<https://baijiahao.baidu.com/s?id=1661565712893820457&wfr=spider&for=pc>

【1】

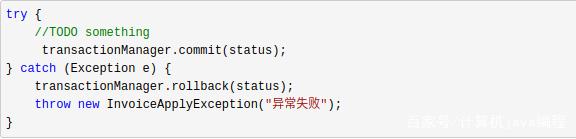
**引言**

昨天公众号粉丝咨询了一个问题，说自己之前面试被问@Transactional注解哪些场景下会失效，一时语塞致使面试失败。所以今天简单的和大家分享一下@Transactional相关的知识。@Transactional 注解相信大家并不陌生，平时开发中很常用的一个注解，它能保证方法内多个数据库操作要么同时成功、要么同时失败。使用@Transactional注解时需要注意许多的细节，不然你会发现@Transactional总是莫名其妙的就失效了。

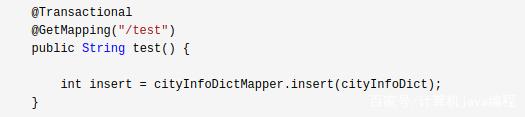
**一、事务**

事务管理在系统开发中是不可缺少的一部分，Spring提供了很好事务管理机制，主要分为编程式事务和声明式事务两种。

**编程式事务**：是指在代码中手动的管理事务的提交、回滚等操作，代码侵入性比较强，如下示例：



**声明式事务**：基于AOP面向切面的，它将具体业务与事务处理部分解耦，代码侵入性很低，所以在实际开发中声明式事务用的比较多。声明式事务也有两种实现方式，是基于TX和AOP的xml配置文件方式，二种就是基于@Transactional注解了。



**二、@Transactional介绍**

**1、@Transactional注解可以作用于哪些地方？**

@Transactional 可以作用在接口、类、类方法

**作用于类**：当把@Transactional 注解放在类上时，表示所有该类的public方法都配置相同的事务属性信息。**作用于方法**：当类配置了@Transactional，方法也配置了@Transactional，方法的事务会覆盖类的事务配置信息。**作用于接口**：不推荐这种使用方法，因为一旦标注在Interface上并且配置了Spring AOP 使用CGLib动态代理，将会导致@Transactional注解失效



**2、@Transactional注有哪些属性？**

**propagation属性**

propagation 代表事务的传播行为，默认值为 Propagation.REQUIRED，其他的属性信息如下：

Propagation.REQUIRED：如果当前存在事务，则加入该事务，如果当前不存在事务，则创建一个新的事务。**(** 也就是说如果A方法和B方法都添加了注解，在默认传播模式下，A方法内部调用B方法，会把两个方法的事务合并为一个事务 **）**Propagation.SUPPORTS：如果当前存在事务，则加入该事务；如果当前不存在事务，则以非事务的方式继续运行。Propagation.MANDATORY：如果当前存在事务，则加入该事务；如果当前不存在事务，则抛出异常。Propagation.REQUIRES\_NEW：重新创建一个新的事务，如果当前存在事务，暂停当前的事务。**(** 当类A中的 a 方法用默认Propagation.REQUIRED模式，类B中的 b方法加上采用 Propagation.REQUIRES\_NEW模式，然后在 a 方法中调用 b方法操作数据库，然而 a方法抛出异常后，b方法并没有进行回滚，因为Propagation.REQUIRES\_NEW会暂停 a方法的事务 **)**Propagation.NOT\_SUPPORTED：以非事务的方式运行，如果当前存在事务，暂停当前的事务。Propagation.NEVER：以非事务的方式运行，如果当前存在事务，则抛出异常。Propagation.NESTED ：和 Propagation.REQUIRED 效果一样。**isolation 属性**

isolation ：事务的隔离级别，默认值为

Isolation.DEFAULT

solation.DEFAULT：使用底层数据库默认的隔离级别。Isolation.READ\_UNCOMMITTEDIsolation.READ\_COMMITTEDIsolation.REPEATABLE\_READIsolation.SERIALIZABLE**timeout 属性**

timeout ：事务的超时时间，默认值为 -1。如果超过该时间限制但事务还没有完成，则自动回滚事务。

**readOnly 属性**

readOnly ：指定事务是否为只读事务，默认值为 false；为了忽略那些不需要事务的方法，比如读取数据，可以设置 read-only 为 true。

**rollbackFor 属性**

rollbackFor ：用于指定能够触发事务回滚的异常类型，可以指定多个异常类型。

**noRollbackFor属性\*\***

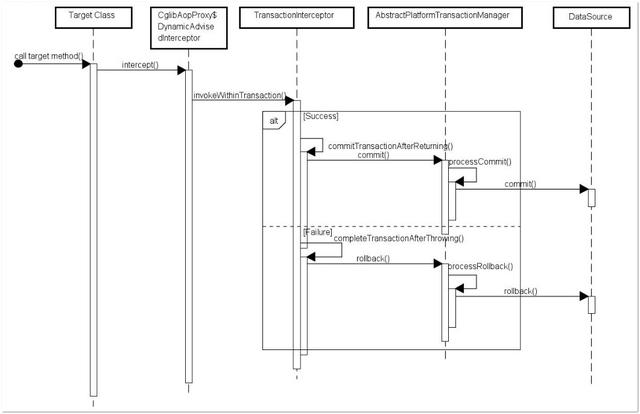
noRollbackFor：抛出指定的异常类型，不回滚事务，也可以指定多个异常类型。

**二、@Transactional失效场景**

接下来我们结合具体的代码分析一下哪些场景下，@Transactional 注解会失效。

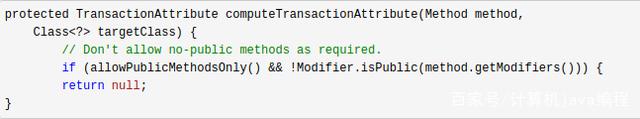
**1、@Transactional 应用在非 public 修饰的方法上**

如果Transactional注解应用在非public修饰的方法上，Transactional将会失效。



之所以会失效是因为在Spring AOP 代理时，如上图所示 TransactionInterceptor （事务拦截器）在目标方法执行前后进行拦截，DynamicAdvisedInterceptor（CglibAopProxy 的内部类）的 intercept 方法或 JdkDynamicAopProxy 的 invoke 方法会间接调用 AbstractFallbackTransactionAttributeSource的 computeTransactionAttribute

方法，获取Transactional 注解的事务配置信息。



此方法会检查目标方法的修饰符是否为 public，不是 public则不会获取@Transactional 的属性配置信息。

**注意：protected、**private**修饰的方法上使用**@Transactional**注解，虽然事务无效，但不会有任何报错，这是我们很容犯错的一点。**

**2、@Transactional 注解属性 propagation 设置错误**

这种失效是由于配置错误，若是错误的配置以下三种 propagation，事务将不会发生回滚。

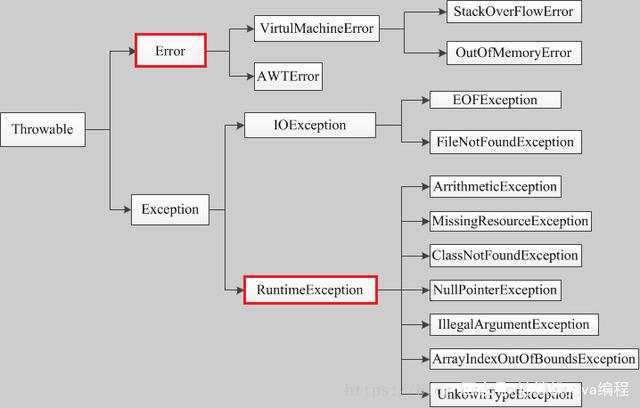
TransactionDefinition.PROPAGATION\_SUPPORTS：如果当前存在事务，则加入该事务；如果当前没有事务，则以非事务的方式继续运行。

TransactionDefinition.PROPAGATION\_NOT\_SUPPORTED：以非事务方式运行，如果当前存在事务，则把当前事务挂起。

TransactionDefinition.PROPAGATION\_NEVER：以非事务方式运行，如果当前存在事务，则抛出异常。

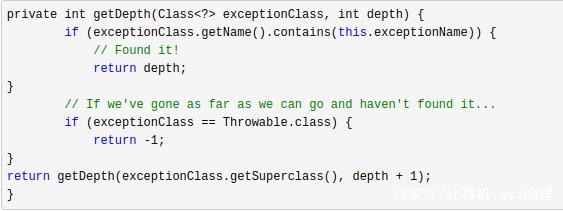
**3、@Transactional 注解属性 rollbackFor 设置错误**

rollbackFor 可以指定能够触发事务回滚的异常类型。Spring默认抛出了未检查unchecked异常（继承自 RuntimeException 的异常）或者 Error才回滚事务；其他异常不会触发回滚事务。如果在事务中抛出其他类型的异常，但却期望 Spring 能够回滚事务，就需要指定 **rollbackFor**属性。



// 希望自定义的异常可以进行回滚@Transactional(propagation= Propagation.REQUIRED,rollbackFor= MyException.class

若在目标方法中抛出的异常是 rollbackFor 指定的异常的子类，事务同样会回滚。Spring源码如下：



**4、同一个类中方法调用，导致@Transactional失效**

开发中避免不了会对同一个类里面的方法调用，比如有一个类Test，它的一个方法A，A再调用本类的方法B（不论方法B是用public还是private修饰），但方法A没有声明注解事务，而B方法有。则外部调用方法A之后，方法B的事务是不会起作用的。这也是经常犯错误的一个地方。

那为啥会出现这种情况？其实这还是由于使用Spring AOP代理造成的，因为只有当事务方法被当前类以外的代码调用时，才会由Spring生成的代理对象来管理。



**5、异常被你的 catch“吃了”导致@Transactional失效**

这种情况是最常见的一种@Transactional注解失效场景，



如果B方法内部抛了异常，而A方法此时try catch了B方法的异常，那这个事务还能正常回滚吗？

答案：不能！

会抛出异常：

org.springframework.transaction.UnexpectedRollbackException: Transaction rolled back because it has been marked as rollback-only

因为当ServiceB中抛出了一个异常以后，ServiceB标识当前事务需要rollback。但是ServiceA中由于你手动的捕获这个异常并进行处理，ServiceA认为当前事务应该正常commit。此时就出现了前后不一致，也就是因为这样，抛出了前面的UnexpectedRollbackException异常。

spring的事务是在调用业务方法之前开始的，业务方法执行完毕之后才执行commit or rollback，事务是否执行取决于是否抛出runtime异常。如果抛出runtime exception 并在你的业务方法中没有catch到的话，事务会回滚。

在业务方法中一般不需要catch异常，如果非要catch一定要抛出throw new RuntimeException()，或者注解中指定抛异常类型@Transactional(rollbackFor=Exception.class)，否则会导致事务失效，数据commit造成数据不一致，所以有些时候try catch反倒会画蛇添足。

**6、数据库引擎不支持事务**

这种情况出现的概率并不高，事务能否生效数据库引擎是否支持事务是关键。常用的MySQL数据库默认使用支持事务的innodb引擎。一旦数据库引擎切换成不支持事务的myisam，那事务就从根本上失效了。

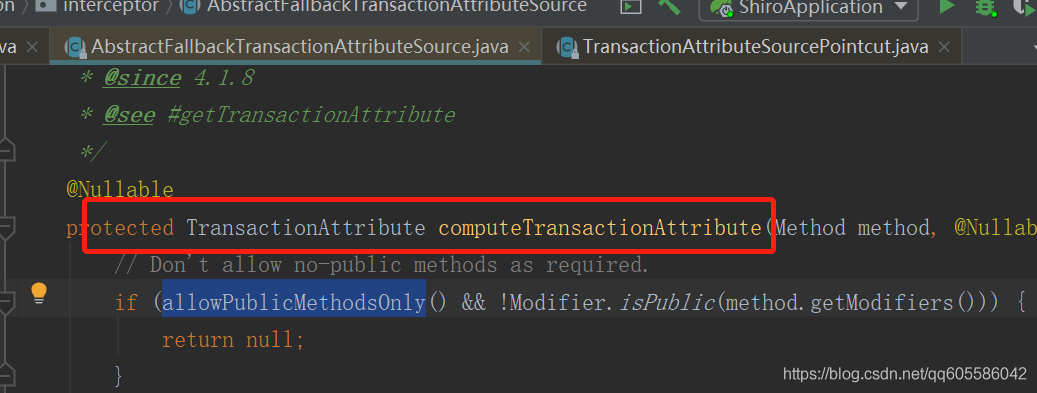
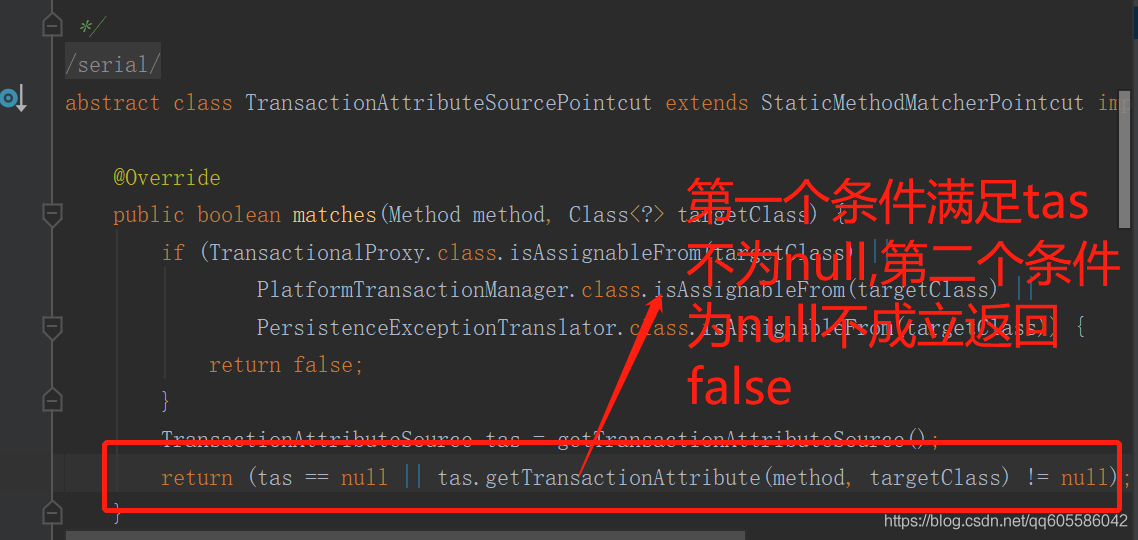
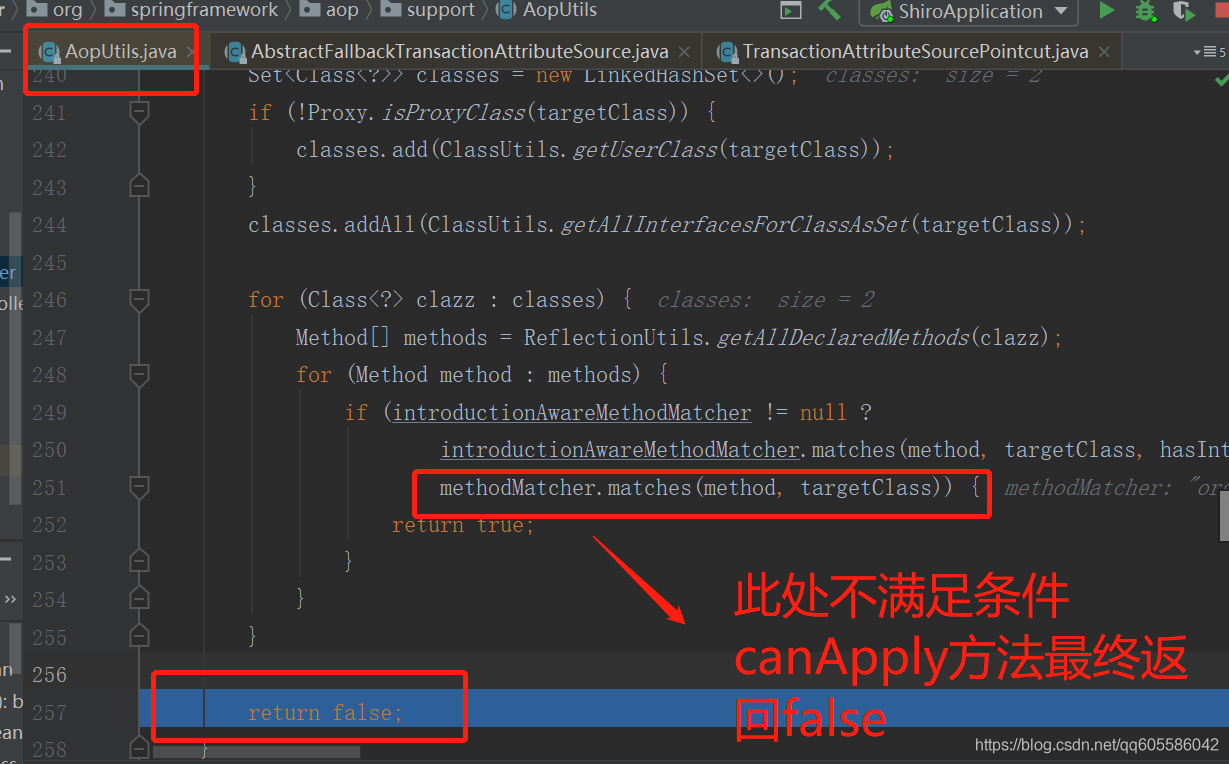
**总结**

@Transactional 注解的看似简单易用，但如果对它的用法一知半解，还是会踩到很多坑的。

【2】

<https://blog.csdn.net/qq605586042/article/details/104362969?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase>



AbstractFallbackTransactionAttributeSource中getTransactionAttribute方法调用了其本身的computeTransactionAttribute方法，如下图  
，其中allowPublicMethodsOnly方法由子类AnnotationTransactionAttributeSource实现，该子类方法中默认是true，所以当你加了事务注解的方法不是public时，该方法直接返回null  
  
接下来是AopUtils.canApply方法  
  
由于canApply方法返回false所以没有添加对应的事务Advisor