1、面试题

spring的事务支持（注解事务、声明事务、编程事务、事务的传播机制）？执行某个操作，前50次成功，第51次失败。a 全部回滚；b 前50次提交，第51次抛异常。ab场景分别如何设置spring事务。

2、面试官心里分析

聊完上面那个问题，面试官估计心里对你感觉相当不错了，但是呢，事儿没玩，还得聊聊实际项目中，你的java系统里的事务咋玩儿的啊？这就涉及到了spring对事务的支持，然后重要的事务传播机制！

3、面试题剖析

这个，你一般就聊下，spring支持编程式事务，和声明式事务。编程式事务就是用个事务类TransactionTemplate来管理事务，这个一般现在没人傻到干这个事儿了；声明式事务分成在xml里配置个AOP来声明个切面加事务，一般现在也没人傻到干这个了；大部分情况下，都是用@Transactional注解。

这个@Transactional注解呢，根据阿里编码规范，一般建议加在方法级别，就是要事务的方法就加事务，不要事务的方法就别加事务，否则很多一个数据库操作的方法，你还加事务，你这不是。。。？可能有同学注意到我们项目阶段一，都是傻乎乎的在类上加了事务对吧？

对，其实不是我傻，是我装傻，我当时为了快速开发，所以都给扔类上了，类里所有方法都开启了事务，但是后续我们会慢慢优化这些细节的。

另外这个注解一般要加rollbackFor，就是指定哪些异常类型才要回滚事务

还有比较重要的，就是有个isolation属性，你可以自己手动调整事务的隔离级别，但是这个一般不调整，记住，别乱调整事务隔离级别，一般可重复读+mysql mvcc机制跑的很好，你别瞎折腾。

另外一个重要的事务属性，就是propagation，事务的传播行为，我们就重点先来聊一下事务的传播行为，这个在项目里可能确实是要用到的。其实说白了，这个事务的传播机制，就是说，一个加了@Transactional的事务方法，和嵌套了另外一个@Transactional的事务方法的时候，包括再次嵌入@Transactional事务方法的时候，这个事务怎么玩儿？

我们的项目阶段一里是不是有大量这种场景？呵呵，所以啊，真实复杂业务系统的好处就在这里，真好，后面我会慢慢优化这些细节，包括事务的传播机制等等，让你各种技术都在复杂业务系统里看看怎么玩儿的。

public class ServiceA {

@Autowired

private ServiceB b;

@Transactional

public void methodA() {

// 一坨数据库操作

for(int i = 0; i < 51; i++) {

try {

b.methodB();

} catch(Exception e) {

// 打印异常日志

}

}

// 一坨数据库操作

}

}

public class ServiceB {

@Transactional(propagation = PROPAGATION\_REQUIRES\_NEW)

public void methodB() throws Exception {

// 一坨数据库操作

}

}

项目阶段一里面，是不是有好多OrderService调用了MembershipService调用了WmsService

一共有7种事务传播行为：

（1）PROPAGATION\_REQUIRED：这个是最常见的，就是说，如果ServiceA.method调用了ServiceB.method，如果ServiceA.method开启了事务，然后ServiceB.method也声明了事务，那么ServiceB.method不会开启独立事务，而是将自己的操作放在ServiceA.method的事务中来执行，ServiceA和ServiceB任何一个报错都会导致整个事务回滚。这就是默认的行为，其实一般我们都是需要这样子的。

（2）PROPAGATION\_SUPPORTS：如果ServiceA.method开了事务，那么ServiceB就将自己加入ServiceA中来运行，如果ServiceA.method没有开事务，那么ServiceB自己也不开事务

（3）PROPAGATION\_MANDATORY：必须被一个开启了事务的方法来调用自己，否则报错

（4）PROPAGATION\_REQUIRES\_NEW：ServiceB.method强制性自己开启一个新的事务，然后ServiceA.method的事务会卡住，等ServiceB事务完了自己再继续。这就是影响的回滚了，如果ServiceA报错了，ServiceB是不会受到影响的，ServiceB报错了，ServiceA也可以选择性的回滚或者是提交。

（5）PROPAGATION\_NOT\_SUPPORTED：就是ServiceB.method不支持事务，ServiceA的事务执行到ServiceB那儿，就挂起来了，ServiceB用非事务方式运行结束，ServiceA事务再继续运行。这个好处就是ServiceB代码报错不会让ServiceA回滚。

（6）PROPAGATION\_NEVER：不能被一个事务来调用，ServiceA.method开事务了，但是调用了ServiceB会报错

（7）PROPAGATION\_NESTED：开启嵌套事务，ServiceB开启一个子事务，如果回滚的话，那么ServiceB就回滚到开启子事务的这个save point。

大家回头想想那个面试题，其实就是ServiceA里循环51调用ServiceB，第51次调用ServiceB失败了。第一个选项，就是两个事务都设置为PROPAGATION\_REQUIRED就好了，ServiceB的所有操作都加入了ServiceA启动的一个大事务里去，任何一次失败都会导致整个事务的回滚；第二个选项，就是将ServiceB设置为PROPAGATION\_REQUIRES\_NEW，这样ServiceB的每次调用都在一个独立的事务里执行，这样的话，即使第51次报错，但是仅仅只是回滚第51次的操作，前面50次都在独立的事务里成功了，是不会回滚的。

其实一般也就PROPAGATION\_REQUIRES\_NEW比较常用，要的效果就是嵌套的那个事务是独立的事务，自己提交或者回滚，不影响外面的大事务，外面的大事务可以获取抛出的异常，自己决定是继续提交大事务还是回滚大事务。

一般在单块系统开发，多人协作的时候比较常见，就是小A调用小B的模块，小A不管小B是成功还是不成功，自己都要提交，这个时候可以这么弄，就是说小B的操作不是构成小A的事务的重要组成部分，就是个分支。