【1】【面试】Spring事务面试考点吐血整理（建议珍藏）

<https://www.cnblogs.com/lixinjie/p/spring-tx-key-point-in-a-interview.html>

## Spring和事务的关系

关系型数据库、某些消息队列等产品或中间件称为事务性资源，因为它们本身支持事务，也能够处理事务。

Spring很显然不是事务性资源，但是它可以管理事务性资源，所以Spring和事务之间是管理关系。

就像Jack Ma虽然不会写代码，但是他却管理者一大批会写代码的码农。

## Spring事务三要素

数据源：表示具体的事务性资源，是事务的真正处理者，如MySQL等。

事务管理器：像一个大管家，从整体上管理事务的处理过程，如打开、提交、回滚等。

事务应用和属性配置：像一个标识符，表明哪些方法要参与事务，如何参与事务，以及一些相关属性如隔离级别、超时时间等。

## Spring事务的注解配置

把一个DataSource（如DruidDataSource）作为一个@Bean注册到Spring容器中，配置好事务性资源。

把一个@EnableTransactionManagement注解放到一个@Configuration类上，配置好事务管理器，并启用事务管理。

把一个@Transactional注解放到类上或方法上，可以设置注解的属性，表明该方法按配置好的属性参与到事务中。

## 事务注解的本质

@Transactional这个注解仅仅是一些（和事务相关的）元数据，在运行时被事务基础设施读取消费，并使用这些元数据来配置bean的事务行为。

大致来说具有两方面功能，一是表明该方法要参与事务，二是配置相关属性来定制事务的参与方式和运行行为。

## Spring声明式事务实现原理

声明式事务成为可能，主要得益于Spring AOP。使用一个事务拦截器，在方法调用的前后/周围进行事务性增强（advice），来驱动事务完成。

## 如何回滚一个事务

就是在一个事务上下文中当前正在执行的代码里抛出一个异常，事务基础设施代码会捕获任何未处理的异常，并且做出决定是否标记这个事务为回滚。

## 默认回滚规则

默认只把runtime, unchecked exceptions标记为回滚，即RuntimeException及其子类，Error默认也导致回滚。Checked exceptions默认不导致回滚。这些规则和EJB是一样的。

## 如何配置回滚异常

使用@Transactional注解的rollbackFor/rollbackForClassName属性，可以精确配置导致回滚的异常类型，包括checked exceptions。

noRollbackFor/noRollbackForClassName属性，可以配置不导致回滚的异常类型，当遇到这样的未处理异常时，照样提交相关事务。

## 事务注解在类/方法上

@Transactional注解既可以标注在类上，也可以标注在方法上。当在类上时，默认应用到类里的所有方法。如果此时方法上也标注了，则方法上的优先级高。

## 事务注解在类上的继承性

@Transactional注解的作用可以传播到子类，即如果父类标了子类就不用标了。但倒过来就不行了。

子类标了，并不会传到父类，所以父类方法不会有事务。父类方法需要在子类中重新声明而参与到子类上的注解，这样才会有事务。

## 事务注解在接口/类上

@Transactional注解可以用在接口上，也可以在类上。在接口上时，必须使用基于接口的代理才行，即JDK动态代理。

事实是Java的注解不能从接口继承，如果你使用基于类的代理，即CGLIB，或基于织入方面，即AspectJ，事务设置不会被代理和织入基础设施认出来，目标对象不会被包装到一个事务代理中。

Spring团队建议注解标注在类上而非接口上。

## 只在public方法上生效？

当采用代理来实现事务时，（注意是代理），@Transactional注解只能应用在public方法上。当标记在protected、private、package-visible方法上时，不会产生错误，但也不会表现出为它指定的事务配置。可以认为它作为一个普通的方法参与到一个public方法的事务中。

如果想在非public方法上生效，考虑使用AspectJ（织入方式）。

## 目标类里的自我调用没有事务？

在代理模式中（这是默认的），只有从外部的方法调用进入通过代理会被拦截，这意味着自我调用（实际就是，目标对象中的一个方法调用目标对象的另一个方法）在运行时不会导致一个实际的事务，即使被调用的方法标有注解。

如果你希望自我调用也使用事务来包装，考虑使用AspectJ的方式。在这种情况下，首先是没有代理。相反，目标类被织入（即它的字节码被修改）来把@Transactional加入到运行时行为，在任何种类的方法上都可以。

## 事务与线程

和JavaEE事务上下文一样，Spring事务和一个线程的执行相关联，底层是一个ThreadLocal<Map<Object, Object>>，就是每个线程一个map，key是DataSource，value是Connection。

## 逻辑事务与物理事务

事务性资源实际打开的事务就是物理事务，如数据库的Connection打开的事务。Spring会为每个@Transactional方法创建一个事务范围，可以理解为是逻辑事务。

在逻辑事务中，大范围的事务称为外围事务，小范围的事务称为内部事务，外围事务可以包含内部事务，但在逻辑上是互相独立的。每一个这样的逻辑事务范围，都能够单独地决定rollback-only状态。

那么如何处理逻辑事务和物理事务之间的关联关系呢，这就是传播特性解决的问题。

## \*\*事务的传播特性

REQUIRED，SUPPORTS，MANDATORY，REQUIRES\_NEW，NOT\_SUPPORTED，NEVER，NESTED

### REQUIRED

强制要求要有一个物理事务。如果没有已经存在的事务，就专门打开一个事务用于当前范围。或者参与到一个已存在的更大范围的外围事务中。在相同的线程中，这是一种很好的默认方式安排。（例如，一个service外观/门面代理到若干个仓储方法，所有底层资源必须参与到service级别的事务里）

在标准的REQUIRED行为情况下，所有这样的逻辑事务范围映射到同一个物理事务。因此，在内部事务范围设置了rollback-only标记，确实会影响外围事务进行实际提交的机会。

注：默认，一个参与到外围事务的事务，会使用外围事务的特性，安静地忽略掉自己的隔离级别，超时值，只读标识等设置。当然可以在事务管理器上设置validateExistingTransactions标识为true，这样当你自己的事务和参与到的外围事务设置不一样时会被拒绝。

### REQUIRES\_NEW

与REQUIRED相比，总是使用一个独立的物理事务用于每一个受影响的逻辑事务范围，从来不参与到一个已存在的外围事务范围。这样安排的话，底层的事务资源是不同的，因此，可以独立地提交或回滚。外围事务不会被内部事务的回滚状态影响。这样一个独立的内部事务可以声明自己的隔离级别，超时时间和只读设置，并不继承外围事务的特性。

### NESTED

使用同一个物理事务，带有多个保存点，可以回滚到这些保存点，可以认为是部分回滚，这样一个内部事务范围触发了一个回滚，外围事务能够继续这个物理事务，尽管有一些操作已经被回滚。典型地，它对应于JDBC的保存点，所以只对JDBC事务资源起作用。

### SUPPORTS

支持当前事务。如果当前有事务，就参与进来，如果没有，就以非事务的方式运行。这样的一个逻辑事务范围，它背后可能没有实际的物理事务，此时的事务也成为空事务。

### NOT\_SUPPORTED

不支持当前事务。总是以非事务方式运行。当前的事务会被挂起，并在适合的时候恢复。

### MANDATORY

支持当前事务。如果当前没有事务存在，就抛出异常。

### NEVER

不支持当前事务。如果当前有事务存在，就抛出异常。

## 事务的隔离级别

DEFAULT，READ\_UNCOMMITTED，READ\_COMMITTED，REPEATABLE\_READ，SERIALIZABLE

### 脏读

一个事务修改了一行数据但没有提交，第二个事务可以读取到这行被修改的数据，如果第一个事务回滚，第二个事务获取到的数据将是无效的。

### 不可重复读

一个事务读取了一行数据，第二个事务修改了这行数据，第一个事务重新读取这行数据，将获得到不同的值。

### 幻读

一个事务按照一个where条件读取所有符合的数据行，第二个事务插入了一行数据且恰好也满足这个where条件，第一个事务再以这个where条件重新读取，将会获取额外多出来的这一行。

帮助记忆：

写读是脏读，读写读是不可重复读，where insert where是幻读。

### DEFAULT

使用底层数据存储的默认隔离级别。MySQL的默认隔离级别是REPEATABLE-READ。

### READ\_UNCOMMITTED

读未提交。脏读、不可重复读、幻读都会发生。

### READ\_COMMITTED

读已提交。脏读不会发生，不可重复读、幻读都会发生。

### REPEATABLE\_READ

可重复读。脏读、不可重复读都不会发生，幻读会发生。

### SERIALIZABLE

可串行化。脏读、不可重复读、幻读都不会发生。

【2】足够应付面试的Spring事务源码阅读梳理（建议珍藏）

<https://www.cnblogs.com/lixinjie/p/a-enough-source-read-of-spring-tx-for-interview.html>

<https://blog.csdn.net/hustspy1990/article/details/83758742?utm_medium=distribute.pc_relevant.none-task-blog-title-1&spm=1001.2101.3001.4242>

<https://blog.csdn.net/weixin_44366439/article/details/89030080>