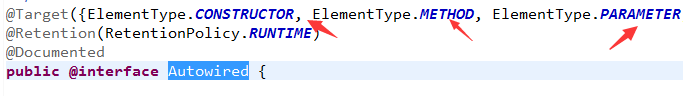
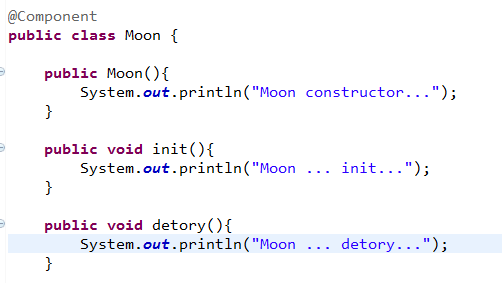
一, @Autowired方法, 参数, 构造方法都可加载, 跟进源码看看



A>放到方法上的测试步骤:

新建Moon.java



新建Sun.java



setMoon()方法使用的参数,自定义类型的值从IOC容器中获取, 方法里的moon会从容器中拿到

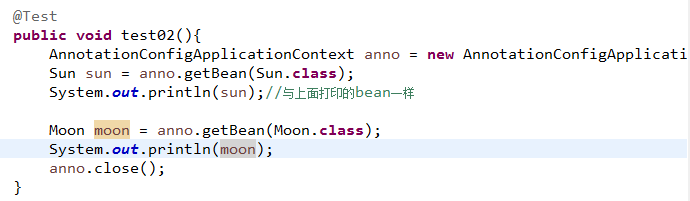
以下是验证方式

在配置类要加入扫描bean包



测试:

新建test02()测试方法

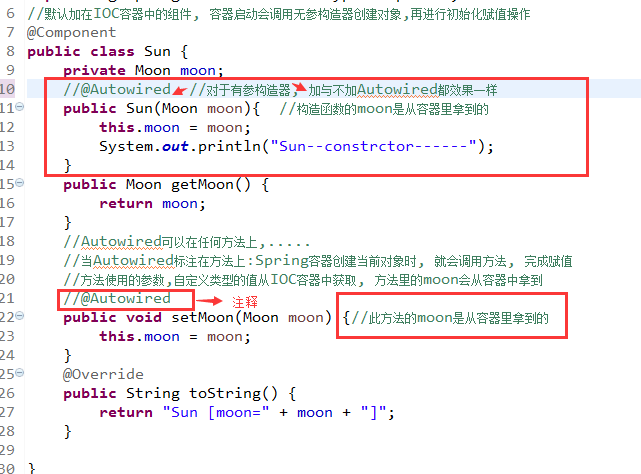


结果如下:(为同一个bean)



方法使用的参数,自定义类型的值从IOC容器中获取, 方法里的moon会从容器中拿到

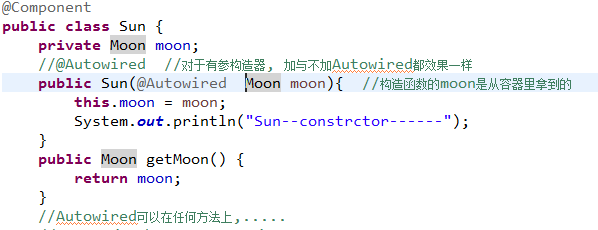
1. 将Autowired标记在有参构造器



构造函数的moon是从容器里拿到的

执行test02()测试;

同样,也可以放在构造器的参数位置也可以获取到IOC容器的bean.



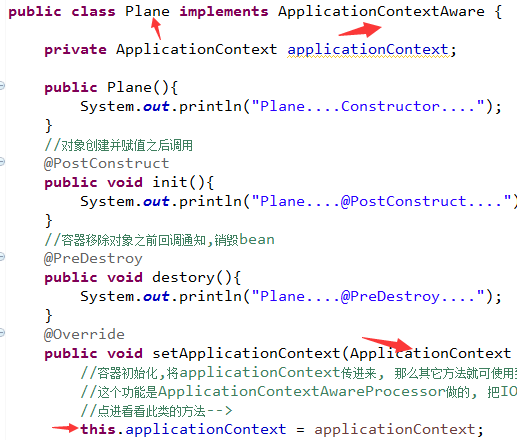
结论: 不管@Autowired是放到参数, 方法还是构造方法, 都是从容器里取到的bean...

**二 自动装配:Aware注入spring底层组件原理**

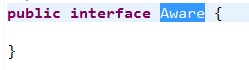
自定义组件想要使用Spring容器底层的组件(ApplicationContext, BeanFactory, ......)

思路: 自定义组件实现xxxAware, 在创建对象的时候, 会调用接口规定的方法注入到相关组件:Aware

之前讲过Plane.java就是使用这个



CTRL+SHIFT+T 找到Aware



查看有哪些接口继承了Aware接口



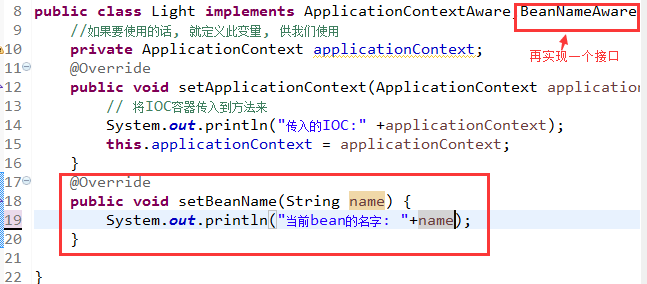
使用ApplicationContextAware接口为例, 实现接口

步骤:

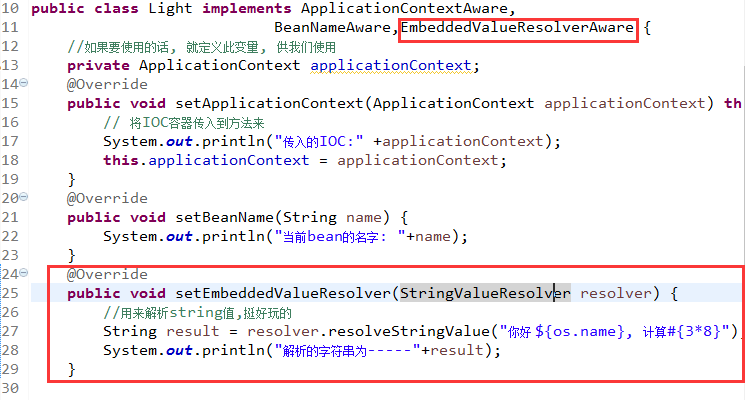
1, 新建Light.java类, 实现ApplicaitonContextAware接口



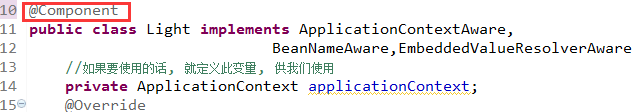
2,实现BeanNameAware接口



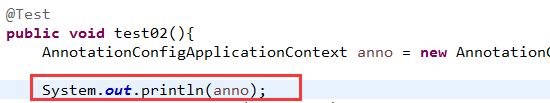
2,实现EmbeddedValueResolverAware接口



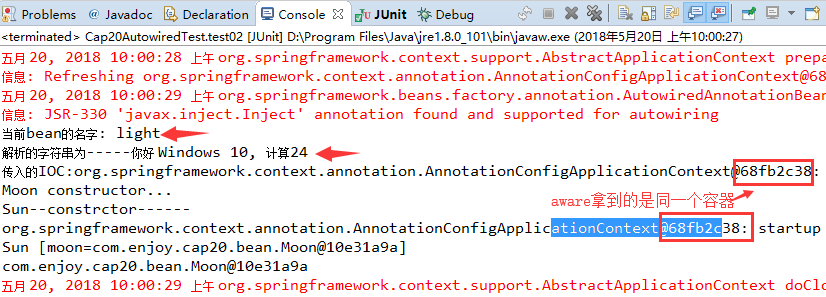
3, 将Light类加入@Component, 声明为被扫描的组件



4,将test02()加入打印anno容器, 用来比如



5,测试, 执行test()02;



**总结**:把Spring底层的组件可以注入到自定义的bean中,ApplicationContextAware是利用ApplicationContextAwareProcessor来处理的, 其它XXXAware也类似, 都有相关的Processor来处理, 其实就是后置处理器来处理;

XXXAware---->功能使用了XXXProcessor来处理的, 这就是后置处理器的作用;

ApplicaitonContextAware--->ApplicationContextProcessor后置处理器来处理的

问题: Spring怎么把applicationContext容器注入进来的呢????

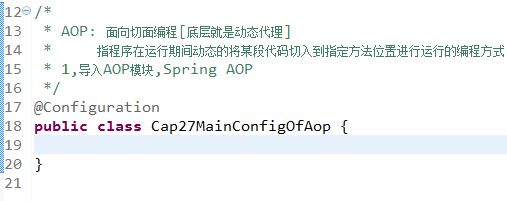
debug栈分析, 之前讲过的.

**三 CAP27 AOP功能测试**

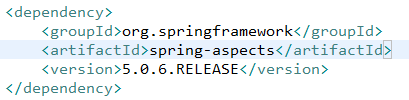
AOP: 面向切面编程[底层就是动态代理]

指程序在运行期间动态的将某段代码切入到指定方法位置进行运行的编程方式

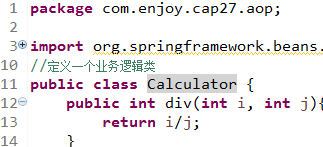
1,先建立MainConfigOfAOP配置类



2,在POM.XML中导入spring-aspects依赖包

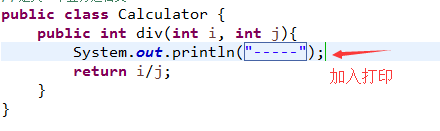


3,新建立一个业务逻辑类Calculator.java



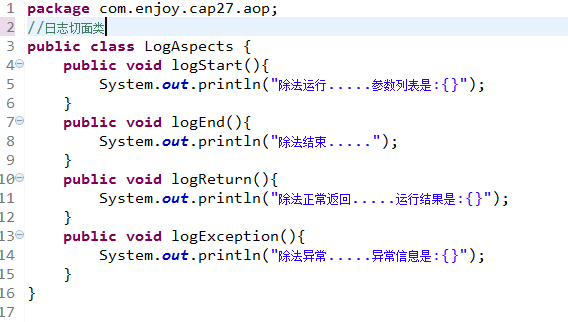
在div()方法运行之前, 记录一下日志, 运行后也记录一下,运行出异常,也打印一下

有的同学可能会这么做, 在每个方法加入个打印.



但这样会有问题, 这种方式耦合了....

新建一个日志切面类....



日志切面类的方法需要动态感知到div()方法运行到哪里了, 然后再执行, 如果除法开始, 就日志开始方法,

也叫通知方法, 分以下几种:

前置通知: logStart(),在目标方法(div)运行之前运行 (@Before)

后置通知:logEnd(), 在目标方法(div)运行结束之后运行,无论正常或异常结束 (@After)

返回通知:logReturn, 在目标方法(div)正常返回之后运行 (@AfterReturning)

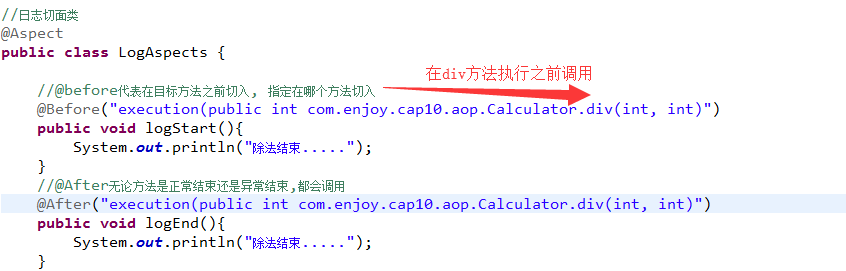
异常通知:logException, 在目标方法(div)出现异常后运行(@AfterThrowing)

环绕通知:以上没写,动态代理, 手动执行目标方法运行joinPoint.procced(),最底层通知,手动指定执行目标方法(@Around), 执行之前相当于前置通知, 执行之后相当于返回通知

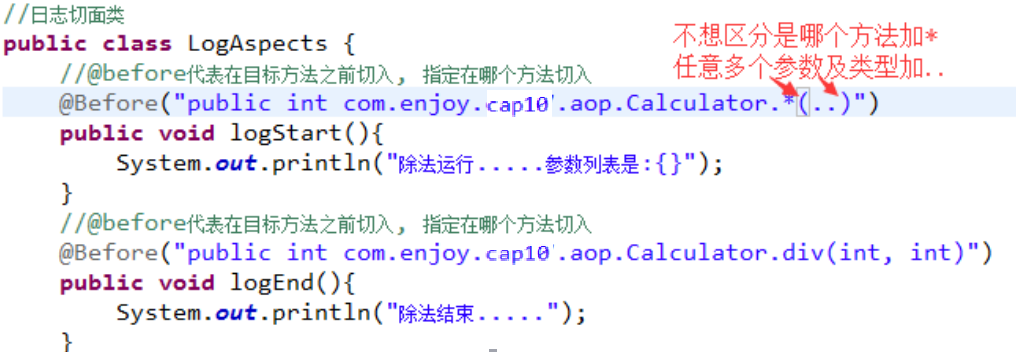
其实就是通过反射执行目标对象的连接点处的方法；

给日志切面类LogAspect的方法标注何时运行(即通知注解)

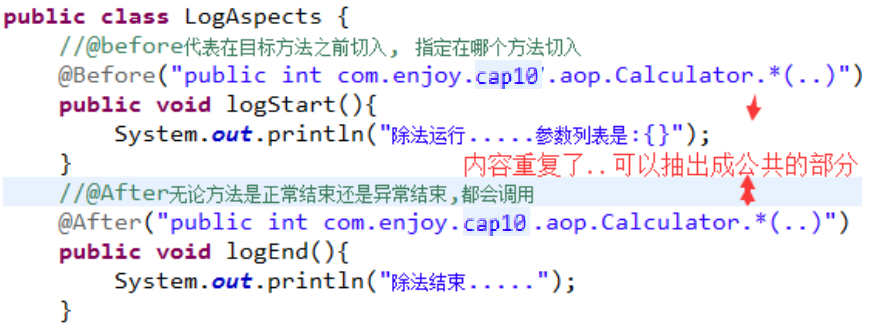
怎么加入 呢?



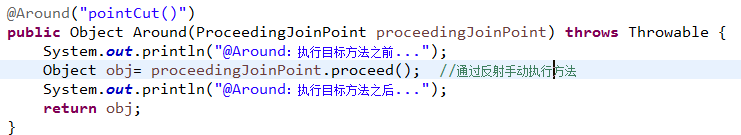
若不想区分切入了哪个方法及参数类型和个数,可以有如下指定方式:



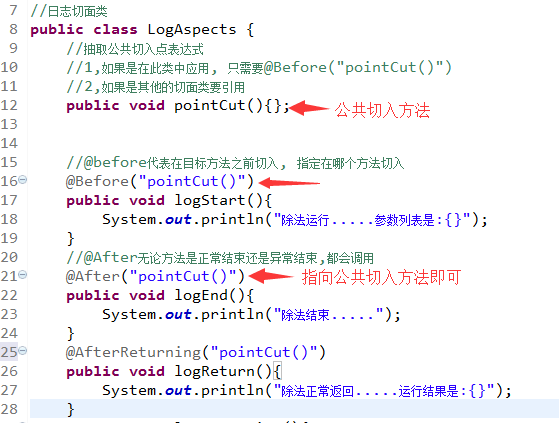
不难发现问题:注解里的内容是冗余重复的, 公共的代码应该抽出来



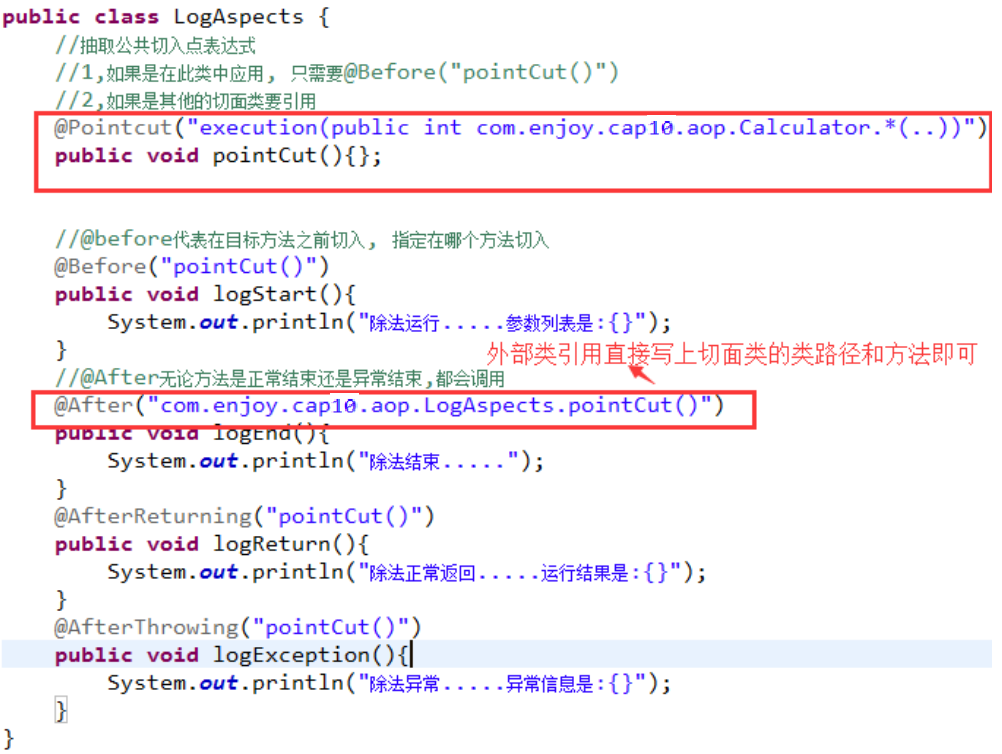
并加上Around环绕通知



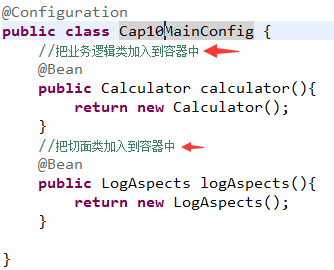
那么应该写一个公共切入点表达式.



公共切入方法加要指定的方法



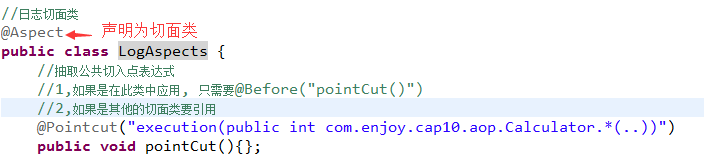
有了以上操作, 我们还需要将切面类和被切面的类, 都加入到容器中



但这个时候还会有问题, Spring无法区别以上的两个bean哪个是切面类, 哪个是业务逻辑类

怎么区别呢????

只需要给切面类LogAspects上加一个注解@Aspect即可.

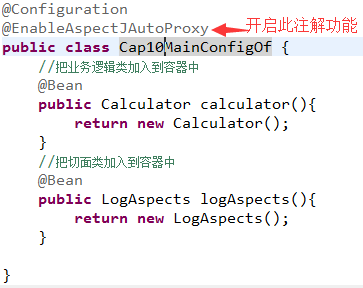


是不是就完了呢?并没有

需要开启基于注解的AOP模式

给配置类中加@EnableAspectJAutoProxy[一定得加上,关键]

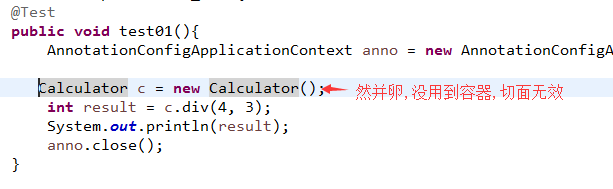
注意: 在spring以后会有很多@EnableXXXX, 表示开启某项功能, 取代XML配置



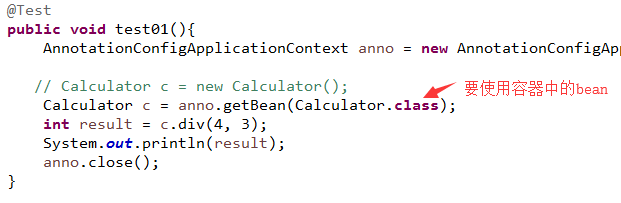
测试: 新建一个测试类Cap10Test.java

同学们在测试的过程中, 应该怎么测? 很容易出问题.

大家可能会这么写:



没用到容器, 肯定是不行的, 获取bean时使用IOC容器取出bean



运行结果如下:

