AspectJ:通往AOSD之路的最佳军火 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/145/2021_2022_AspectJ__E9_80_c104_145041.htm 在AOSD：应用AOP实现业务逻辑中， 我提出关注的接口； 其原文在javaeye上的讨论狼平方也提出可以用event，或者intercepter。 当然不同的方式可以解决不同的问题。这里要讨论一下AspectJ和Event以及intercepter的不同。1. 先来看看event的方式：需要两个对象Event和EventHandler（EventListener），event和eventHandler，属于数据契约。换句话说，牺牲了编译检查的好处，当然可以应对变化时有一定的好处。一旦需求变化，代码上的变动需要的工作量不少，更为严重的是，如狼平方所说的侵入性太强。2 接着看interceptor的AOP，相对于Event方式，虽然都是基于数据契约，都给自己做转型，但是代码量少了（因为代码生成），重要的是没有event那样的侵入性。3. 狼平方做了改进，利用AOP来做brocastEvent的工作。需要做的工作还是很多，甚至我以为这个工作还不如直接用Interceptor来得直接。如我在小议领域模型(Domain Model) 所说的Domain Service处理两个逻辑：业务规则和流程逻辑，而AOSD：应用AOP实现业务逻辑要试图解决的问题是流程上逻辑，而以上无论是那种方式，都无法解决一个问题：流程信息。假定现有两个流程如下： public class DomainService { public void bizProcessA() { SomeObject instance = doSomeAction(). } public void bizProcessB() { SomeObject instance = doSomeAction(). } public SomeObject doSomeAction() { } } 这里我们关注的目标方法是doSomeAction（），在doSomeAction中我们希望做些额

外的工作，以狼平方的例子，比如：financialService.createRequestOfMoney(...).那么无论是用event：public SomeObject doSomeAction() { brocastEvent(new Event()).}还是interceptor：public class SomeInterceptor implements MethodInterceptor { public Object invoke(MethodInvocation invoke) { obj = invoke.proceed(). // 执行被拦截的方法完成业务操作 . financialService.createRequestOfMoney(.). } } 我们注意到,这样做的结果是在bizProcessA()和bizProcessB()的不同业务流程都将导致执行financialService.createRequestOfMoney不过我们的流程逻辑要求是：在bizProcessA()的流程下，执行financialService.createRequestOfMoney而在bizProcessB的流程下，不执行financialService.createRequestOfMoney。很明显无论是之上event，还是interceptor的AOP都无法解决这个问题。面对这样的需求，就需要改进或者重构：1. 最简单也最直接改doSomeAction(...)为doSomeAction(...boolean flag)。很明显这样的做法很不理想2. 来点OO的， 将doSomeAction（...）分出去，作为一个接口，采用多态解决。不过依然需要解决动态加载问题，处理起来又需要费点手段。真的只有这两个手段了吗？不！还有，那就是AspectJAspectJ提供了如下within withincodewithincodecflowcflowbelow几个内置的pointcut，将可以做到我们要的效果，代码如下： public aspect ServiceAspect { pointcut bp(): cflow(call(void DomainService.bizProcessA(..))). pointcut action(): call(doSomeAction（）). SomeObject around() : bp() amp. action() amp. args() { SomeObject instance = proceed(). financialService.createRequestOfMoney(.). return instance. } } 很显

然，AspectJ提供了AOSD所需要的军火。BTW：本文不针对狼平方同学，只是借由其两篇blog讨论开来，解释了AspectJ优点，以及为什么AspectJ是实现AOSD的最佳也是唯一军火。事实上很多时候，解决问题的方法有很多，本文旨在阐明AOSD为我们提供了另一条思路。欢迎讨论和拍砖！