链滴

Spring 事务传播行为

作者：jackqiu

原文链接：https://ld246.com/article/1543196180234

来源网站：链滴

传播行为	含义
PROPAGATION_REQUIRED	表示当前方法必须

行在事务中。如果当前事务存在，方法将会在该事务中运行。否则，会启动一个新事务

PROPAGATION_SUPPORTS	表示当前方法不需

事务上下文，但是如果存在当前事务的话，那么该方法会在这个事务中运行

PROPAGATION_MANDATORY	g表示 该方法必

在事务中运行，如果当前事务不存在，则会抛出一个异常

PROPAGATION_REQUIRED_NEW	g表示当前方

必须运行在它自己的事务中。一个新的事务将被启动。如果存在当前事务，在该方法执行期间，当前务会被挂起。如果使用TransactionManager的话，则需要访问TransactionManager

PROPAGATION_NOT_SUPPORTED	g表示该方

不应该运行在事务。如果存在当前事务，在该方法运行期间，当前事务将被挂起。如果使用TransactinManager的话，则需要访问TransactionManager

PROPAGATION_NEVER	g表示当前方法不应该

行在事务上下文中。如果当前正有一个事务在运行，则会抛出异常

PROPAGATION_NESTED	g表示如果当前已经存

一个事务，那么该方法将会在嵌套事务中运行。嵌套的事务可以独立于当前事务进行单独地提交或回。如果当前事务不存在，那么其行为与PROPAGATION_REQUIRED一样。

一、PROPAGATION_REQUIRED

```
//事务属性是PROPAGATION_REQUIRED
@Transactional(propagation = Propagation.REQUIRED)
methodA{
 methodB();
}
```

使用spring声明式事务，spring使用AOP来支持声明式事务，会根据事务属性，自动在方法调用之间定是否开启一个事务，并在方法执行之后决定事务提交或回滚事务。

单独调用methodB方法:

```
@Transactional(propagation = Propagation.REQUIRED)
main{
 methodB();
}
```

相当于

```
Main{
 Connection con = null;
 try{
 con = getConnection();
 con.setAutoCommit(false);
 //方法调用
 methodB();
 //提交事务
 con.commit();
 }catch(Exception e){
 con.rollback();
 }
}
```

Spring保证在methodB方法中所有的调用都获得到一个相同的连接。在调用methodB时，没有一个在的事务，所以获得一个新的连接，开启了一个新的事务。

第二个例子：

```
@Transaction(propagation = Propagation.REQUIRED)
methodA(){
 methodB();
}
@Transaction(propagation = Propagation.REQUIRED)
methodB(){
 // do somthing
}
```

单独调用MethodA时，在MethodA内又会调用MethodB.

执行效果相当于:

```
main{
 Connection con = null;
 try{
 con = getConnection();
 methodA();
 con.commit();
 }catch(Exception e){
 con.rollback();
 }
}
```

调用MethodA时，环境中没有事务，所以开启一个新的事务。当在MethodA中调用MethodB时，境中已经有一个事务，所以methodB就加入当前事务。

二、PROPAGATION_SUPPORTS 如果存在一个事务，支当前事务。如果没有事务，则非事务的执行。

```
//事务属性 PROPAGATION_REQUIRED
@Transaction(propagation = Propagation.REQUIRED)
methodA(){
 methodB();
}
//事务属性 PROPAGATION_SUPPORTS
@Transaction(propagation = Propagation.SUPPORTS)
methodB(){

}
```

单纯的调用methodB时，methodB方法是非事务执行的。当调用methodA时，methodB则加入了mthodA的事务中，事务的执行。

三、PROPAGATION_MANDATORY 如果已经存在一个事，支持当前事务。如果没有事务，则抛出异常。

```

@Transactional(propagation = Propagation.REQUIRED)
methodA(){
 methodB();
}
//事务属性MANDATORY
@Transactional(propagation = Propagation.MANDATORY)
methodB(){

}
```

当单独调用methodB时，因为当前没有一个活动的事务，则会抛出异常，当调用methodA时，methdB则加入到methodA的事务中，事务地执行。

四、PROPAGATION_REQUIRES_NEW 总是开启一个新的务。如果一个事务已经存在，则将这个存在的事务挂起。

```
//事务属性 PROPAGATION_REQUIRED
@Transactional(propagation = Propagation.REQUIRED)
methodA(){
 doSomeThingA();
 methodB();
 doSomeThingB();
}
//事务属性 PROPAGATION_REQUIRES_NEW
@Transactional(propagation = Propagation.REQUIRED_NEW)
methodB(){

}
```

调用A方法

```
main(){
 mehtodA();
}
```

相当于

```
main(){
 TransactionManager tm = null;
 try{
 //获得一个JTA事务管理器
 tm = getTransactionManager();
 tm.begin();//开户一个新的事务
 Transaction ts1 = tm.getTransaction();
 doSomeThing();
 tm.suspend();//挂起当前事务
 try{
 tm.begin();//重新开启第二个事务
 Transaction ts2 = tm.getTransaction();
  methodB();
  ts2.commit();//提交第二个事务
 }catch(Exception e){
 ts2.rollback();
 }
 tm.resume(ts1);
 doSomeThingB();
 ts1.commit();//提交第一个事务
 }catch(Exception e){
 ts1.rollback();
 }
}
```

在这，ts1称为外层事务，ts2称为内层事务。从上面的代码来看，ts2与ts1是两个独立的事务，互不干。ts2是否成功并不依赖于ts1。如果methodA方法在调用methodB方法后在doSomeThingB方法败了，而methodB方法所做的结果依然被提交。而除了methodB之外的其他代码导致的结果去被回了。

五、PROPAGATION_NOT_SUPPORTED 总是非事务地执，并挂起任何存在的事务。

六、PROPAGATION_NEVER 总是非事务地执行，如果存一个活动事务，则抛出异常。

七、PROPAGATION_NESTED 如果一个活动的事务存在，运行在一个嵌套的事务中，如果没有活动事务，则按TransactonDefinition.PROPAGATION_REQUIRED属性执行。

```
@Transctional(propagation = Propagetion.REQUIRED)
methodA(){
 doSomeThingA();
 methodB();
```

原文链接：Spring 事务传播行为

```
 doSomeThingB();
}
@Transctional(propagation = Propagation.NEWSTED)
methodB(){
 ....
}
```

如果单独调用methodB方法，则按REQUIRED属性执行。如果调用methodA方法相当于下面的效果：

```
main(){
 Connection con = null;
 Savepoint savepoint = null;
 try{
 con = getCoonnection();
 con.setAutoCommit(false);
 doSomeThingA();
 savepoint = con2.setSavepoint();
 try{
 methodB();
 }catch(RuntimeException ex){
 con.rollback(savepoint);
 }
 doSomeThingB();
 con.commit();
 }
}
```

当methodB方法调用之前，调用setSavepoint方法，保存当前的状态到savepoint。如果methodB法调用失败，则恢复到之前保存的状态。但是需要注意的是，这时的事务并没有进行提交，如果后续代码(doSomeThingB())调用失败，则回滚包括methodB方法的所有操作。嵌套事务一个非常重要的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。而内层事务操作失并不会引起包层事务的回滚。