Spring声明式事务

Spring声明式事务的底层就是AOP思想，要记住事务的ACID原则

编程式事务：通过编程代码在业务逻辑时需要时自行实现，粒度更小；
声明式事务：通过注解或XML配置实现；

回顾一下事务：

> 一个事务其实就是一个完整的业务逻辑。

Service类里的每一个方法都是一个完整的业务。所以需要事务控制的，一定是Service类中涉及DML操作的方法。

1.1 注解方式实现声明式事务

如果需要对Service类中的一个方法进行事务控制，只要在这个方法上方加上@Transactional即可。

1.2 事务传播行为

什么是事务的传播行为?

在service类中有a()方法和b()方法，a()方法上有事务，b()方法上也有事务，当a()方法执行过程中调用了b()方法，事务是如何传递的？合并到一个事务里？还是开启一个新的事务？这就是事务传播行为。

事务传播行为在spring框架中被定义为枚举类型：

```
package org.springframework.transaction.annotation;

public enum Propagation {
    REQUIRED( value: 0),
    SUPPORTS( value: 1),
    MANDATORY( value: 2),
    REQUIRES_NEW( value: 3),
    NOT_SUPPORTED( value: 4),
    NEVER( value: 5),
    NESTED( value: 6);
```


一共有七种传播行为：

- REQUIRED(默认)：没有就新建，有就加入
- SUPPORTS：有就加入，没有就没有
- MANDATORY：有就加入，没有就抛异常
- REQUIRES_NEW：开启一个新的事务，如果一个事务已经存在，则将这个存在的事务挂起【不管有没有，直接开启一个新事务，开启的新事务和之前的事务不存在嵌套关系，之前事务被挂起（挂起就是暂停的意思）】
- NOT_SUPPORTED：如果有事务存在，挂起当前事务【不支持事务，存在就挂起】
- NEVER：如果有事务存在，抛出异常【不支持事务，存在就抛异常】
- NESTED：有事务的话，就在这个事务里再嵌套一个完全独立的事务，嵌套的事务可以独立的提交和回滚。没有事务就和REQUIRED一样。

在代码中设置事务的传播行为：

```
@Transactional(propagation = Propagation.REQUIRED)
```

1.3 事务隔离级别

数据库中读取数据存在的三大问题：（三大读问题）

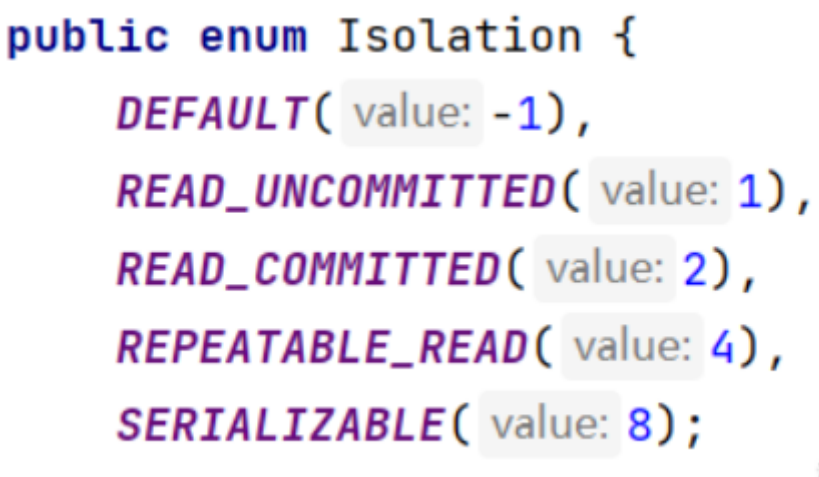
- 脏读：读取到没有提交到数据库的数据，叫做脏读。
- 不可重复读：在同一个事务当中，第一次和第二次读取的数据不一样。
- 幻读：脑子中认为的数据和数据库中的真实数据不同，就是幻读。只要是多事务并发，就无法避免幻读问题。

事务隔离级别包括四个级别：

- 读未提交：READ_UNCOMMITTED
-
  - 这种隔离级别，存在脏读问题，所谓的脏读(dirty read)表示能够读取到其它事务未提交的数据。
- 读提交：READ_COMMITTED
-
  - 解决了脏读问题，其它事务提交之后才能读到，但不可重复读和幻读问题仍然有可能发生，没有避免不可重复读和幻读问题。
- 可重复读：REPEATABLE_READ
-
  - 解决了不可重复读，可以达到可重复读效果，只要当前事务不结束，读取到的数据一直都是一样的。但幻读问题仍然有可能发生，没有避免幻读问题。
- 序列化：SERIALIZABLE
-
  - 解决了幻读问题，事务排队执行。不支持并发。

在Spring代码中如何设置隔离级别?

隔离级别在spring中以枚举类型存在：




```
@Transactional(isolation = Isolation.READ_COMMITTED)
```

1.4 事务超时

代码如下：

```
@Transactional(timeout = 10)
```

以上代码表示设置事务的超时时间为10秒。

表示超过10秒如果该事务中所有的DML语句还没有执行完毕的话，最终结果会选择回滚。

默认值-1，表示没有时间限制。

这里有个坑，事务的超时时间指的是哪段时间?

在当前事务当中，最后一条DML语句执行之前的时间。如果最后一条DML语句后面很有很多业务逻辑，这些业务代码执行的时间不被计入超时时间。

以下代码的超时不会被计入超时时间：

```
@Transactional(timeout = 10) // 设置事务超时时间为10秒。
public void save(Account act) {
    accountDao.insert(act);
    // 睡眠一会
    try {
        Thread.sleep(1000 * 15);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

以下代码超时时间会被计入超时时间：

```
@Transactional(timeout = 10) // 设置事务超时时间为10秒。
public void save(Account act) {
    // 睡眠一会
    try {
        Thread.sleep(1000 * 15);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    accountDao.insert(act);
}
```

当然，如果想让整个方法的所有代码都计入超时时间的话，可以在方法最后一行添加一行无关紧要的DML语句。

1.5 只读事务

```
@Transactional(readOnly = true)
```

将当前事务设置为只读事务，在该事务执行过程中只允许select语句执行，delete insert update均不可执行。

如果该事务中确实没有增删改操作，建议设置为只读事务。

按道理没有必要给只含select的业务方法添加事务，设置只读事务的原因是设置只读事务有个作用：**启动spring的优化策略。提高select语句执行效率。**

1.6 设置哪些异常回滚事务

```
@Transactional(rollbackFor = RuntimeException.class)
```

表示只有发生RuntimeException异常或该异常的子类异常才回滚。

1.7 设置哪些异常不回滚事务

```
@Transactional(noRollbackFor = NullPointerException.class)
```

表示发生NullPointerException或该异常的子类异常不回滚，其他异常则回滚。