链滴

关于幂等性的总结

作者：xf616510229
原文链接：https://ld246.com/article/1560589750858
来源网站：链滴

关于幂等性的总结

幂等性

幂等性是数学概念，即f(x)=f(f(x))。在计算机领域，则是意为对同一个系统，使用同样的条件，一次求和重复的多次请求对系统资源的影响是一致的。在调用接口时，总有一些特殊情况会导致接口进行复的调用，如果不对这些情况做出处理，就可能导致脏数据，甚至是业务流程上的问题。

比如，后台管理员在新增数据时，由于特殊原因，比如按钮抖动，而导致目标数据插入两条；用户下时，没有做逻辑校验，导致用户下了多笔相同的订单。这些情况都是不被允许的，我们要保证这些操无论执行多少次，最终产生的结果都是相同的，这类业务通常需要拥有较强的一致性。幂等性就是描这一接口特性的名词。

幂等性的保证需要从两个方面下手：

- 空间维度
    - 空间维度定义了接口幂等的范围
    - 比如我是订单数据，范围就是不同的用户id和订单id(相同订单id不能重复下单)，或者是商品(相商品不能重复下单)
- 时间维度
    - 时间维度则是定义了接口幂性的有效期
    - 比如，订单需要保证永久性幂等，永远不能拥有相同的订单
    - 部分业务只需要保证一段时间的幂等性

关于永久性的幂等性校验：

- 永久性的幂等通常与业务有着强关联
- 所以，永久性的幂等性判断无法做出统一的处理，通常放置在接口的参数校验上

并发安全性问题：

- 在高并发的情况下，幂等性的判断可能具有并发安全问题
- 比如，两个客户同时登录一个账户，并同时提交相同表单，`if (form not exist) {execute();}` 两个求可能会同时进入if代码块内执行 `execute()`，此时就出现了并发安全问题
- 因为是并发安全问题，所以我们可以使用锁来解决这个问题

幂等问题出现的场景：

- 接口的幂等性问题通常出现在修改操作或者增添操作上，而针对查询与删除操作，通常不会拥有此问题
- 需要保证幂等性的主要是业务上要求一致性较高的场景(比如支付，数据不一致会导致严重的业务问)以及事务性强的场景

解决方案

select + insert

解决永久性幂等问题的一种解决方案。就是在执行操作前，判断之前是否已经操作过了，比如下单之检查是否具有相同的订单ID已经存在，这样就可以避免重复提交的问题。永久性幂等问题通常与业务辑具有强关联，所以校验通常放置在参数校验上。

```
if (!order exists) {
    addOrder();
} else {
    throw OrderExistsException();
}
```

针对并发问题带来的脏数据的问题无法解决，需要通过下面几种方式来实现。

唯一索引

这是最容易想到的方式。页面的数据通常只能被提交一次，多次提交可能会产生脏数据。比如，同一称的商品只能被创建一次，为了防止创建多次，可以给商品名称添加唯一索引。当在添加一个已有名的商品时，数据库插入操作就会因为唯一索引而引发异常，避免了脏数据的产生。类似的案例还有博点赞，订单创建等场景。

唯一索引不仅可以解决并发下的脏数据问题，也可以解决永久性幂等的问题。

缺点： 无法适用分布式存储系统，需要维护数据库的唯一索引，多的情况下不容易管理

分布式锁

如果是分布式系统，全局的唯一索引就很难构建，此时可以使用分布式锁的方式解决此类问题。我们以在执行第操作时先获取分布式锁，做完操作后，再将分布式锁释放。这样可以解决高并发性下的幂性问题。

```
@Controller
String addOrder(order){
    redisLock.lock();
    addOrder(order);
    redisLock.unLock();
}
```

需要配合select+insert使用，如果不配合，无法解决表单重复提交的问题。

token机制

token机制是一种比较常用的机制，核心原理在于给每个操作执行前，需要去服务中获取一个token执行操作时需要携带token进行操作。如果发现token存在，则使用token，并将其置为已使用，并执操作，执行完毕后并且会将token对应的执行结果存储起来。否则将会检查是否存在执行结果，直接出。

比如下单操作需要一次进行添加订单、更改库存、更改优惠券三个操作。每个操作执行前都去使用tokn验证该操作是否已经执行，从而防止重复执行的问题。并且，缓存的结果也可以用于事务控制（如下单失败，增加记录的库存和优惠券）。

```
Token getToken() {
```

```
    return tokenPool.getToken();
}
addOrder(order, token) {
    if (tokenPool.removeToken(token) == 1) { // 说明池中有token，token是有效的
        addOrder()
    } else {
        throw Exception;
    }
}

// 客户端调用
token = ajax.get("/token");
result = ajax.post("/addOrder", token， order)
```

注：这种方式也有缺点，使用token机制，那么就会意味着在需要保证幂等性的接口在被调用前，必先调用接口获取token

MVCC机制

MVCC(Multi-Version Concurrency Control) 多版本并发控制。在数据**更新**时需要去比较持有数据版本号，版本号不一致的操作无法进行更新，更新成功后版本号将会发生变化。

```
updateStock(int num, int version) {
    if (version != currentVersion) {
        throw Exception;
    } else {
        stockDao.minStock();
    }
}
```

注意，只适用于**更新接口**

状态机幂等

所谓状态机，就是任务或者业务在执行的过程中，拥有的状态以及状态的变更图。在执行某个操作前需要先对当前状态进行验证，如果状态不是该有的状态，则拒绝操作。

```
String pay() {
    if (order.status == "待支付") {
        doPay();
    }
}
```

我们也可以将请求放入到缓冲区中，并去除不符合状态的请求。

参考资料

- 简书-GameKing-幂等性浅谈
- CSDN-抽离的心-高并发下接口幂等性解决方案