链滴

如何保证接口的幂等性？常见的实现方案有哪些？

作者：jianzh5

原文链接：https://ld246.com/article/1593579328218

来源网站：链滴

幂等性问题是面试中常见的面试问题，也是分布式系统最常遇到的问题之一。在说幂等性之前，我们来看一种情况，假如老王在某电商平台进行购物，付款的时候不小心手抖了一下，连续点击了两次支，但此时服务器没做任何验证，于是老王账户里面的钱被扣了两次，这显然对当事人造成了一定的经损失，并且还会让用户丧失对平台的信任。而**幂等性问题**说的就是如何防止接口的重复无效请求。

看完本文你会了解到：什么是幂等性？如何保证接口的幂等性？

典型回答

幂等性最早是数学里面的一个概念，后来被用于计算机领域，用于表示任意多次请求均与一次请求执的结果相同，也就是说对于一个接口而言，无论调用了多少次，最终得到的结果都是一样的。比如以代码：

```
public class IdempotentExample {
    // 变量
    private static int count = 0;
    /**
     * 非幂等性方法
     */
    public static void addCount() {
        count++;
    }
    /**
     * 幂等性方法
     */
    public static void printCount() {
        System.out.println(count);
    }
}
```

对于变量 count 来说，如果重复调用 addCount() 方法的话，会一直累加 count 的值，因为 addCout() 方法就是非幂等性方法；而 printCount() 方法只是用来打印控制台信息的。因此，它无论调用多次结果都是一样的，所以它是幂等性方法。

知道了幂等性的概念，那如何保证幂等性呢？

幂等性的实现方案通常分为以下几类：

- 前端拦截
- 使用数据库实现幂等性
- 使用 JVM 锁实现幂等性
- 使用分布式锁实现幂等性

下面我们分别来看它们的具体实现过程。

1. 前端拦截

前端拦截是指通过 Web 站点的页面进行请求拦截，比如在用户点击完“提交”按钮后，我们可以把钮设置为不可用或者隐藏状态，避免用户重复点击。

核心的实现代码如下：

```
<script>
   function subCli(){
      // 按钮设置为不可用
      document.getElementById("btn_sub").disabled="disabled";
      document.getElementById("dv1").innerText = "按钮被点击了~";
   }
</script>
<body style="margin-top: 100px;margin-left: 100px;">
   <input id="btn_sub" type="button"  value=" 提 交 "  onclick="subCli()">
   <div id="dv1" style="margin-top: 80px;"></div>
</body>
```

但前端拦截有一个致命的问题，如果是懂行的程序员或者黑客可以直接绕过页面的 JS 执行，直接模请求后端的接口，这样的话，我们前端的这些拦截就不能生效了。因此除了前端拦截一部分正常的误作之外，后端的验证必不可少。

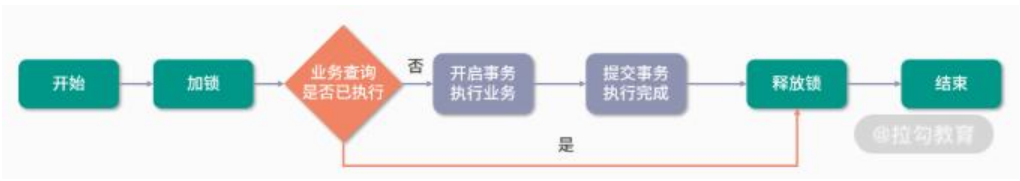
2. 数据库实现

数据库实现幂等性的方案有三个：

- 通过悲观锁来实现幂等性
- 通过唯一索引来实现幂等性
- 通过乐观锁来实现幂等性

3. JVM 锁实现

JVM 锁实现是指通过 JVM 提供的内置锁如 Lock 或者是 synchronized 来实现幂等性。使用 JVM 锁实现幂等性的一般流程为：首先通过 Lock 对代码段进行加锁操作，然后再判断此订单是否已经被处过，如果未处理则开启事务执行订单处理，处理完成之后提交事务并释放锁，执行流程如下图所示：

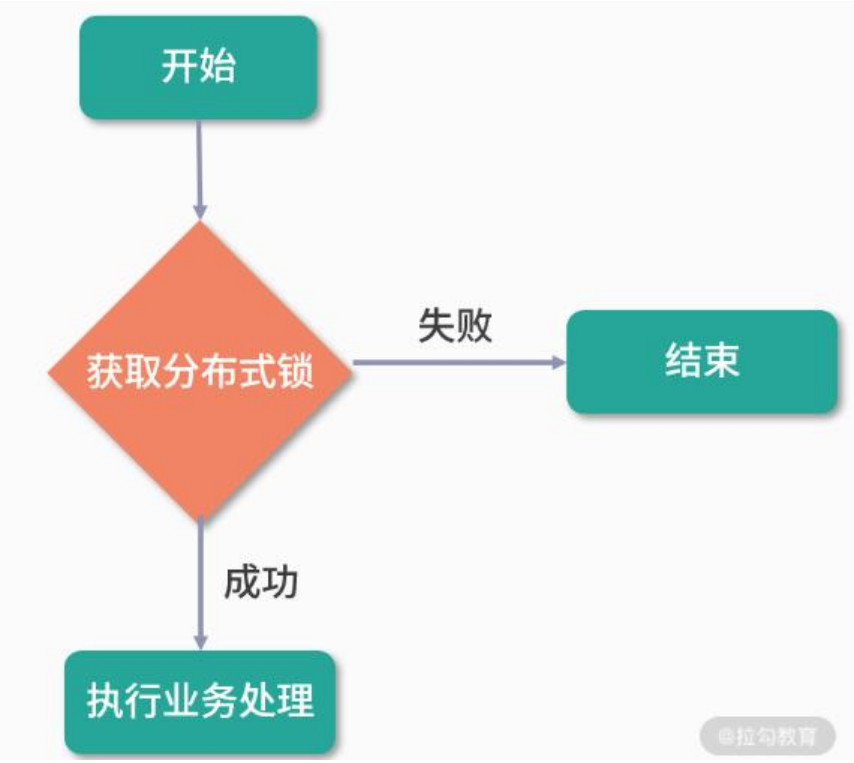


原文链接：如何保证接口的幂等性？常见的实现方案有哪些？

JVM 锁存在的最大问题在于，它只能应用于单机环境，因为 Lock 本身为单机锁，所以它就不适应于布式多机环境。

4. 分布式锁实现

分布式锁实现幂等性的逻辑是，在每次执行方法之前先判断是否可以获取到分布式锁，如果可以，则示为第一次执行方法，否则直接舍弃请求即可，执行流程如下图所示：



分布式锁执行流程图

需要注意的是分布式锁的 key 必须为业务的唯一标识，我们通常使用 Redis 或者 ZooKeeper 来实现布式锁；如果使用 Redis 的话，则用 set 命令来创建和获取分布式锁，执行示例如下：

```
127.0.0.1:6379> set lock true ex 30 nx
OK # 创建锁成功
```

其中，ex 是用来设置超时时间的；而 nx 是 not exists 的意思，用来判断键是否存在。如果返回的结为“OK”，则表示创建锁成功，否则表示重复请求，应该舍弃。更多关于 Reids 实现分布式的内容以查看第 20 课时的内容。

考点分析

幂等性问题看似“高大上”其实说白了就是如何避免重复请求提交的问题，出于安全性的考虑，我们须在前后端都进行幂等性验证，同时幂等性问题在日常工作中又特别常见，解决的方案也有很多，但虑到分布式系统情况，我们应该优先使用分布式锁来实现。

和此知识点相关的面试题还有以下这些：

- 幂等性需要注意什么问题？
- 实现幂等性的关键步骤有哪些？

- 说一说数据库实现幂等性的执行细节？

知识扩展

1. 幂等性注意事项

幂等性的实现与判断需要消耗一定的资源，因此不应该给每个接口都增加幂等性判断，要根据实际的务情况和操作类型来进行区分。例如，我们在进行查询操作和删除操作时就无须进行幂等性判断。查操作查一次和查多次的结果都是一致的，因此我们无须进行幂等性判断。删除操作也是一样，删除一和删除多次都是把相关的数据进行删除（这里的删除指的是条件删除而不是删除所有数据），因此也须进行幂等性判断。

2. 幂等性的关键步骤

实现幂等性的关键步骤分为以下三个：

- 每个请求操作必须有唯一的 ID，而这个 ID 就是用来表示此业务是否被执行过的关键凭证，例如，单支付业务的请求，就要使用订单的 ID 作为幂等性验证的 Key；
- 每次执行业务之前必须要先判断此业务是否已经被处理过；
- 第一次业务处理完成之后，要把此业务处理的状态进行保存，比如存储到 Redis 中或者是数据库中这样才能防止业务被重复处理。

3. 数据库实现幂等性

使用数据库实现幂等性的方法有三种：

- 通过悲观锁来实现幂等性
- 通过唯一索引来实现幂等性
- 通过乐观锁来实现幂等性

接下来我们分别来看这些实现方式的具体执行过程。

① 悲观锁

使用悲观锁实现幂等性，一般是配合事务一起来实现，在没有使用悲观锁时，我们通常的执行过程是样的，首先来判断数据的状态，执行 SQL 如下：

```
select status from table_name where id='xxx';
```

然后再进行添加操作：

```
insert into table_name (id) values ('xxx');
```

最后再进行状态的修改：

```
update table_name set status='xxx';
```

但这种情况因为是非原子操作，所以在高并发环境下可能会造成一个业务被执行两次的问题，当一个序在执行中时，而另一个程序也开始状态判断的操作。因为第一个程序还未来得及更改状态，所以第个程序也能执行成功，这就导致一个业务被执行了两次。

在这种情况下我们就可以使用悲观锁来避免问题的产生，实现 SQL 如下所示：

```
begin;  # 1.开始事务
select * from table_name where id='xxx' for update; # 2.查询状态
insert into table_name (id) values ('xxx'); # 3.添加操作
update table_name set status='xxx'; # 4.更改操作
commit; # 5.提交事务
```

在实现的过程中需要注意以下两个问题：

- 如果使用的是 MySQL 数据库，必须选用 innodb 存储引擎，因为 innodb 支持事务；
- id 字段一定要是主键或者是唯一索引，不然会锁表，影响其他业务执行。

② 唯一索引

我们可以创建一个唯一索引的表来实现幂等性，在每次执行业务之前，先执行插入操作，因为唯一字就是业务的 ID，因此如果重复插入的话会触发唯一约束而导致插入失败。在这种情况下（插入失败）们就可以判定它为重复提交的请求。

唯一索引表的创建示例如下：

```
CREATE TABLE `table_name` (
  `id` int NOT NULL AUTO_INCREMENT,
  `orderid` varchar(32) NOT NULL DEFAULT '' COMMENT '唯一id',
  PRIMARY KEY (`id`),
  UNIQUE KEY `uq_orderid` (`orderid`) COMMENT '唯一约束'
) ENGINE=InnoDB;
```

③ 乐观锁

乐观锁是指在执行数据操作时（更改或添加）进行加锁操作，其他时间不加锁，因此相比于整个执行程都加锁的悲观锁来说，它的执行效率要高很多。

乐观锁可以通过版本号来实现，例如以下 SQL：

```
update table_name set version=version+1 where version=0;
```

小结

幂等性不但可以保证程序正常执行，还可以杜绝一些垃圾数据以及无效请求对系统资源的消耗。本课我们讲了幂等性的 6 种实现方式，包括前端拦截、数据库悲观锁实现、数据唯一索引实现、数据库乐锁实现、JVM 锁实现，以及分布式锁的实现等方案，其中前端拦截无法防止懂行的人直接绕过前端进模拟请求的操作。因此后端一定要实现幂等性处理，推荐的做法是使用分布式锁来实现，这样的解决案更加通用。

文章来源：https://kaiwu.lagou.com/course/courseInfo.htm?courseId=59#/detail/pc?id=1791