**Spring的事务实现原理是什么？能聊聊你对事务传播机制的理解吗？**[](" \l "spring" \o "Permanent link)

**事务的实现原理？**[](" \l "_1" \o "Permanent link)

如果说你加了一个@Transactional注解，**此时就Spring会使用AOP思想**，对你的这个方法在执行之前，先去开启事务，执行完毕之后，根据你的方法是否报错，来决定回滚还是提交事务。

@Transactionl**(**propagation **=** Propagation**.**REQUIRED**)**

**public** **void** **methodA(){**

doSomethingPre**();**

methodB**();**

doSomethingPost**();**

**}**

@Transactionl**(**propagation **=** Propagation**.**NESTED**)**

因此事务的实现原理就是 **AOP**

**事务的传播机制？**[](" \l "_2" \o "Permanent link)

**Spring有哪些事务传播机制**[](" \l "spring_1" \o "Permanent link)

1. PROPAGATION\_REQUIRED(默认) required: 如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入该事务，该设置是最常用的设置。
   * 比如 methodA和meghodB方法上都添加了@Transactional注解，但是没有指明事务的类型
   * 这个时候methodA()和methodB()都会使用PROPAGATION\_REQUIRED传播机制
   * 但是两个方法 **只会开启一个事务**，把methodB()的代码加入到methodA当中了。即执行流程如下：
     + 开启一个事务
     + 执行方法A的代码，接着执行方法B的代码
     + 提交或者回滚事务
   * 如果methodB()方法里有执行失败，会导致methodA()方法进行回滚，因为mthodA和methodB都在一个事务中。
2. PROPAGATION\_SUPPORTS supports: 支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就以非事务执行。（比较少用）
   * 当methodB的事务传播机制是：support
     + 通过methodA去调用methodB()的时候，因为methodA中已经有一个事务了，则methodB就加入到methodA的事务当中。
     + 当直接调用methodB，而不通过methodA时候，会发现之前并没有开启事务，这个时候methodB是不会开启事务的，会以非事务开启
3. PROPAGATION\_MANDATORY mandatory(强制的): 中文翻译为强制，支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就抛出异常Exception。
   * 当methodB的事务传播机制是：mandatory
     + 通过methodA去调用methodB()的时候，因为methodA中已经有一个事务了，则methodB就加入到methodA的事务当中。就和support类似
     + 当直接调用methodB，而不通过methodA时候，会发现之前并没有开启事务，这个时候methodB会抛出异常。
4. PROPAGATION\_REQUIRES\_NEW requires\_new: 创建新事务，无论当前存不存在事务，都创建新事务. - 当methodB的事务传播机制是：requireds\_new + 通过methodA去调用methodB()的时候，因为methodA中已经有一个事务了，但是methodB依然会创建一个新的事务。具体流程如下： \* 开启一个事务1 \* 执行方法A里的一些代码：doSomethingPre() \* 开启一个事务2 \* 执行方法B里的一些代码 \* 提交或者回滚事务2（注意此处，只是回滚方法B） \* 执行方法A里的一些代码，doSomethingPost() \* 提交或者回滚过事务1（注意此处，若dosomethingPost()中报错，方法B中的内容已经成功提交了，是不会回滚的）
5. 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。即强制以非事务方式执行。很少用。
6. PROPAGATION\_NEVER never: 以非事务方式执行，如果当前存在事务，则抛出异常Exception。
7. PROPAGATION\_NESTED nested(嵌套的): 嵌套事务，如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则按REQUIRED属性执行。 - 当methodB的事务传播机制是：nested + 通过methodA去调用methodB()的时候，因为methodA中已经有一个事务了，但是methodB依然会创建一个新的事务。具体流程如下： \* 开启一个事务1 \* 执行方法A里的一些代码：doSomethingPre() \* 设置一个回滚点，savePoint \* 执行方法B里的一些代码 \* 如果方法B里抛出了异常，此时进行回滚，回滚到之前的savePoint \* 执行方法A里的一些代码，doSomethingPost() \* 提交或者回滚过事务（注意此处，若dosomethingPost()中报错，会回滚方法B及方法A中执行的所有操作） - 这个其实叫做嵌套事务，外层的事务如果回滚，会导致内层的事务也回滚；但是内层的事务回滚了，仅仅是回滚自己的代码。

上面事务传播机制中：required、required\_new和nested是最容易弄混的，因为他们看起来很相似，但是却有不同的地方。

mandatory和support很相似；never和not\_support很相似

**Spring事务传播机制解决的是一个什么样的问题？**[](" \l "spring_2" \o "Permanent link)

Spring的事务传播机制解决的就是：**两个或者多个加了@Transactional的事务方法，他们之间互相串起来调用的时候，事务是如何传播的。**

@Transactionl**(**propagation **=** "Propagation."**)**

**public** **void** **methodA(){**

methodB**();**

*//do something*

**}**

@Transactionl**(**propagation **=** "Propagation."**)**

**public** **void** **methodB(){**

*//do something*

**}**

**面试题：**[](#_3)

不会直接问你事务的传播机制的，会这么问：

**我们现在有一段业务逻辑，方法A调用方法B,我希望的是如果说方法A出错了，此时仅仅回滚方法A,不能回滚方法B，必须得用REQUIRES\_NEW，传播机制，让他们两个的事务是不同的。**

**方法A调用方法B，如果方法B出错，方法B只能回滚他自己，方法A可以带着方法B一起回滚，NESTED嵌套事务**