Spring事务的传播机制与隔离级别

题目难度：★★★

知识点标签：Spring事务的传播机制与隔离级别

学习时长：30分钟

题目描述

描述一下Spring事务的传播机制与隔离级别

解题思路

> 面试官问题可以从几个方面来回答：说清楚Spring事务，传播机制与隔离级别，并能描述清楚，分布式事务

Spring事务

事务是逻辑处理原子性的保证手段，通过使用事务控制，可以极大的避免出现逻辑处理失败导致的脏数据等问题。

事务最重要的两个特性，是事务的传播级别和数据隔离级别。传播级别定义的是事务的控制范围，事务隔离级别定义的是事务在数据库读写方面的控制范围。

Spring事务传播机制

1） **PROPAGATION_REQUIRED**，Spring默认的事务传播级别，使用该级别的特点是，如果上下文中已经存在事务，那么就加入到事务中执行，如果当前上下文中不存在事务，则新建事务执行。所以这个级别通常能满足处理大多数的业务场景。

2）**PROPAGATION_SUPPORTS**，从字面意思就知道，supports，支持，该传播级别的特点是，如果上下文存在事务，则支持事务加入事务，如果没有事务，则使用非事务的方式执行。所以说，并非所有的包在transactionTemplate.execute中的代码都会有事务支持。这个通常是用来处理那些并非原子性的非核心业务逻辑操作。应用场景较少。

3）**PROPAGATION_MANDATORY**， 该级别的事务要求上下文中必须要存在事务，否则就会抛出异常！配置该方式的传播级别是有效的控制上下文调用代码遗漏添加事务控制的保证手段。比如一段代码不能单独被调用执行，但是一旦被调用，就必须有事务包含的情况，就可以使用这个传播级别。

4）**PROPAGATION_REQUIRES_NEW**，从字面即可知道，new，每次都要一个新事务，该传播级别的特点是，每次都会新建一个事务，并且同时将上下文中的事务挂起，执行当前新建事务完成以后，上下文事务恢复再执行。

这是一个很有用的传播级别，举一个应用场景：现在有一个发送100个红包的操作，在发送之前，要做一些系统的初始化、验证、数据记录操作，然后发送100封红包，然后再记录发送日志，发送日志要求100%的准确，如果日志不准确，那么整个父事务逻辑需要回滚。
怎么处理整个业务需求呢？就是通过这个PROPAGATION_REQUIRES_NEW 级别的事务传播控制就可以完成。发送红包的子事务不会直接影响到父事务的提交和回滚。

5）**PROPAGATION_NOT_SUPPORTED**，这个也可以从字面得知，not supported，不支持，当前级别的特点就是上下文中存在事务，则挂起事务，执行当前逻辑，结束后恢复上下文的事务。

这个级别有什么好处？可以帮助你将事务极可能的缩小。我们知道一个事务越大，它存在的风险也就越多。所以在处理事务的过程中，要保证尽可能的缩小范围。比如一段代码，是每次逻辑操作都必须调用的，比如循环1000次的某个非核心业务逻辑操作。这样的代码如果包在事务中，势必造成事务太大，导致出现一些难以考虑周全的异常情况。所以这个事务这个级别的传播级别就派上用场了。用当前级别的事务模板抱起来就可以了。

6）**PROPAGATION_NEVER**，该事务更严格，上面一个事务传播级别只是不支持而已，有事务就挂起，而PROPAGATION_NEVER传播级别要求上下文中不能存在事务，一旦有事务，就抛出runtime异常，强制停止执行！这个级别上辈子跟事务有仇。

7）**PROPAGATION_NESTED**，字面也可知道，nested，嵌套级别事务。该传播级别特征是，如果上下文中存在事务，则嵌套事务执行，如果不存在事务，则新建事务

Spring事务隔离级别

1、**Serializable**： 最严格的级别，事务串行执行，资源消耗最大；

2、**REPEATABLE READ**： 保证了一个事务不会修改已经由另一个事务读取但未提交（回滚）的数据。避免了“脏读取”和“不可重复读取”的情况，但是带来了更多的性能损失。

3、**READ COMMITTED** :大多数主流数据库的默认事务等级，保证了一个事务不会读到另一个并行事务已修改但未提交的数据，避免了“脏读取”。该级别适用于大多数系统。

4、**Read Uncommitted**： 保证了读取过程中不会读取到非法数据。

上面的解释其实每个定义都有一些拗口，其中涉及到几个术语：脏读、不可重复读、幻读。
这里解释一下：

脏读 :所谓的脏读，其实就是读到了别的事务回滚前的脏数据。比如事务B执行过程中修改了数据X，在未提交前，事务A读取了X，而事务B却回滚了，这样事务A就形成了脏读。

不可重复读： 不可重复读字面含义已经很明了了，比如事务A首先读取了一条数据，然后执行逻辑的时候，事务B将这条数据改变了，然后事务A再次读取的时候，发现数据不匹配了，就是所谓的不可重复读了。

幻读： 小的时候数手指，第一次数十10个，第二次数是11个，怎么回事？产生幻觉了？
幻读也是这样子，事务A首先根据条件索引得到10条数据，然后事务B改变了数据库一条数据，导致也符合事务A当时的搜索条件，这样事务A再次搜索发现有11条数据了，就产生了幻读。

分布式事务

在分布式的高并发环境下，对于核心业务逻辑的检查，要采用加锁机制。
比如事务开启需要读取一条数据进行验证，然后逻辑操作中需要对这条数据进行修改，最后提交。
这样的一个过程，如果读取并验证的代码放到事务之外，那么读取的数据极有可能已经被其他的事务修改，当前事务一旦提交，又会重新覆盖掉其他事务的数据，导致数据异常。
所以在进入当前事务的时候，必须要将这条数据锁住，使用for update就是一个很好的在分布式环境下的控制手段。

一种好的实践方式是使用编程式事务而非生命式，尤其是在较为规模的项目中。对于事务的配置，在代码量非常大的情况下，将是一种折磨，而且人肉的方式，绝对不能避免这种问题。
将DAO保持针对一张表的最基本操作，然后业务逻辑的处理放入manager和service中进行，同时使用编程式事务更精确的控制事务范围。
特别注意的，对于事务内部一些可能抛出异常的情况，捕获要谨慎，不能随便的catch Exception 导致事务的异常被吃掉而不能正常回滚。