# **@Transactional 踩坑记录(不生效,并发,回滚问题)**

### **文章目录**

* [坑1: @Transactional 不生效?](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "1_Transactional__3" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [1. 是否添加依赖?](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "1__7" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [2. 方法是否是公开的( pubilc ) ?](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "2__pubilc___14" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [3. @Transactional 所属类被 spring 所管理? 类上是否包含 @Controller | @Service | @Component ....](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "3_Transactional__spring___Controller__Service__Component__24" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [4. @Transactional 有些异常没有回滚？ 注明 rollbackFor （阿里巴巴规范也要求）](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "4_Transactional___rollbackFor__39" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [5. 查看数据库或表，设置的引擎。MyISAM是不支持事务的，必须改为InnoDB](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "5_MyISAMInnoDB_53" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
* [坑2: 本类方法调用本类事务方法会导致事务不生效](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "2__59" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [问题源码](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "_61" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [解决方案1: 配置暴露aop代理类](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "1_aop_90" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [解决方案2: 利用ApplicationContext 获取实例[1]调用目标方法](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "2_ApplicationContext_1_105" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [解决方案3: 注入自身实例](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "3__121" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
* [坑3: Transactional 结合 try-finally 使用偶尔感觉 finally 方法块不执行?](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "3_Transactional__tryfinally__finally__147" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [问题源码](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "_149" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [解决方案1: 添加 catch 代码块](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "1__catch__164" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [解决方案2: 独立finally代理块, 开启新事物提交](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "2_finally__181" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
* [坑4: Transactional 结合 synchronized 使用仍存在并发问题](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "4_Transactional__synchronized__203" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [问题源码](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "_207" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [解决方案1: 数据库表设置限制](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "1__233" \t "https://blog.csdn.net/JinglongSource/article/details/_self)
  + [解决方案2: synchronized 作用域包含 proxy 处理事务](https://blog.csdn.net/JinglongSource/article/details/105026665" \l "2_synchronized__proxy__236" \t "https://blog.csdn.net/JinglongSource/article/details/_self)

该博文主要记录我在实际项目开发中遇到的 Transactional 坑，因为我遇到好几次了类似的不太相同坑了，所以周末决定还是花点时间整理一下，并且深入了解一下 Transactional 实现机制以及记录一下这几个坑。。。

# **坑1: @Transactional 不生效?**

解决思路如下:

## **1. 是否添加依赖?**

新项目经常会忘记添加各种依赖导致（Transactional依赖AOP实现，因此需要导入aop相关依赖）

compile 'org.springframework.boot:spring-boot-starter-aop'

## **2. 方法是否是公开的( pubilc ) ?**

@Transactionalpublic void test() {

*// 要求 test() 方法必须是 public 修饰, 如果是IDEA编辑器, 甚至直接警告了*

*// TODO*}

## **3. @Transactional 所属类被 spring 所管理? 类上是否包含 @Controller | @Service | @Component …**

@Service*// 要求所属类必须被 spring 容器所管理, 否则不生效那就很正常啦*public class TestService {

@Transactional

public void test() {

}

}

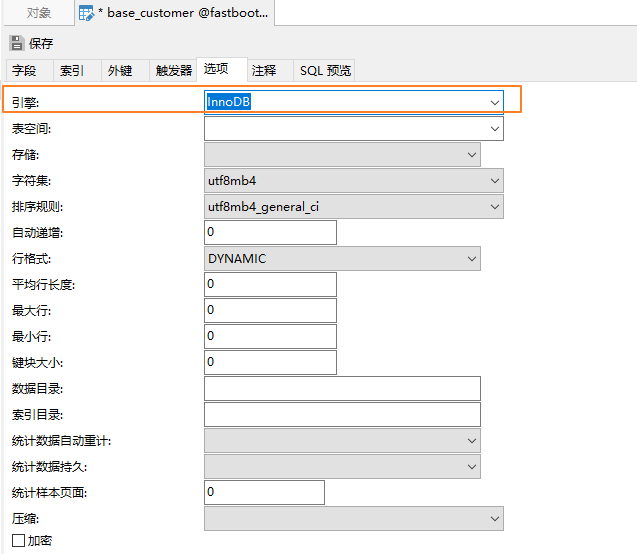
## **4. @Transactional 有些异常没有回滚？ 注明 rollbackFor （阿里巴巴规范也要求）**

可查的异常（checked exceptions）：Exception下除了RuntimeException外的异常  
不可查的异常（unchecked exceptions）：RuntimeException及其子类和错误（Error）

@Transactional 事务的回滚仅仅对于unchecked的异常有效。对于checked异常无效。也就是说事务回滚仅仅发生在出现RuntimeException或Error的时候。  
显示设置 rollbackFor 非常有必要，否则程序出现莫名异常可能会导致事务没有回滚生效

@Transactional(rollbackFor = Throwable.class)public void test() {}

## **5. 查看数据库或表，设置的引擎。MyISAM是不支持事务的，必须改为InnoDB**



# **坑2: 本类方法调用本类事务方法会导致事务不生效**

## **问题源码**

public void doBusiness() {

*// ...*

*//*

*// update sql*

this.update();

*// ...*}

@Transactional(rollbackFor = Throwable.class)public void update() {

*// do many update sql*

*// ...*

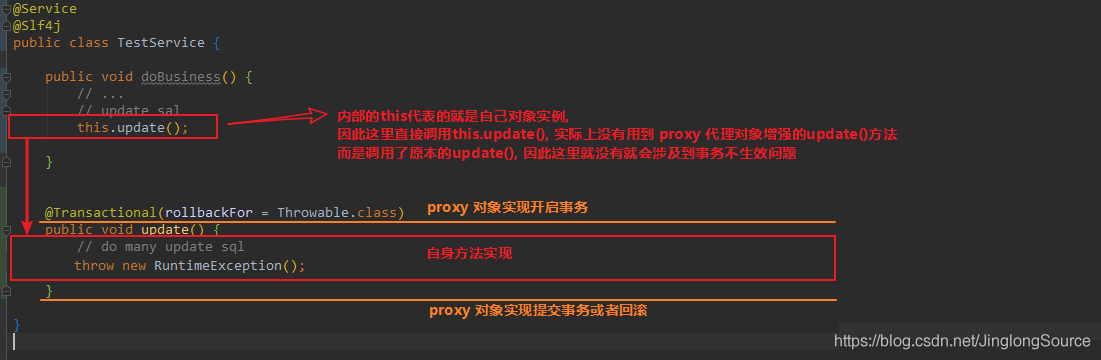
*// throw new RuntimeException();*

*// 这个时候 update() 并不会回滚*}

这里面涉及挺多知识点，在这里主要简单说一下原因就好了，其实也不能说是 @Transactional 的坑， 应该说是 aop 的坑， 因为所有的aop都会有这个问题，原因如下:

我们都知道aop(切面，方法增强)，至于它是怎么做到的？其实就涉及到设计模式的代理问题了，[了解更多设计模式之代理问题请看我的另一篇博客](https://blog.csdn.net/JinglongSource/article/details/103791964)

简单问题说明:



## **解决方案1: 配置暴露aop代理类**

1. 配置类添加配置 暴露代理类

@EnableAspectJAutoProxy(exposeProxy = true)

1. 使用 AopContext.currentProxy() 获取当前代理对象调用目标方法

*// this.update();*((TestService) AopContext.currentProxy()).update();

## **解决方案2: 利用ApplicationContext 获取实例[1]调用目标方法**

这里的实例[1]可以理解为代理对象, 但是又不是真正的代理对象

@Autowiredprivate ApplicationContext applicationContext;

public void doBusiness() {

*// ...*

*// update sql*

*// this.update();*

applicationContext.getBean(this.getClass()).update();

throw new RuntimeException();}

## **解决方案3: 注入自身实例**

@Service@Slf4jpublic class TestService {

@Autowired

private TestService testService; *// 注入自身实例*

public void doBusiness() {

*// ...*

*// update sql*

*// this.update();*

testService.update(); *// 利用注入实例调用目标方法*

}

@Transactional(rollbackFor = Throwable.class, propagation = Propagation.NESTED)

public void update() {

*// do many update sql*

}

}

# **坑3: Transactional 结合 try-finally 使用偶尔感觉 finally 方法块不执行?**

## **问题源码**

@Transactional(rollbackFor = Throwable.class)public void update() {

try {

*// 操作一些业务,*

} finally {

*// 业务最终都需要 删除一些临时数据, 因此确保代码执行, 放在finally代码块中*

}}

这个问题我是在实际开发中遇到的，当时排查了十几分钟大概知道问题所在，问题导致原因是因为 try-finally 中 try 可能会发生异常, 那就会抛出异常，因此就算执行到了finally代码块中, 执行了业务（删除一些临时数据），但是由于整个方法没有catch住异常，导致异常上抛，事务生效，给人假象就是 finally 没有执行一样

## **解决方案1: 添加 catch 代码块**

添加 catch 代码块捕获异常, 不让异常上抛

@Transactional(rollbackFor = Throwable.class)public void update() {

try {

*// 操作一些业务,*

} catch(Exception e) {

log.warn("update() ", e);

} finally {

*// 业务最终都需要 删除一些临时数据, 因此确保代码执行, 放在finally代码块中*

}}

## **解决方案2: 独立finally代理块, 开启新事物提交**

独立finally代理块, 开启新事物提交

@Transactional(rollbackFor = Throwable.class)public void update() {

try {

*// 操作一些业务*

} finally {

*// 业务最终都需要 删除一些临时数据, 因此确保代码执行, 再放finally代码块中*

applicationContext.getBean(this.getClass()).releaseData();

}}

@Transactional(rollbackFor = Throwable.class, propagation = Propagation.REQUIRES\_NEW)public void releaseData() {

*// do many update sql*}

# **坑4: Transactional 结合 synchronized 使用仍存在并发问题**

这里假设单实例环境，并且不考虑事务级别问题

## **问题源码**

@Transactional(rollbackFor = Throwable.class)public synchronized void doProcess(String id) {

*// 操作一些业务*

*// 根据ID查询 如果存在就更新, 否则新增*

if(this.getById(id) == null) {

*// 根据 id 新增一条记录*

*// ...*

*// 这里操作多张表 do many update sql*

} else {

*// 根据ID 更新某些字段*

*// ...*

*// 这里操作另外一些表 do many update sql*

}

}

产生问题：在并发情况下有可能会新增多条重复的记录  
会有这种情况：A线程根据ID查询了不存在，新增流程往下执行,出了synchronized 作用域准备提交事务的时候（还没提交），被B线程抢夺CPU执行权了，获得执行权的B线程携带ID刚好就是A线程的ID，因此B线程查询了也是不存在，执行新增流程,最终就会有多个新增记录



## **解决方案1: 数据库表设置限制**

数据库表设置唯一主键或者唯一索引进行限制，那么就会确保不会新增重复记录，如果重复记录就会抛出异常

## **解决方案2: synchronized 作用域包含 proxy 处理事务**

synchronized 作用域包含 proxy 处理事务即可， 如下：

public synchronized void doProcess(String id) {

*// 开始事务 在 synchronized 作用域*

applicationContext.getBean(this.getClass()).doBusiness(id);

*// 提交或者回滚 在 synchronized 作用域*}

@Transactional(rollbackFor = Throwable.class)public void doBusiness(String id) {

*// 操作一些业务*

*// 根据ID查询 如果存在就更新, 否则新增*

if(this.getById(id) == null) {

*// 根据 id 新增一条记录*

*// ...*

*// 这里操作多张表 do many update sql*

} else {

*// 根据ID 更新某些字段*

*// ...*

*// 这里操作另外一些表 do many update sql*

}}

追求卓越，然时间、经验、能力有限。写出来的博文有很多不足的地方，希望大家包容、不吝赐教，给我提意见、建议。感谢你们！

blog: https://shaines.cn  
mail : for.houyu@foxmail.com  
csdn: https://blog.csdn.net/jinglongsou