常见用来保证幂等的手段

1.MVCC方案

多版本并发控制，该策略主要使用update with condition（更新带条件来防止）来保证多次外部请求调用对系统的影响是一致的。在系统设计的过程中，合理的使用乐观锁，通过version或者updateTime（timestamp）等其他条件，来做乐观锁的判断条件，这样保证更新操作即使在并发的情况下，也不会有太大的问题。例如

> select * from tablename where condition=#condition# //取出要跟新的对象，带有版本versoin
>
> update tableName set name=#name#,version=version+1 where version=#version#

在更新的过程中利用version来防止，其他操作对对象的并发更新，导致更新丢失。为了避免失败，通常需要一定的重试机制。

2.去重表

在插入数据的时候，插入去重表，利用数据库的唯一索引特性，保证唯一的逻辑。

这种方法适用于在业务中有唯一标的插入场景中，比如在以上的支付场景中，如果一个订单只会支付一次，所以订单ID可以作为唯一标识。这时，我们就可以建一张去重表，并且把唯一标识作为唯一索引，在我们实现时，把创建支付单据和写入去去重表，放在一个事务中，如果重复创建，数据库会抛出唯一约束异常，操作就会回滚。

3.悲观锁

select for update，整个执行过程中锁定该订单对应的记录。注意：这种在DB读大于写的情况下尽量少用。

4. select + insert

并发不高的后台系统，或者一些任务JOB，为了支持幂等，支持重复执行，简单的处理方法是，先查询下一些关键数据，判断是否已经执行过，在进行业务处理，就可以了。注意：核心高并发流程不要用这种方法。

5.状态机幂等

在设计单据相关的业务，或者是任务相关的业务，肯定会涉及到状态机，就是业务单据上面有个状态，状态在不同的情况下会发生变更，一般情况下存在有限状态机，这时候，如果状态机已经处于下一个状态，这时候来了一个上一个状态的变更，理论上是不能够变更的，这样的话，保证了有限状态机的幂等。

这种方法适合在有状态机流转的情况下，比如就会订单的创建和付款，订单的付款肯定是在之前，这时我们可以通过在设计状态字段时，使用int类型，并且通过值类型的大小来做幂等，比如订单的创建为0，付款成功为100。付款失败为99

在做状态机更新时，我们就这可以这样控制

update `order` set status=#{status} where id=#{id} and status<#{status}

6. token机制，防止页面重复提交

业务要求：页面的数据只能被点击提交一次

发生原因：由于重复点击或者网络重发，或者nginx重发等情况会导致数据被重复提交

解决办法：

集群环境：采用token加redis（redis单线程的，处理需要排队）
单JVM环境：采用token加redis或token加jvm内存
处理流程：

数据提交前要向服务的申请token，token放到redis或jvm内存，token有效时间
提交后后台校验token，同时删除token，生成新的token返回
token特点:要申请，一次有效性，可以限流

7. 对外提供接口的api如何保证幂等

如银联提供的付款接口：需要接入商户提交付款请求时附带：source来源，seq序列号。source+seq在数据库里面做唯一索引，防止多次付款，(并发时，只能处理一个请求)

总结：幂等性应该是合格程序员的一个基因，在设计系统时，是首要考虑的问题，尤其是在像支付宝，银行，互联网金融公司等涉及的都是钱的系统，既要高效，数据也要准确，所以不能出现多扣款，多打款等问题，这样会很难处理，用户体验也不好 。

8.全局唯一ID

如果使用全局唯一ID，就是根据业务的操作和内容生成一个全局ID，在执行操作前先根据这个全局唯一ID是否存在，来判断这个操作是否已经执行。如果不存在则把全局ID，存储到存储系统中，比如数据库、redis等。如果存在则表示该方法已经执行。

从工程的角度来说，使用全局ID做幂等可以作为一个业务的基础的微服务存在，在很多的微服务中都会用到这样的服务，在每个微服务中都完成这样的功能，会存在工作量重复。另外打造一个高可靠的幂等服务还需要考虑很多问题，比如一台机器虽然把全局ID先写入了存储，但是在写入之后挂了，这就需要引入全局ID的超时机制。

使用全局唯一ID是一个通用方案，可以支持插入、更新、删除业务操作。但是这个方案看起来很美但是实现起来比较麻烦，下面的方案适用于特定的场景，但是实现起来比较简单。