链滴

mybatis 手动回滚事务

作者：baozebing
原文链接：https://ld246.com/article/1587550650497
来源网站：链滴

@Transactional事物声明方式有两种

- 一种是在配置文件（xml）中做相关的事务规则声明，
- 另一种是基于@Transactional 注解的方式

在SpringBoot则非常简单，只需在业务层添加事务注解(@Transactional)即可快速开启事务（网上多文章说需要在启动类上添加注解@EnableTransactionManagement 开启事务, 本人实际开发中并需要添加，正确配置数据源后都是自动开启的）。虽然事务很简单，但对于数据方面是需要谨慎对待。

@Transactional注解用于两种场景：

- 标于类上：表示所有方法都进行事务处理
- 标于方法上：仅对该方法有效

@Transactional运行解读

在应用系统调用声明了 @Transactional 的目标方法时，Spring Framework 默认使用 AOP 代，在代码运行时生成一个代理对象，根据 @Transactional 的属性配置信息，这个代理对象决定该声明 @Transactional 的目标方法是否由拦截器 TransactionInterceptor 来使用拦截，在 TransactionInteceptor 拦截时，会在目标方法开始执行之前创建并加入事务，并执行目标方法的逻辑, 最后根据执行况是否出现异常，利用抽象事务管理器 AbstractPlatformTransactionManager 操作数据源 DataSouce 提交或回滚事务。

Spring AOP 代理有 CglibAopProxy 和 JdkDynamicAopProxy 两种，以 CglibAopProxy 为例对于 CglibAopProxy，需要调用其内部类的 DynamicAdvisedInterceptor 的 intercept 方法。对于dkDynamicAopProxy，需要调用其 invoke 方法。

事务传播行为

@Transactional(propagation=Propagation.REQUIRED)： 如果有事务, 那么加入事务, 没有的话新一个(默认情况下)

@Transactional(propagation=Propagation.NOT_SUPPORTED)： 容器不为这个方法开启事务

@Transactional(propagation=Propagation.REQUIRES_NEW)： 不管是否存在事务,都创建一个新事务,原来的挂起,新的执行完毕,继续执行老的事务

@Transactional(propagation=Propagation.MANDATORY)： 必须在一个已有的事务中执行,否则出异常

@Transactional(propagation=Propagation.NEVER)： 必须在一个没有的事务中执行,否则抛出异常与Propagation.MANDATORY相反)

@Transactional(propagation=Propagation.SUPPORTS)： 如果其他bean调用这个方法,在其他bea中声明事务,那就用事务.如果其他bean没有声明事务,那就不用事务.

事务超时设置

@Transactional(timeout=30) //默认是30秒

事务隔离级别

@Transactional(isolation = Isolation.READ_UNCOMMITTED)：读取未提交数据(会出现脏读, 不重复读) 基本不使用

@Transactional(isolation = Isolation.READ_COMMITTED)：读取已提交数据(会出现不可重复读幻读)

@Transactional(isolation = Isolation.REPEATABLE_READ)：可重复读(会出现幻读)

@Transactional(isolation = Isolation.SERIALIZABLE)：串行化

MYSQL: 默认为REPEATABLE_READ级别

SQLSERVER: 默认为READ_COMMITTED

ORACLE：默认为READ COMMITTED

脏读 : 一个事务读取到另一事务未提交的更新数据

不可重复读 : 在同一事务中, 多次读取同一数据返回的结果有所不同, 换句话说, 后续读取可以读到另事务已提交的更新数据. 相反, "可重复读"在同一事务中多次读取数据时, 能够保证所读数据一样, 也就后续读取不能读到另一事务已提交的更新数据

幻读 : 一个事务读到另一个事务已提交的insert数据

注意事项

1. @Transactional 只能被应用到public方法上, 对于其它非public的方法,如果标记了@Transactiona也不会报错,但方法没有事务功能。

2. 用 spring 事务管理器,由spring来负责数据库的打开,提交,回滚.默认遇到运行期例外(throw new RuntimeException("注释");)会回滚，即遇到不受检查（unchecked）的例外时回滚；而遇到需要捕获的外(throw new Exception("注释");)不会回滚,即遇到受检查的例外（就是非运行时抛出的异常，编译会检查到的异常叫受检查例外或说受检查异常）时，需我们指定方式来让事务回滚要想所有异常都回滚要加上 @Transactional(rollbackFor={Exception.class,其它异常}) .如果让unchecked例外不回滚：@Transactional(notRollbackFor=RunTimeException.class)

注意：异常的超类是Throwable，两个分支分别是运行时异常Error和Exception。Exception又分为行时异常和受检查异常。

3. @Transactional 注解应该只被应用到 public 可见度的方法上。 如果你在 protected、private 或 package-visible 的方法上使用 @Transactional 注解，它也不会报错， 但是这个被注解的方法将不展示已配置的事务设置。

4. @Transactional 注解可以被应用于接口定义和接口方法、类定义和类的 public 方法上。然而，请意仅仅 @Transactional 注解的出现不足于开启事务行为，它仅仅 是一种元数据，能够被可以识别 @ransactional 注解和上述的配置适当的具有事务行为的beans所使用。

5. Spring团队的建议是你在具体的类（或类的方法）上使用 @Transactional 注解，而不要使用在类要实现的任何接口上。你当然可以在接口上使用 @Transactional 注解，但是这将只能当你设置了基接口的代理时它才生效。因为注解是不能继承的，这就意味着如果你正在使用基于类的代理时，那么务的设置将不能被基于类的代理所识别，而且对象也将不会被事务代理所包装（将被确认为严重的）因此，请接受Spring团队的建议并且在具体的类上使用 @Transactional 注解。

6. 这一点是最常见的，就是既添加了@Transactional 注解，方法里却又添加了catch语句对业务进行常捕获，你都把异常“吃”掉了，Spring自然不知道这里有错，更不会主动去回滚数据。

代码示例

```
@Transactional(rollbackFor = Exception.class)
@Override
```

```
public void saveEntity() throws Exception{
    try {
        userDao.saveUser();
        studentDao.saveStudent();
    }catch (Exception e){
        System.out.println("异常了=====" + e);
        //手动强制回滚事务，这里一定要第一时间处理
        TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();
    }
}
```

摘1 摘2