bobby · 更新于 2022-11-16

上一节 16. grpc的超时... 1. 什么是链路追踪 下一节

哪些情况下需要考虑幂等性 - 同样的请求发送多次：

1. http请求的类型：
    1. get
        1. 获取商品信息， 这个会引起商品的数据的变化吗?
    2. post
        1. 比较常见，这种接口需要考虑到幂等性
    3. put
        1. 不一定要实现幂等性
            1. put 把1号商品的价格改为200，网络返回的时候抖动了，重试
            2. 第二次接口还是会把1号商品的价格改为200 - 这种情况下没有幂等的问题
        2. 出现幂等性问题的情况：
            1. 购物车中的商品，调用一次 这个商品的数量加一
                1. 第一次调用 原本的值 10 之后价格变为11 - 但是返回的时候出现了网络抖动
                2. 第二次发送 原本的值 11 之后价格变为12 - 但是返回的时候出现了网络抖动
                3. 第三次发送 原本的值 12 之后价格变为13- 但是返回的时候出现了网络抖动
    4. delete
        1. 一般不具备幂等性的要求
        2. 第一次调用 删除数据
        3. 第二次调用 还是删除当前的数据

现在假设，你自己开发了一个支付宝，然后别人在二维码支付页面，先支付了，结果由于网络问题，当前页面一直没有刷新，就让人以为我没有支付，然后我就再次扫码支付，这下完蛋了：你扣了两次款。

一、背景

**我们实际系统中有很多操作，是不管做多少次，都应该产生一样的效果或返回一样的结果。 **
get请求一般没有幂等性需求、delete请求一般也没有幂等性需求，post、update视情况而定**例如：

2. 我们发起一笔付款请求，应该只扣用户账户一次钱，当遇到网络重发或系统bug重发，也应该只扣一次钱；
3. 发送消息，也应该只发一次，同样的短信发给用户，用户会哭的；
4. 创建业务订单，一次业务请求只能创建一个，创建多个就会出大问题。

等等很多重要的情况，这些逻辑都需要幂等的特性来支持。

二、幂等性概念

幂等（idempotent、idempotence）是一个数学与计算机学概念，常见于抽象代数中。

在编程中.一个幂等操作的特点是其任意多次执行所产生的影响均与一次执行的影响相同。幂等函数，或幂等方法，是指可以使用相同参数重复执行，并能获得相同结果的函数。这些函数不会影响系统状态，也不用担心重复执行会对系统造成改变。例如，“getUsername()和setTrue()”函数就是一个幂等函数.

更复杂的操作幂等保证是利用唯一交易号(流水号)实现.

我的理解：幂等就是一个操作，不论执行多少次，产生的效果和返回的结果都是一样的

三、技术方案

1. 查询操作
   查询一次和查询多次，在数据不变的情况下，查询结果是一样的。select是天然的幂等操作

2. 删除操作
   删除操作也是幂等的，删除一次和多次删除都是把数据删除。(注意可能返回结果不一样，删除的数据不存在，返回0，删除的数据多条，返回结果多个)

1.唯一索引，防止新增脏数据

比如：新建用户的时候将手机号码设置为唯一索引，那么即使你重试，也只会新建一个用户，不会因为重试导致当前用户注册了两个用户

要点：
唯一索引或唯一组合索引来防止新增数据存在脏数据
（当表存在唯一索引，并发时新增报错时，再查询一次就可以了，数据应该已经存在了，返回结果即可**）**

2. 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：

1. 数据提交前要向服务的申请token，token放到redis或内存，token有效时间
2. 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

注意：redis要用删除操作来判断token，删除成功代表token校验通过，如果用select+delete来校验token，存在并发问题，不建议使用

3. 悲观锁

获取数据的时候加锁获取

select * from table_xxx where id='xxx' for update;

注意：id字段一定是主键或者唯一索引，不然是锁表，会死人的

悲观锁使用时一般伴随事务一起使用，数据锁定时间可能会很长，根据实际情况选用

4. 乐观锁

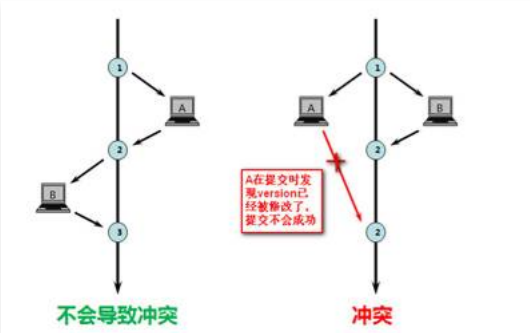
乐观锁只是在更新数据那一刻锁表，其他时间不锁表，所以相对于悲观锁，效率更高。

乐观锁的实现方式多种多样可以通过version或者其他状态条件：

1. 通过版本号实现

update table_xxx set name=#name#,version=version+1 where version=#version#

如下图(来自网上)：



2. 通过条件限制

   update table_xxx set avai_amount=avai_amount-#subAmount# where avai_amount-#subAmount# >= 0

   要求：quality-#subQuality# >= ，这个情景适合不用版本号，只更新是做数据安全校验，适合库存模型，扣份额和回滚份额，性能更高

   注意：乐观锁的更新操作，最好用主键或者唯一索引来更新,这样是行锁，否则更新时会锁表，上面两个sql改成下面的两个更好

   update table_xxx set name=#name#,version=version+1 where id=#id# and version=#version#

   update table_xxx set avai_amount=avai_amount-#subAmount# where id=#id# and avai_amount-#subAmount# >= 0

5. 分布式锁

还是拿插入数据的例子，如果是分布是系统，构建全局唯一索引比较困难，例如唯一性的字段没法确定，这时候可以引入分布式锁，通过第三方的系统(redis或zookeeper)，在业务系统插入数据或者更新数据，系统，也就

要点：某个长流程处理过程要求不能并发执行，可以在流程执行之前根据某个标志(用户ID+后缀等)获取分布式锁，其他流程执行时获取锁就会失败，也就是同一时间该流程只能有一个能执行成功，执行完成后，释放分布式锁(分布式锁要第三方系统提供)

6. 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

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)

重点：
对外提供接口为了支持幂等调用，接口有两个字段必须传，一个是来源source，一个是来源方序列号seq，这个两个字段在提供方系统里面做联合唯一索引，这样当第三方调用时，先在本方系统里面查询一下，是否已经处理过，返回相应处理结果；没有处理过，进行相应处理，返回结果。注意，为了幂等友好，一定要先查询一下，是否处理过该笔业务，不查询直接插入业务系统，会报错，但实际已经处理了。

总结：

幂等与你是不是分布式高并发没有关系。关键是你的操作是不是幂等的。一个幂等的操作典型如：把编号为5的记录的A字段设置为0这种操作不管执行多少次都是幂等的。一个非幂等的操作典型如：把编号为5的记录的A字段增加1这种操作显然就不是幂等的。要做到幂等性，从接口设计上来说不设计任何非幂等的操作即可。譬如说需求是：当用户点击赞同时，将答案的赞同数量+1。改为：当用户点击赞同时，确保答案赞同表中存在一条记录，用户、答案。赞同数量由答案赞同表统计出来。总之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。

16. grpc的超时和重试 上一节 下一节 1. 什么是链路追踪

我要提出意见反馈