[https://mp.weixin.qq.com/s?\_\_biz=MzA5MTkxMDQ4MQ==&mid=2648933334&idx=1&sn=3a68da36e4e21b7339418e40ab9b6064&chksm=88621be8bf1592fe5301aab732fbed8d1747475f4221da341350e0cc9935225d41bf79375d43&token=1919005508&lang=zh\_CN&scene=21#wechat\_redirect](https://mp.weixin.qq.com/s?__biz=MzA5MTkxMDQ4MQ==&mid=2648933334&idx=1&sn=3a68da36e4e21b7339418e40ab9b6064&chksm=88621be8bf1592fe5301aab732fbed8d1747475f4221da341350e0cc9935225d41bf79375d43&token=1919005508&lang=zh_CN&scene=21" \l "wechat_redirect)

## 接口幂等性这么重要，它是什么？怎么实现？

### 什么是幂等性？

对于同一笔业务操作，不管调用多少次，得到的结果都是一样的。

### 幂等性设计

我们以对接支付宝充值为例，来分析支付回调接口如何设计？

如果我们系统中对接过支付宝充值功能的，我们需要给支付宝提供一个回调接口，支付宝回调信息中会携带（out\_trade\_no【商户订单号】，trade\_no【支付宝交易号】），trade\_no在支付宝中是唯一的，out\_trade\_no在商户系统中是唯一的。

回调接口实现有以下实现方式。

### 方式1（普通方式）

过程如下：

1.接收到支付宝支付成功请求  
2.根据trade\_no查询当前订单是否处理过  
3.如果订单已处理直接返回，若未处理，继续向下执行  
4.开启本地事务  
5.本地系统给用户加钱  
6.将订单状态置为成功  
7.提交本地事务

上面的过程，对于同一笔订单，如果支付宝同时通知多次，会出现什么问题？当多次通知同时到达第2步时候，查询订单都是未处理的，会继续向下执行，最终本地会给用户加两次钱。

此方式适用于单机器，通知按顺序执行的情况，只能用于自己写着玩玩。

### 方式2（jvm加锁方式）

方式1中由于并发出现了问题，此时我们使用java中的Lock加锁，来防止并发操作，过程如下：

1.接收到支付宝支付成功请求  
2.调用java中的Lock加锁  
3.根据trade\_no查询当前订单是否处理过  
4.如果订单已处理直接返回，若未处理，继续向下执行  
5.开启本地事务  
6.本地系统给用户加钱  
7.将订单状态置为成功  
8.提交本地事务  
9.释放Lock锁

分析问题：  
Lock只能在一个jvm中起效，如果多个请求都被同一套系统处理，上面这种使用Lock的方式是没有问题的，不过互联网系统中，多数是采用集群方式部署系统，同一套代码后面会部署多套，如果支付宝同时发来多个通知经过负载均衡转发到不同的机器，上面的锁就不起效了。此时对于多个请求相当于无锁处理了，又会出现方式1中的结果。此时我们需要分布式锁来做处理。

### 方式3（悲观锁方式）

使用数据库中悲观锁实现。悲观锁类似于方式二中的Lock，只不过是依靠数据库来实现的。数据中悲观锁使用for update来实现，过程如下：

1.接收到支付宝支付成功请求  
2.打开本地事物  
3.查询订单信息并加悲观锁



4.判断订单是已处理  
5.如果订单已处理直接返回，若未处理，继续向下执行  
6.给本地系统给用户加钱  
7.将订单状态置为成功  
8.提交本地事物

重点在于for update，对for update，做一下说明：  
1.当线程A执行for update，数据会对当前记录加锁，其他线程执行到此行代码的时候，会等待线程A释放锁之后，才可以获取锁，继续后续操作。  
2.事物提交时，for update获取的锁会自动释放。

方式3可以正常实现我们需要的效果，能保证接口的幂等性，不过存在一些缺点：  
1.如果业务处理比较耗时，并发情况下，后面线程会长期处于等待状态，占用了很多线程，让这些线程处于无效等待状态，我们的web服务中的线程数量一般都是有限的，如果大量线程由于获取for update锁处于等待状态，不利于系统并发操作。

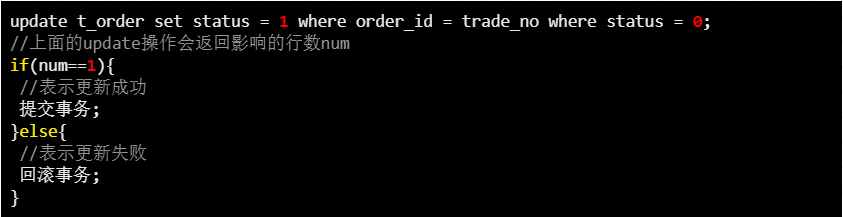
### 方式4（乐观锁方式）

依靠数据库中的乐观锁来实现。

1.接收到支付宝支付成功请求  
2.查询订单信息



1. 判断订单是已处理  
   4.如果订单已处理直接返回，若未处理，继续向下执行  
   5.打开本地事物  
   6.给本地系统给用户加钱  
   7.将订单状态置为成功，注意这块是重点，伪代码：

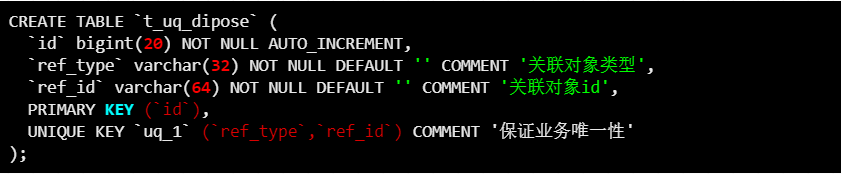


注意：  
update t\_order set status = 1 where order\_id = trade\_no where status = 0; 是依靠乐观锁来实现的，status=0作为条件去更新，类似于java中的cas操作；关于什么是cas操作，可以移步：什么是 CAS 机制 ( http://www.itsoku.com/article/63 )?  
执行这条sql的时候，如果有多个线程同时到达这条代码，数据内部会保证update同一条记录会排队执行，最终最有一条update会执行成功，其他未成功的，他们的num为0，然后根据num来进行提交或者回滚操作。

### 方式5（唯一约束方式）

依赖数据库中唯一约束来实现。

我们可以创建一个表：



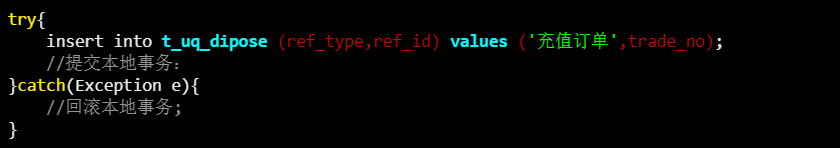
对于任何一个业务，有一个业务类型(ref\_type)，业务有一个全局唯一的订单号，业务来的时候，先查询t\_uq\_dipose表中是否存在相关记录，若不存在，继续放行。

过程如下：

1.接收到支付宝支付成功请求  
2.查询t\_uq\_dipose(条件ref\_id,ref\_type)，可以判断订单是否已处理



1. 判断订单是已处理  
   4.如果订单已处理直接返回，若未处理，继续向下执行  
   5.打开本地事物  
   6.给本地系统给用户加钱  
   7.将订单状态置为成功  
   8.向t\_uq\_dipose插入数据，插入成功，提交本地事务，插入失败，回滚本地事务，伪代码：



说明：  
对于同一个业务，ref\_type是一样的，当并发时，插入数据只会有一条成功，其他的会违法唯一约束，进入catch逻辑，当前事务会被回滚，最终最有一个操作会成功，从而保证了幂等性操作。  
关于这种方式可以写成通用的方式，不过业务量大的情况下，t\_uq\_dipose插入数据会成为系统的瓶颈，需要考虑分表操作，解决性能问题。  
上面的过程中向t\_uq\_dipose插入记录，最好放在最后执行，原因：插入操作会锁表，放在最后能让锁表的时间降到最低，提升系统的并发性。

关于消息服务中，消费者如何保证消息处理的幂等性？  
每条消息都有一个唯一的消息id，类似于上面业务中的trade\_no，使用上面的方式即可实现消息消费的幂等性。

### 总结

1.实现幂等性常见的方式有：悲观锁（for update）、乐观锁、唯一约束  
2.几种方式，按照最优排序：乐观锁 > 唯一约束 > 悲观锁