73_精选面试题：什么情况导致 Spring 事务失效？

1、开篇

上节课带大家讲创建订单的整个流程测试了一遍，验证了系统的整个下单流程，让大家从感性走入了理性。本节课加入开胃下菜，聊聊面试中经常遇到的问题：什么情况导致 Spring 事务失效？今天课程的内容包括以下几个部分：

前面几节课聊到 Spring 事务通过 AOP 的方式对事务进行启动、执行和管理，似乎只要声明了@Transactional 注释的方法就可以执行事务操作了。其实不然，这里列举了 7 个不支持事务的场景，也是面试官比较青睐的问题，我们一起来看看吧。

2、没有被 Spring 管理

没有被 Spring 管理的 Bean，如果其中出现了方法需要进行事务处理的情况，此时的事务不会执行。如图 1 所示，在 OrderServiceImpl 类中如果将@Service 注释掉，那么此类对应的 bean 也就不会被 Spring IoC 容器管理。即便 updateOrder 上面注释了@Transactional，这个方法也不会执行事务。

```
// @Service
public class OrderServiceImpl implements OrderService {
 @Transactional
public void updateOrder(Order order) {
 // update order;
  }
}
```

图 1

3、方法不是 public

需要注意的是，@Transactional 只能用于 public 的方法上，否则事务不会失效，如果要用在非 public 方法上，可以开启 AspectJ 代理模式。如图 2 所示，

saveAll 方法上面标注了@Transactional 的注释，由于 saveAll 方法没有标注 public，因此 saveAll 中的内容是不会执行事务的。

```
@Service
public class DemoServiceImpl implements  DemoService {

    @Transactional(rollbackFor = SQLException.class)
    @Override
     int saveAll(){  // 编译器一般都会在这个地方给出错误提示
        // do someThing;
        return  1;
    }
}
```

图 2

4、自身调用问题

当一个服务中存在多个方法，都标注了事务@Transactional，当其中一个方法调用另外一个方法的时候，被调用方法中的事务是不会执行的。如图 3 所示，在 ServiceA 中存在方法 doSomething 有@Transactional 的标注，在方法体内部会调用 insert 方法，insert 方法上面也有@Transactional 的标注。此时 doSomething 中的事务可以执行，但是 insert 方法的事务就无法执行。

```java
@Service
public class ServiceA {

  @Transactional
  public void doSomething(){

    insert();

    调用其他系统;
  }

  @Transactional(propagation = Propagation.REQUIRES_NEW)
  public void insert(){
    向数据库中添加数据;
  }
}
```

图 3

为了解决这个问题可以将 insert 方法放到另外一个类中进行调用。如图 4， 将 doSomething 和 insert 方法分别放到 ServiceA 和 ServiceB 两个类中，然后再通过 doSomething 调用 insert 此时两个方法中的事务都会执行。

```
@Service
public class ServiceA {

  @Autowired
  private ServiceB serviceB;
  @Transactional
  public void doSomething(){

    serviceB.insert();

    调用其他系统;
  }
}
@Service
public class ServiceB {

  @Transactional(propagation = Propagation.REQUIRES_NEW)
  public void insert(){
    向数据库中添加数据;
  }
}
```

图 4

5、数据源没有配置事务管理器

如图 5 所示数据源若没有配置事务管理器，那么下面的方法 transactionManager 是无法实现事务的。

```
@Bean
public PlatformTransactionManager transactionManager(DataSource dataSource) {
    return new DataSourceTransactionManager(dataSource);
}
```

图 5

6、事务传播级别选择

如果在方法嵌套调用时存在事务传播的场景，在定义事务级别的时候使用了 NOT_SUPPORTED 那么对应的方法就不会执行事务。如图 6 所示，在 OrderServiceImpl 中执行的 update 方法执行了事务，同时该方法调用 updateOrder 方法。由于 updateOrder 方法的@Transactional 注释中标注了 Propagation. NOT_SUPPORTED，因此 updateOrder 方法的事务是不执行的。

```
@Service
public class OrderServiceImpl implements OrderService {
    @Transactional
    public void update(Order order) {
        updateOrder(order);
    }
    @Transactional(propagation = Propagation.NOT_SUPPORTED)
    public void updateOrder(Order order) {
        // update order;
    }
}
```

图 6

7、事务中出现异常

如果在执行的方法中执行的事务，由于异常退出的情况，那么这个事务是无法完成的。如图 7 所示，在 updateOrder 方法中使用了 try catch，一旦在方法体中出现异常，此时事务会中断无法执行。

```
@Service
public class OrderServiceImpl implements OrderService {
    @Transactional
    public void updateOrder(Order order) {
        try {
            // update order;
         }catch (Exception e){
            //do something;
        }
    }
}
```

图 7

8、错误定义异常类型

在事务中出现异常的时候，需要对异常进行定义才能正确执行事务。如果出现错误的定义，当出现事务中的异常时，还可以继续执行事务。如图 8 所示，在方法 updateOrder 中对 Exception 进行了捕捉，但是在方法上面的 Transactional 注释中 rollbackFor 中定义的是 SQLException.class ，正确的定义应该是 Exception.class。

```
@Service
public class OrderServiceImpl implements OrderService {
 @Transactional
   // @Transactional(rollbackFor = SQLException.class)
    public void updateOrder(Order order) {
        try {            // update order
          }catch (Exception e){
           throw new Exception("更新错误");
        }
    }
}
```

图 8

9、总结

本节课给大家讲解了面试经典题，有哪 7 种情况会导致 Spring 事务失效。本周从创建订单的业务流程作为切入点，思考如何使用 Spring 的机制完成日期输入的验证，同时提出了订单、优惠券、积分数据一致性的问题。并且利用 Spring 的声明式事务讲解了多操作的事务问题，最后带大家讲创建订单的代码测试了一遍。

下周课程回围绕订单取消功能进行讲解，会使用到 Spring JMS 和 RocketMQ 延迟消息等功能。下期见，拜拜。