其实数据库的七种传播行为在网上查询了一下，和Spring中使用到的7种事务传播行为是没什么差别的，因为我写的博文基本和JAVA有相关，所以就结合Spring中事务传播和例子来讲一下咯。

事务传播行为如下：

Require：支持当前事务，如果没有事务，就建一个新的，**这是最常见的**；

Supports：支持当前事务，如果当前没有事务，**就以非事务方式执行**；

Mandatory：支持当前事务，如果当前没有事务，**就抛出异常**；

RequiresNew：**新建事务**，如果当前存在事务，把当前事务挂起；

NotSupported：**以非事务方式执行操**作，如果当前存在事务，就把事务挂起；

Never：以非事务方式执行，如果当前存在事务，**则抛出异常**。

Nested：新建事务，如果当前存在事务，把当前事务挂起。**与RequireNew的区别是与父事务相关，且有一个savepoint。即则在嵌套事务内执行。**

其中，Require、Supports、NotSupported、Never两个看文字也就能了解，就不多说了。而Mandatory是要求所有的操作必须在一个事务里，较Require来说，对事务要求的更加严格。

RequireNew：当一个Require方法A调用RequireNew方法B时，B方法会新new一个事务，并且这个事务和A事务没有关系，也就是说B方法出现异常，不会导致A的回滚，同理当B已提交，A再出现异常，B也不会回滚。

Nested：这个和RequireNew的区别是B方法的事务和A方法的事务是相关的。只有在A事务提交的时候，B事务都会提交。也就是说当A发生异常时，A、B事务都回滚，而当B出现异常时，B回滚，而A回滚到savepoint，如下代码所示：

publicvoid A(){

    //操作1

    //操作2

    //操作3

    try{

        //savepoint

        B();//一个Nested的方法

    } catch{

        //出现异常，B方法回滚，A方法回滚到

        //savepoint，也就是说操作1、2、3 都还在

       C();

    } finally{

    }

}

**使用步骤：**

**步骤一、在spring配置文件中引入<tx:>命名空间**

<beansxmlns="http://www.springframework.org/schema/beans"

 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

**xmlns:tx="http://www.springframework.org/schema/tx"**

 xsi:schemaLocation="http://www.springframework.org/schema/beans

 http://www.springframework.org/schema/beans/spring-beans-2.0.xsd

**http://www.springframework.org/schema/tx**

**http://www.springframework.org/schema/tx/spring-tx-2.0.xsd">**

**步骤二、具有@Transactional 注解的bean自动配置为声明式事务支持**

<!-- 事务管理器配置, Hibernate单数据源事务 --> <bean id="defaultTransactionManager"class="org.springframework.orm.hibernate3.HibernateTransactionManager"><property name="sessionFactory"ref="sessionFactory"/> </bean> <!-- 使用annotation定义事务--> <tx:annotation-driventransaction-manager="defaultTransactionManager" proxy-target-class="true"/>

**步骤三、在接口或类的声明处 ,写一个@Transactional.**

要是只在接口上写, 接口的实现类就会继承下来、接口的实现类的具体方法,可以覆盖类声明处的设置

@Transactional   //类级的注解、适用于类中所有的public的方法

**事务的传播行为和隔离级别**

大家在使用spring的注解式事务管理时，对事务的传播行为和隔离级别可能有点不知所措，下边就详细的介绍下以备方便查阅。

**事物注解方式: @Transactional**

当标于类前时, 标示类中所有方法都进行事物处理 , 例子:

@Transactional public classTestServiceBean implements TestService {}

**当类中某些方法不需要事物时:**

@Transactional public classTestServiceBean implements TestService { private TestDao dao; publicvoid setDao(TestDao dao) { this.dao = dao; } @Transactional(propagation =Propagation.NOT\_SUPPORTED) publicList<Object> getAll() {return null; } }

**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.