

<!-- Bootstraps the root web application context before servlet initialization -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

在dispathcerServlet调用init方法（在父类httpServletBean中）进行初始化，init方法调用initServletBean()方法，并在该方法中初始化子容器。

<!-- 配置springmvc 的DispatcherServlet -->

<servlet>

<servlet-name>dispatcherServlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:config/spring-mvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<!-- Map all requests to the DispatcherServlet for handling -->

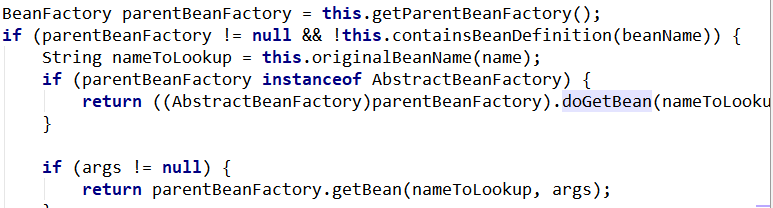
<servlet-mapping>

<servlet-name>dispatcherServlet</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

在配置spring-mvc.xml和applicationContext.xml时，父容器和子容器扫描的包分开。拿单例来说，往容器中注入对象时，首先看该容器中是否有该对象，若没有则先看是否有父容器并且在子容器中没有该类的BeanDefinition。这时候去父容器中取对象，否则在子容器中创建。



**注意：这里是该类的BeanDefinition，如果在spring-mvc.xml中也配置了除controller层外的类,则也存在这些类的Beandefinition，那么即使父类中有这些类的对象，还是会加载并注册进子类容器中。所以该对象会在子父类容器中各存在一份，浪费资源。另外，如果在spring-mvc中没有配置开启AOP代理，则其加载进子容器中的是没有被代理过的对象，这时，当从子容器取对象时，会优先从子容器中取对象，从而拿到没有被代理的对象。**

Xml配置整合参考网站：https://blog.csdn.net/qq\_41401555/article/details/81009804