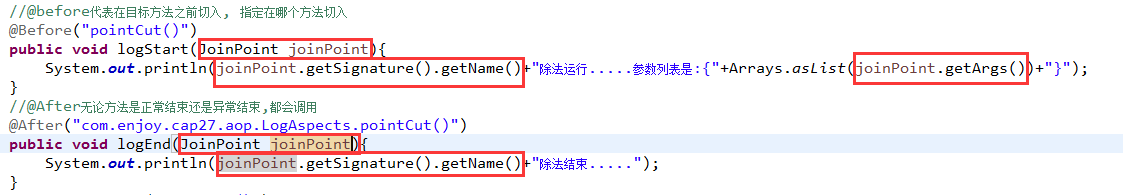
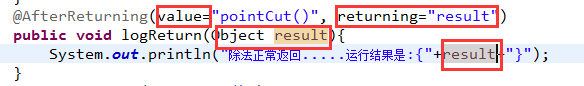
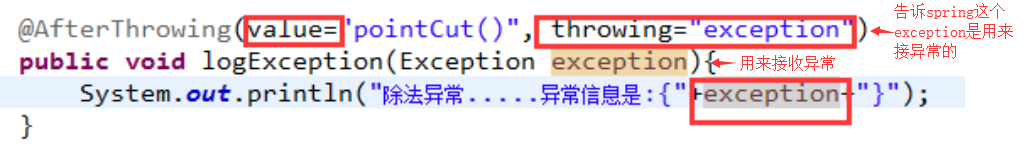
使用JoinPoint可以拿到相关的内容, 比如方法名, 参数



那么方法正常返回, 怎么拿方法的返回值呢?



那么如果是异常呢?定义



**小结:** AOP看起来很麻烦, 只要3步就可以了:

1,将业务逻辑组件和切面类都加入到容器中, 告诉spring哪个是切面类(@Aspect)

2,在切面类上的每个通知方法上标注通知注解, 告诉Spring何时运行(写好切入点表达式,参照官方文档)

3,开启基于注解的AOP模式 @EableXXXX

**九 CAP27 AOP源码透析**

\* AOP原理：【看给容器中注册了什么组件，这个组件什么时候工作，这个组件的功能是什么？】

\* **@EnableAspectJAutoProxy**；核心从这个入手,AOP整个功能要启作用,就是靠这个,加入它才有AOP

跟进**@EnableAspectJAutoProxy**源码:

//导入了此类,点进去看

@Import(AspectJAutoProxyRegistrar.**class**)

**public** **@interface** EnableAspectJAutoProxy {

//proxyTargetClass属性，默认false，采用JDK动态代理织入增强(实现接口的方式)；如果设为true，则采用CGLIB动态代理织入增强

**boolean** proxyTargetClass() **default** **false**;

//通过aop框架暴露该代理对象，aopContext能够访问

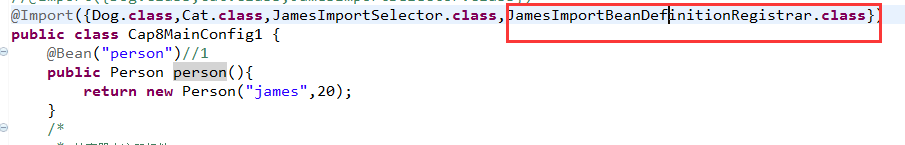
**boolean** exposeProxy() **default** **false**;

}

它引入AspectJAutoProxyRegistrar, 并实现了ImportBeanDefinitionRegistrar接口



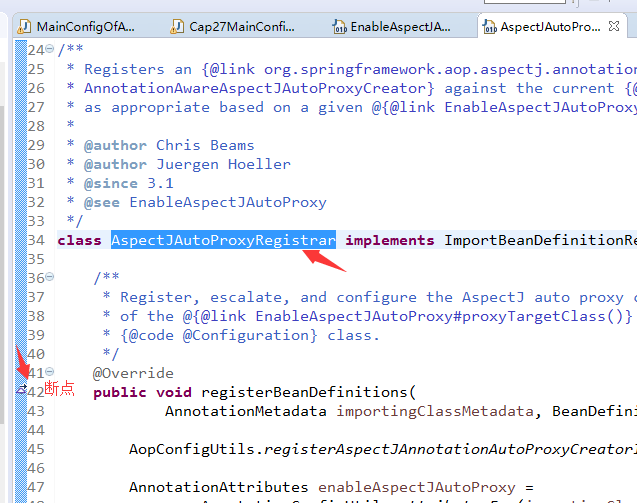
ImportBeanDefinitionRegistrar接口作用: 能给容器中自定义注册组件, 以前也使用过, 比如我们以前也使用过这个类



在**AspectJAutoProxyRegistrar**里可以自定义注册一些bean

那么注册了什么bean呢? 以debug模式进行测试一下

给**AspectJAutoProxyRegistrar类**的**registerBeanDefinitions**()方法打上断点.



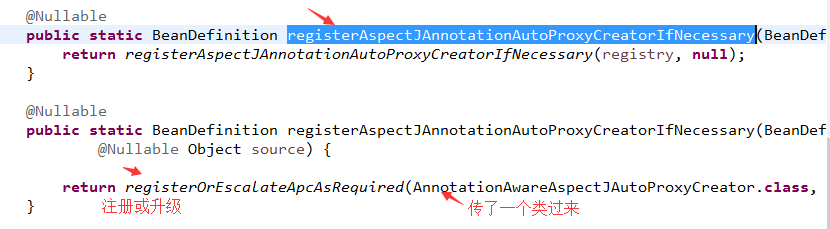
看注册bean的如何处理?

AopConfigUtils.*registerAspectJAnnotationAutoProxyCreatorIfNecessary*(registry);

注册一个这个组件, 如果有需要的话....



点进去看看

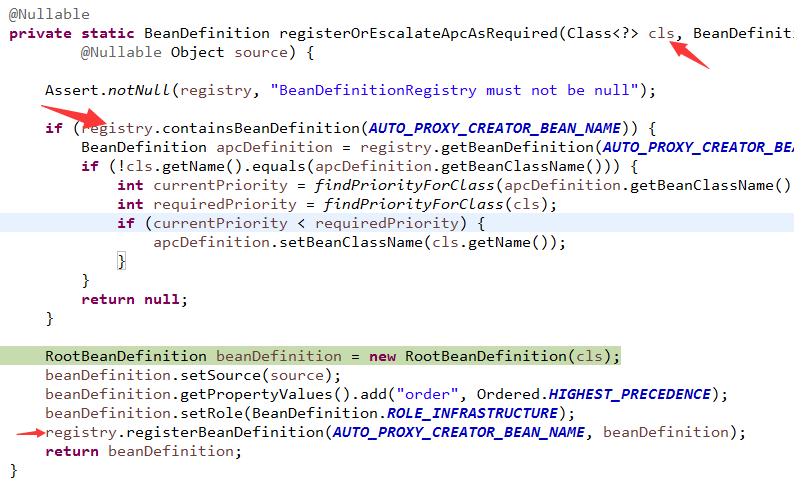


想注册一个AnnotationAwareAspectJAutoProxyCreator的组件, 如果registry已经有了的话,就执行以下操作;

但是我们的注册中还没有, 第一次, 所以来创建一个cls, 用registry把bean的定义做好, bean的名叫做internalAutoProxyCreator

其实就是利用**@EnableAspectJAutoProxy中的AspectJAutoProxyRegistrar给我们容器中注册一个**AnnotationAwareAspectJAutoProxyCreator组件;

翻译过来其实就叫做 ”注解装配模式的ASPECT切面自动代理创建器”组件



判断if(registry.containsBeanDefinition(*ATUO\_PROXY\_CREATOR\_BEAN\_NAME*))

{

如果容器中bean已经有了 internalAutoProxyCreator, 执行内部内容

}

else

创建AnnotationAwareAspectJAutoProxyCreator信息; 把此bean注册在registry中.

做完后, 相当于

其实就是 *ATUO\_PROXY\_CREATOR\_BEAN\_NAME值为*internalAutoProxyCreator,给容器中注册internalAutoProxyCreator组件, 该组件类型为AnnotationAwareAspectJAutoProxyCreator.class

可以打开之前讲过的Cap6Test, 用到了registry...

因此我们要重点研究AnnotationAwareAspectJAutoProxyCreator组件(ASPECT自动代理创建器), 研究这个透了, 整个原理也就明白了, 所有的原理就是看容 器注册了什么组件, 这个组件什么时候工作, 及工作时候的功能是什么? 只要把这几个研究清楚了,原理就都清楚了.

AnnotationAwareAspectJAutoProxyCreator神奇的组件分析:

类关系图如下,继承关系:

AnnotationAwareAspectJAutoProxyCreator：

\* AnnotationAwareAspectJAutoProxyCreator

\* ->AspectJAwareAdvisorAutoProxyCreator

\* ->AbstractAdvisorAutoProxyCreator

\* ->AbstractAutoProxyCreator

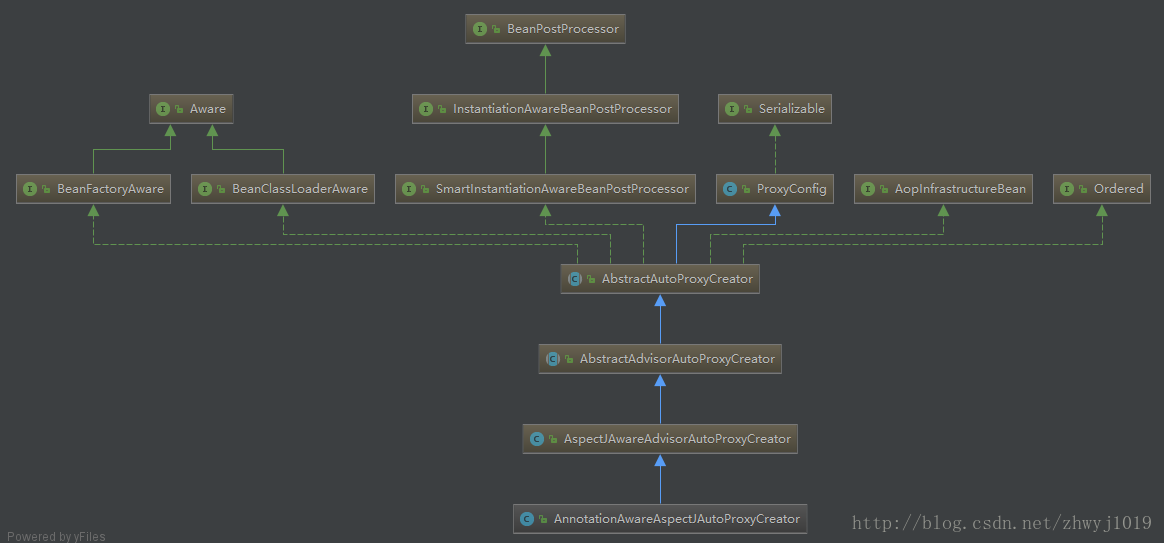
\* implements SmartInstantiationAwareBeanPostProcessor, BeanFactoryAware

\* 关注后置处理器（在bean初始化完成前后做事情）、自动装配BeanFactory

SmartInstantiationAwareBeanPostProcessor: bean的后置处理器

BeanFactoryAware 能把beanFacotry bean工厂传进来

通过分析以上的bean继承关系我们发现, 具有BeanPostProcessor特点, 也有Aware接口的特点, 实现了BeanFactoryAware 接口



那我们来分析做为beanPostProcessor后置处理器做了哪些工作, 做为BeanFactoryAware又做了哪些工作

**一, 分析创建和注册AnnotationAwareAspectJAutoProxyCreator的流程:**

1）、register()传入配置类，准备创建ioc容器

2）、注册配置类，调用refresh（）刷新创建容器；

3）、registerBeanPostProcessors(beanFactory);注册bean的后置处理器来方便拦截bean的创建(主要是分析创建AnnotationAwareAspectJAutoProxyCreator)；

1）、 先获取ioc容器已经定义了的需要创建对象的所有BeanPostProcessor

2）、给容器中加别的BeanPostProcessor

3）、优先注册实现了PriorityOrdered接口的BeanPostProcessor；

4）、再给容器中注册实现了Ordered接口的BeanPostProcessor；

5）、注册没实现优先级接口的BeanPostProcessor；

6）、注册BeanPostProcessor，实际上就是创建BeanPostProcessor对象，保存在容器中；

创建internalAutoProxyCreator的BeanPostProcessor【其实就是AnnotationAwareAspectJAutoProxyCreator】

1）、创建Bean的实例

2）、populateBean；给bean的各种属性赋值

3）、initializeBean：初始化bean；

1）、invokeAwareMethods()：处理Aware接口的方法回调

2）、applyBeanPostProcessorsBeforeInitialization()：应用后置处理器的postProcessBeforeInitialization（）

3）、invokeInitMethods()；执行自定义的初始化方法

4）、applyBeanPostProcessorsAfterInitialization()；执行后置处理器的postProcessAfterInitialization（）；

4）、BeanPostProcessor(AnnotationAwareAspectJAutoProxyCreator)创建成功；--》aspectJAdvisorsBuilder

7）、把BeanPostProcessor注册到BeanFactory中；

beanFactory.addBeanPostProcessor(postProcessor);

注意:以上是创建和注册AnnotationAwareAspectJAutoProxyCreator的过程

AnnotationAwareAspectJAutoProxyCreator => InstantiationAwareBeanPostProcessor

**二, 如何创建增强的Caculator增强bean的流程:**

1,refresh--->finishBeanFactoryInitialization(beanFactory);完成BeanFactory初始化工作；创建剩下的单实例bean

1）、遍历获取容器中所有的Bean，依次创建对象getBean(beanName);

getBean->doGetBean()->getSingleton()->

2）、创建bean

【AnnotationAwareAspectJAutoProxyCreator在所有bean创建之前会有一个拦截，InstantiationAwareBeanPostProcessor，会调用postProcessBeforeInstantiation()】

2.1）、先从缓存中获取当前bean，如果能获取到，说明bean是之前被创建过的，直接使用，否则再创建；

只要创建好的Bean都会被缓存起来

2.2）、createBean（）;创建bean；

AnnotationAwareAspectJAutoProxyCreator 会在任何bean创建之前先尝试返回bean的实例

【BeanPostProcessor是在Bean对象创建完成初始化前后调用的】

【InstantiationAwareBeanPostProcessor是在创建Bean实例之前先尝试用后置处理器返回对象的】

2.2.1）、resolveBeforeInstantiation(beanName, mbdToUse);解析BeforeInstantiation,如果能返回代理对象就使用，如果不能就继续,后置处理器先尝试返回对象；

bean = applyBeanPostProcessorsBeforeInstantiation（）：

拿到所有后置处理器，如果是InstantiationAwareBeanPostProcessor;

就执行postProcessBeforeInstantiation

if (bean != null) {

bean = applyBeanPostProcessorsAfterInitialization(bean, beanName);

}

2.2.2）、doCreateBean(beanName, mbdToUse, args);真正的去创建一个bean实例；和单实例bean创建流程一样；

**三,【AnnotationAwareAspectJAutoProxyCreator】作用:**

**InstantiationAwareBeanPostProcessor**

：

1）、每一个bean创建之前，调用postProcessBeforeInstantiation()；

关心MathCalculator和LogAspect的创建

1）、判断当前bean是否在advisedBeans中（保存了所有需要增强bean）

2）、判断当前bean是否是基础类型的Advice、Pointcut、Advisor、AopInfrastructureBean，

或者是否是切面（@Aspect）

3）、是否需要跳过

1）、获取候选的增强器（切面里面的通知方法）【List<Advisor> candidateAdvisors】

每一个封装的通知方法的增强器是 InstantiationModelAwarePointcutAdvisor；

判断每一个增强器是否是 AspectJPointcutAdvisor 类型的；返回true

2）、永远返回false

2）、创建对象

postProcessAfterInitialization；

return wrapIfNecessary(bean, beanName, cacheKey);//包装如果需要的情况下

1）、获取当前bean的所有增强器（通知方法） Object[] specificInterceptors

1、找到候选的所有的增强器（找哪些通知方法是需要切入当前bean方法的）

2、获取到能在bean使用的增强器。

3、给增强器排序

2）、保存当前bean在advisedBeans中；

3）、如果当前bean需要增强，创建当前bean的代理对象；

1）、获取所有增强器（通知方法）

2）、保存到proxyFactory

3）、创建代理对象：Spring自动决定

JdkDynamicAopProxy(config);jdk动态代理；

ObjenesisCglibAopProxy(config);cglib的动态代理；

4）、给容器中返回当前组件使用cglib增强了的代理对象；

5）、以后容器中获取到的就是这个组件的代理对象，执行目标方法的时候，代理对象就会执行通知方法的流程；