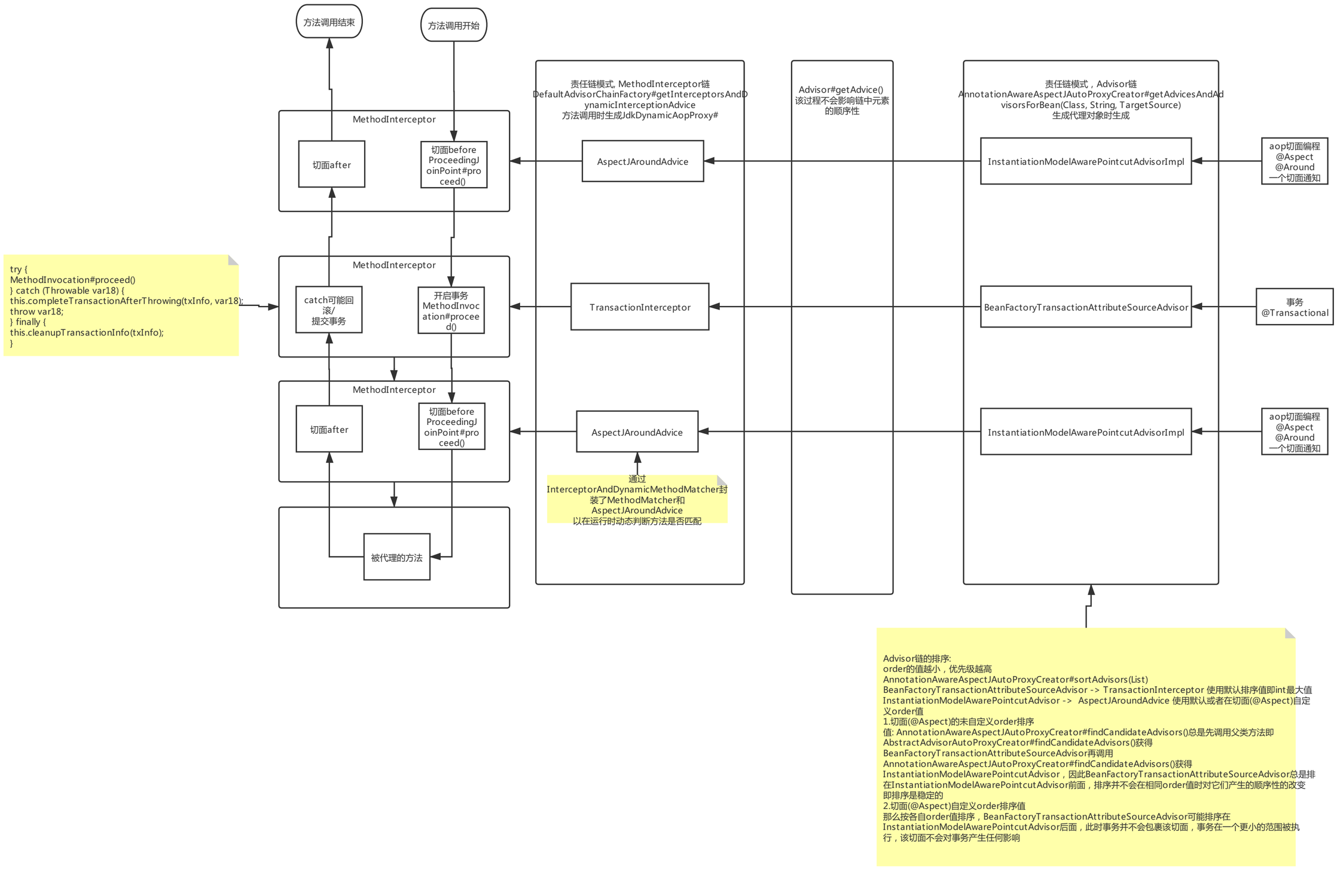
方法调用结束
方法调用开始
MethodInterceptor
切面after
切面before ProceedingJoinPoint#proceed()
MethodInterceptor
catch可能回滚/提交事务
开启事务 MethodInvocation#proceed()
MethodInterceptor
切面after
切面before ProceedingJoinPoint#proceed()
被代理的方法
try {
MethodInvocation#proceed()
} catch (Throwable var18) {
this.completeTransactionAfterThrowing(txInfo, var18);
throw var18;
} finally {
this.cleanupTransactionInfo(txInfo);
}
责任链模式, MethodInterceptor链
DefaultAdvisorChainFactory#getInterceptorsAndDynamicInterceptionAdvice
方法调用时生成JdkDynamicAopProxy#
AspectJAroundAdvice
TransactionInterceptor
AspectJAroundAdvice
通过InterceptorAndDynamicMethodMatcher封装了MethodMatcher和AspectJAroundAdvice以在运行时动态判断方法是否匹配
Advisor#getAdvice()
该过程不会影响链中元素的顺序性
责任链模式，Advisor链
AnnotationAwareAspectJAutoProxyCreator#getAdvicesAndAdvisorsForBean(Class, String, TargetSource)
生成代理对象时生成
InstantiationModelAwarePointcutAdvisorImpl
BeanFactoryTransactionAttributeSourceAdvisor
InstantiationModelAwarePointcutAdvisorImpl
aop切面编程
@Aspect
@Around
一个切面通知
事务
@Transactional
aop切面编程
@Aspect
@Around
一个切面通知
Advisor链的排序:
order的值越小，优先级越高
AnnotationAwareAspectJAutoProxyCreator#sortAdvisors(List)
BeanFactoryTransactionAttributeSourceAdvisor -> TransactionInterceptor 使用默认排序值即int最大值
InstantiationModelAwarePointcutAdvisor -> AspectJAroundAdvice 使用默认或者在切面(@Aspect)自定义order值
1.切面(@Aspect)的未自定义order排序
值: AnnotationAwareAspectJAutoProxyCreator#findCandidateAdvisors()总是先调用父类方法即AbstractAdvisorAutoProxyCreator#findCandidateAdvisors()获得
BeanFactoryTransactionAttributeSourceAdvisor再调用AnnotationAwareAspectJAutoProxyCreator#findCandidateAdvisors()获得
InstantiationModelAwarePointcutAdvisor，因此BeanFactoryTransactionAttributeSourceAdvisor总是排在InstantiationModelAwarePointcutAdvisor前面，排序并不会在相同order值时对它们产生的顺序性的改变即排序是稳定的
2.切面(@Aspect)自定义order排序值
那么按各自order值排序，BeanFactoryTransactionAttributeSourceAdvisor可能排序在InstantiationModelAwarePointcutAdvisor后面，此时事务并不会包裹该切面，事务在一个更小的范围被执行，该切面不会对事务产生任何影响