@Transactional注解的属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 说明 |
| propagation | 枚举型：Propagation | （可选）事务传播行为。例如：  propagation=Propagation.REQUIRES\_NEW  详见后文 |
| readOnly | 布尔型 | 是否为只读型事务。例如：  readOnly=false |
| isolation | 枚举型：isolation | （可选）事务隔离级别。例如：  isolation=Isolation.READ\_COMMITTED |
| timeout | int型（单位：秒） | 事务超时时间。例如：  timeout=20 |
| rollbackFor | 一组Class类的实例，必须继承自Throwable | 一组异常类，遇到时必须进行回滚。例如：  rollbackFor={SQLException.class,  ArithmeticException.class} |
| rollbackForClassName | 一组Class类的名称，必须继承自Throwable | 一组异常类名，遇到时必须进行回滚。例如：  rollbackForClassName={"SQLException",  "ArithmeticException"} |
| noRollbackFor | 一组Class类的实例，必须继承自Throwable | 一组异常类，遇到时必须不回滚。 |
| noRollbackForClassName | 一组Class类的名称，必须继承自Throwable | 一组异常类名，遇到时必须不回滚。 |

一、Propagation （事务的传播属性）

Propagationkey属性确定代理应该给哪个方法增加事务行为。这样的属性最重要的部份是传播行为。有以下选项可供使用：PROPAGATION\_REQUIRED--支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。

PROPAGATION\_SUPPORTS--支持当前事务，如果当前没有事务，就以非事务方式执行。

PROPAGATION\_MANDATORY--支持当前事务，如果当前没有事务，就抛出异常。

PROPAGATION\_REQUIRES\_NEW--新建事务，如果当前存在事务，把当前事务挂起。

PROPAGATION\_NOT\_SUPPORTED--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。

PROPAGATION\_NEVER--以非事务方式执行，如果当前存在事务，则抛出异常。

1： PROPAGATION\_REQUIRED  
加入当前正要执行的事务不在另外一个事务里，那么就起一个新的事务  
比如说，ServiceB.methodB的事务级别定义为PROPAGATION\_REQUIRED, 那么由于执行ServiceA.methodA的时候，  
ServiceA.methodA已经起了事务，这时调用ServiceB.methodB，ServiceB.methodB看到自己已经运行在ServiceA.methodA  
的事务内部，就不再起新的事务。而假如ServiceA.methodA运行的时候发现自己没有在事务中，他就会为自己分配一个事务。  
这样，在ServiceA.methodA或者在ServiceB.methodB内的任何地方出现异常，事务都会被回滚。即使ServiceB.methodB的事务已经被  
提交，但是ServiceA.methodA在接下来fail要回滚，ServiceB.methodB也要回滚

2： PROPAGATION\_SUPPORTS  
如果当前在事务中，即以事务的形式运行，如果当前不再一个事务中，那么就以非事务的形式运行

3： PROPAGATION\_MANDATORY  
必须在一个事务中运行。也就是说，他只能被一个父事务调用。否则，他就要抛出异常

4： PROPAGATION\_REQUIRES\_NEW  
这个就比较绕口了。 比如我们设计ServiceA.methodA的事务级别为PROPAGATION\_REQUIRED，ServiceB.methodB的事务级别为PROPAGATION\_REQUIRES\_NEW，  
那么当执行到ServiceB.methodB的时候，ServiceA.methodA所在的事务就会挂起，ServiceB.methodB会起一个新的事务，等待ServiceB.methodB的事务完成以后，  
他才继续执行。他与PROPAGATION\_REQUIRED 的事务区别在于事务的回滚程度了。因为ServiceB.methodB是新起一个事务，那么就是存在  
两个不同的事务。如果ServiceB.methodB已经提交，那么ServiceA.methodA失败回滚，ServiceB.methodB是不会回滚的。如果ServiceB.methodB失败回滚，  
如果他抛出的异常被ServiceA.methodA捕获，ServiceA.methodA事务仍然可能提交。

5： PROPAGATION\_NOT\_SUPPORTED  
当前不支持事务。比如ServiceA.methodA的事务级别是PROPAGATION\_REQUIRED ，而ServiceB.methodB的事务级别是PROPAGATION\_NOT\_SUPPORTED ，  
那么当执行到ServiceB.methodB时，ServiceA.methodA的事务挂起，而他以非事务的状态运行完，再继续ServiceA.methodA的事务。

6： PROPAGATION\_NEVER  
不能在事务中运行。假设ServiceA.methodA的事务级别是PROPAGATION\_REQUIRED， 而ServiceB.methodB的事务级别是PROPAGATION\_NEVER ，  
那么ServiceB.methodB就要抛出异常了。

7： PROPAGATION\_NESTED  
理解Nested的关键是savepoint。他与PROPAGATION\_REQUIRES\_NEW的区别是，PROPAGATION\_REQUIRES\_NEW另起一个事务，将会与他的父事务相互独立，  
而Nested的事务和他的父事务是相依的，他的提交是要等和他的父事务一块提交的。也就是说，如果父事务最后回滚，他也要回滚的。  
而Nested事务的好处是他有一个savepoint。  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
ServiceA {

/\*\*  
\* 事务属性配置为 PROPAGATION\_REQUIRED  
\*/  
void methodA() {  
try {  
//savepoint  
ServiceB.methodB(); //PROPAGATION\_NESTED 级别  
} catch (SomeException) {  
// 执行其他业务, 如 ServiceC.methodC();  
}  
}

}  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
也就是说ServiceB.methodB失败回滚，那么ServiceA.methodA也会回滚到savepoint点上，ServiceA.methodA可以选择另外一个分支，比如  
ServiceC.methodC，继续执行，来尝试完成自己的事务。  
但是这个事务并没有在EJB标准中定义。

二、Spring事务的隔离级别  
 1. ISOLATION\_DEFAULT： 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  
 2. ISOLATION\_READ\_UNCOMMITTED： 这是事务最低的隔离级别，它充许令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。  
 3. ISOLATION\_READ\_COMMITTED： 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  
 4. ISOLATION\_REPEATABLE\_READ： 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，还保证了避免下面的情况产生(不可重复读)。  
 5. ISOLATION\_SERIALIZABLE 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。

什么是脏数据，脏读，不可重复读，幻觉读？

脏读: 指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据， 那么另外一个事务读到的这个数据是脏数据，依据脏数据所做的操作可能是不正确的。

不可重复读: 指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。  
              
幻觉读: 指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，这种修改涉及到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象发生了幻觉一样。