关于AnnotationAwareAspectJAutoProxyCreator和Caculator动态代理增强实例的创建见第六次笔记，创建完后开始调用目标方法，可以目标方法div处打上断点，再F5跟进；

注意：本次（第七次）视频可反复多看几遍，可理解AOP核心流程。

**一，目标方法执行（caculator.div()方法执行切面拦截）**；

容器中保存了组件的代理对象（cglib增强后的对象），这个对象里面保存了详细信息（比如增强器，目标对象，xxx）；

1）、CglibAopProxy.intercept();拦截目标方法的执行

2）、根据ProxyFactory对象获取将要执行的目标方法拦截器链；

List<Object> chain = this.advised.getInterceptorsAndDynamicInterceptionAdvice(method, targetClass);

1）、List<Object> interceptorList保存所有拦截器 5

一个默认的ExposeInvocationInterceptor 和 4个增强器；

2）、遍历所有的增强器，将其转为Interceptor；

registry.getInterceptors(advisor);

3）、将增强器转为List<MethodInterceptor>；

如果是MethodInterceptor，直接加入到集合中

如果不是，使用AdvisorAdapter将增强器转为MethodInterceptor；

转换完成返回MethodInterceptor数组；

3）、如果没有拦截器链，直接执行目标方法;

拦截器链（每一个通知方法又被包装为方法拦截器，利用MethodInterceptor机制）

4）、如果有拦截器链，把需要执行的目标对象，目标方法，

拦截器链等信息传入创建一个 CglibMethodInvocation 对象，

并调用 Object retVal = mi.proceed();

5）、拦截器链的触发过程;

1)、如果没有拦截器执行执行目标方法，或者拦截器的索引和拦截器数组-1大小一样（指定到了最后一个拦截器）执行目标方法；

2)、链式获取每一个拦截器，拦截器执行invoke方法，每一个拦截器等待下一个拦截器执行完成返回以后再来执行；

拦截器链的机制，保证通知方法与目标方法的执行顺序；

总结：

1）、 @EnableAspectJAutoProxy 开启AOP功能

2）、 @EnableAspectJAutoProxy 会给容器中注册一个组件 AnnotationAwareAspectJAutoProxyCreator

3）、AnnotationAwareAspectJAutoProxyCreator是一个后置处理器；

4）、容器的创建流程：

1）、registerBeanPostProcessors（）注册后置处理器；创建AnnotationAwareAspectJAutoProxyCreator对象

2）、finishBeanFactoryInitialization（）初始化剩下的单实例bean

1）、创建业务逻辑组件和切面组件

2）、AnnotationAwareAspectJAutoProxyCreator拦截组件的创建过程

3）、组件创建完之后，判断组件是否需要增强

是：切面的通知方法，包装成增强器（Advisor）;给业务逻辑组件创建一个代理对象（cglib）；

5）、执行目标方法：

1）、代理对象执行目标方法

2）、CglibAopProxy.intercept()；

1）、得到目标方法的拦截器链（增强器包装成拦截器MethodInterceptor）

2）、利用拦截器的链式机制，依次进入每一个拦截器进行执行；

3）、效果：

正常执行：前置通知-》目标方法-》后置通知-》返回通知

出现异常：前置通知-》目标方法-》后置通知-》异常通知

拦截流程图如下：

