[**spring事务传播属性和隔离级别**](http://www.cnblogs.com/jimmy-muyuan/p/5722708.html)

 1 事务的传播属性（Propagation）

1) REQUIRED ，这个是默认的属性   
Support a current transaction, create a new one if none exists.   
如果存在一个事务，则支持当前事务。如果没有事务则开启一个新的事务。   
被设置成这个级别时，会为每一个被调用的方法创建一个逻辑事务域。如果前面的方法已经创建了事务，那么后面的方法支持当前的事务，如果当前没有事务会重新建立事务。   
如图所示：   
  
2) MANDATORY   
Support a current transaction, throw an exception if none exists.支持当前事务，如果当前没有事务，就抛出异常。   
  
3) NEVER   
Execute non-transactionally, throw an exception if a transaction exists.   
以非事务方式执行，如果当前存在事务，则抛出异常。   
  
4) NOT\_SUPPORTED   
Execute non-transactionally, suspend the current transaction if one exists.   
以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。   
  
5) REQUIRES\_NEW   
Create a new transaction, suspend the current transaction if one exists.   
新建事务，如果当前存在事务，把当前事务挂起。   
如图所示：   
  
6) SUPPORTS   
Support a current transaction, execute non-transactionally if none exists.   
支持当前事务，如果当前没有事务，就以非事务方式执行。   
  
7) NESTED   
Execute within a nested transaction if a current transaction exists, behave like PROPAGATION\_REQUIRED else.   
支持当前事务，新增Savepoint点，与当前事务同步提交或回滚。   
嵌套事务一个非常重要的概念就是内层事务依赖于外层事务。外层事务失败时，会回滚内层事务所做的动作。而内层事务操作失败并不会引起外层事务的回滚。   
  
8) PROPAGATION\_NESTED 与PROPAGATION\_REQUIRES\_NEW的区别   
它们非常 类似,都像一个嵌套事务，如果不存在一个活动的事务，都会开启一个新的事务。使用PROPAGATION\_REQUIRES\_NEW时，内层事务与外层事务就像两个独立的事务一样，一旦内层事务进行了提交后，外层事务不能对其进行回滚。两个事务互不影响。两个事务不是一个真正的嵌套事务。同时它需要JTA 事务管理器的支持。   
使用PROPAGATION\_NESTED时，外层事务的回滚可以引起内层事务的回滚。而内层事务的异常并不会导致外层事务的回滚，它是一个真正的嵌套事务。   
  
2 事务的隔离级别（Isolation Level）   
  
1) 首先说明一下事务并发引起的三种情况   
  
i. Dirty Reads 脏读   
一个事务正在对数据进行更新操作，但是更新还未提交，另一个事务这时也来操作这组数据，并且读取了前一个事务还未提交的数据，而前一个事务如果操作失败进行了回滚，后一个事务读取的就是错误数据，这样就造成了脏读。

ii. Non-Repeatable Reads 不可重复读   
一个事务多次读取同一数据，在该事务还未结束时，另一个事务也对该数据进行了操作，而且在第一个事务两次次读取之间，第二个事务对数据进行了更新，那么第一个事务前后两次读取到的数据是不同的，这样就造成了不可重复读。

iii. Phantom Reads 幻像读   
第一个数据正在查询符合某一条件的数据，这时，另一个事务又插入了一条符合条件的数据，第一个事务在第二次查询符合同一条件的数据时，发现多了一条前一次查询时没有的数据，仿佛幻觉一样，这就是幻像读。

iv. 非重复度和幻像读的区别   
非重复读是指同一查询在同一事务中多次进行，由于其他提交事务所做的修改或删除，每次返回不同的结果集，此时发生非重复读。(A transaction rereads data it has previously read and finds that another committed transaction has modified or deleted the data. )

幻像读是指同一查询在同一事务中多次进行，由于其他提交事务所做的插入操作，每次返回不同的结果集，此时发生幻像读。(A transaction reexecutes a query returning a set of rows that satisfies a search condition and finds that another committed transaction has inserted additional rows that satisfy the condition. )

表面上看，区别就在于非重复读能看见其他事务提交的修改和删除，而幻像能看见其他事务提交的插入。   
  
2) DEFAULT （默认）   
这是一个PlatfromTransactionManager默认的隔离级别，使用数据库默认的事务隔离级别.另外四个与JDBC的隔离级别相对应   
  
3) READ\_UNCOMMITTED （读未提交）   
这是事务最低的隔离级别，它允许另外一个事务可以看到这个事务未提交的数据。这种隔离级别会产生脏读，不可重复读和幻像读。   
  
4) READ\_COMMITTED （读已提交）   
保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据。这种事务隔离级别可以避免脏读出现，但是可能会出现不可重复读和幻像读。   
  
5) REPEATABLE\_READ （可重复读）   
这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻像读。它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了不可重复读   
  
6) SERIALIZABLE（串行化）   
这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。除了防止脏读，不可重复读外，还避免了幻像读。   
  
7) 隔离级别解决事务并行引起的问题   
Dirty reads non-repeatable reads phantom reads   
Serializable 不会 不会 不会   
REPEATABLE READ 不会 不会 会   
READ COMMITTED 不会 会 会   
Read Uncommitted 会 会 会

**事物传播行为介绍:**   
@Transactional(propagation=Propagation.REQUIRED)   
如果有事务, 那么加入事务, 没有的话新建一个(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,都创建一个新的事务,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,否则抛出异常  
@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bean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.

**事物超时设置:**@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, 不可重复读) 基本不使用  
@Transactional(isolation = Isolation.READ\_COMMITTED)  
读取已提交数据(会出现不可重复读和幻读)  
@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

**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一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, 也就是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

 关于嵌套事物  
  
  可能大家对PROPAGATION\_NESTED还不怎么了解，觉得有必要再补充一下^\_^！  
PROPAGATION\_NESTED: 嵌套事务类型，是相对上面提到的六种情况（上面的六种应该称为平面事务类型），打个比方我现在有一个事务主要有一下几部分：  
      1，从A用户帐户里面减去100元钱  
      2，往B用户帐户里面添加100元钱  
       这样看和以前不同的事务可能没有什么区别，那我现在有点特殊的要求就是，A用户有3个帐户，B用户有2个帐户，现在我的要求就是只要再A用户的3个帐户里面任意一个减去100元，往B用户的两个帐户中任意一个里面增加100元就可以了！  
       一旦你有这样的要求那嵌套事务类型就非常适合你！我们可以这样理解，  
       一：将“从A用户帐户里面减去100元钱” 和 “往B用户帐户里面增加100元钱”我们暂时认为是一级事务操作  
       二：将从A用户的3个帐户的任意一个帐户里面减钱看做是“从A用户帐户里面减去100元钱”这个一级事务的子事务（二级事务），同样把后面存钱的看成是另一个的二级事务。  
      问题一：当二级事务被rollback一级事务会不会被rollback？  
      答案是不会的，二级事务的rollback只针对自己。  
      问题二：什么时候这个一级事务会commit，什么时候会被rollback呢？  
      我们主要看二级里面出现的情况，当所有的二级事务被commit了并且一级事务没有失败的操作，那整个事务就算是一个成功的事务，这种情况整个事务会被commit。  
当任意一个二级事务没有被commit那整个事务就是失败的，整个事务会被roolback。  
还是拿上面的例子来说明吧！如果我在a的三个帐户里面减钱的操作都被二级事务给rollback了，也就是3个帐户里面都没有减钱成功，整个事务就失败了就会被rollback。如果A用户帐户三个帐户里面有一个可以扣钱而且B用户的两个帐户里面也有一个帐户可以增加钱，那整个事务就算成功的，会被 commit。  
看了一下觉得上面的例子好像不是很深刻，看这个情况（A用户的3个帐户都是有信用额度的，也就是说可以超支，但是超支有金额限制）。不过原理是一样的，简单点也好说明一点，祝你好运！^\_^

分类: [Spring](http://www.cnblogs.com/jimmy-muyuan/category/780045.html)