94-项目实战二：设计实现一个通用的接口幂等框架（设计）

上一节课，我们介绍了幂等框架的一个重要需求场景，接口超时重试。为了避免同一业务被多次重复执行，接口需要支持幂等特性。同时，我们还对功能性需求和非功能性需求做了梳理。今天，我们来讲解幂等框架的设计思路。

跟限流框架类似，幂等框架的功能性需求也比较简单，但要考虑处理的异常情况有很多，比如业务代码异常、业务系统宕机、幂等框架异常。今天，我们重点讲解如何应对这些异常情况，设计一个高度容错的幂等框架。

话不多说，让我们正式开始今天的学习吧！

幂等处理正常流程

调用方从发起接口请求到接收到响应，一般要经过三个阶段。第一个阶段是调用方发送请求并被实现方接收，第二个阶段是执行接口对应的业务逻辑，第三个阶段是将执行结果返回给调用方。为了实现接口幂等，我们需要将幂等相关的逻辑，添加在这三个阶段中。

正常情况下，幂等号随着请求传递到接口实现方之后，接口实现方将幂等号解析出来，传递给幂等框架。幂等框架先去数据库（比如Redis）中查找这个幂等号是否已经存在。如果存在，说明业务逻辑已经或者正在执行，就不要重复执行了。如果幂等号不存在，就将幂等号存储在数据库中，然后再执行相应的业务逻辑。

正常情况下，幂等处理流程是非常简单的，难点在于如何应对异常情况。在这三个阶段中，如果第一个阶段出现异常，比如发送请求失败或者超时，幂等号还没有记录下来，重试请求会被执行，符合我们的预期。如果第三个阶段出现异常，业务逻辑执行完成了，只是在发送结果给调用方的时候，失败或者超时了，这个时候，幂等号已经记录下来，重试请求不会被执行，也符合我们的预期。也就是说，第一、第三阶段出现异常，上述的幂等处理逻辑都可以正确应对。

但是，如果第二个阶段业务执行的过程出现异常，处理起来就复杂多了。接下来，我们就看下幂等框架该如何应对这一阶段的各种异常。我分了三类异常来讲解，它们分别是业务代码异常、业务系统宕机、幂等框架异常。

业务代码异常处理

当业务代码在执行过程中抛出异常的时候，我们是否应该认定为业务处理失败，然后将已经记录的幂等号删除，允许重新执行业务逻辑呢？

对于这个问题，我们要分业务异常和系统异常来区分对待。那什么是业务异常？什么是系统异常呢？我举个例子解释一下。比如，A用户发送消息给B用户，但是查询B用户不存在，抛出UserNotExisting异常，我们把这种业务上不符合预期叫做业务异常。因为数据库挂掉了，业务代码访问数据库时，就会报告数据库异常，我们把这种非业务层面的、系统级的异常，叫做系统异常。

遇到业务异常（比如UserNotExisting异常），我们不删除已经记录的幂等号，不允许重新执行同样的业务逻辑，因为再次重新执行也是徒劳的，还是会报告异常。相反，遇到系统异常（比如数据库访问异常），我们将已经记录的幂等号删除，允许重新执行这段业务逻辑。因为在系统级问题修复之后（比如数据库恢复了），重新执行之前失败的业务逻辑，就有可能会成功。

实际上，为了让幂等框架尽可能的灵活，低侵入业务逻辑，发生异常（不管是业务异常还是系统异常），是否允许再重试执行业务逻辑，交给开发这块业务的工程师来决定是最合适的了，毕竟他最清楚针对每个异常该如何处理。而幂等框架本身不参与这个决定，它只需要提供删除幂等号的接口，由业务工程师来决定遇到异常的时候，是否需要调用这个删除接口，删除已经记录的幂等号。

业务系统宕机处理

刚刚分析的是代码异常，我们再来看下，如果在业务处理的过程中，业务系统宕机了（你可以简单理解为部署了业务系统的机器宕机了），幂等框架是否还能正确工作呢？

如果幂等号已经记录下了，但是因为机器宕机，业务还来得及执行，按照刚刚的幂等框架的处理流程，即便机器重启，业务也不会再被触发执行了，这个时候该怎么办呢？除此之外，如果记录幂等号成功了，但是在捕获到系统异常之后，要删除幂等号之前，机器宕机了，这个时候又该怎么办？

如果希望幂等号的记录和业务的执行完全一致，我们就要把它们放到一个事务中。执行成功，必然会记录幂等号；执行失败，幂等号记录也会被自动回滚。因为幂等框架和业务系统各自使用独立的数据库来记录数据，所以，这里涉及的事务属于分布式事务。如果为了解决这个问题，引入分布式事务，那幂等框架的开发难度提高了很多，并且框架使用起来也复杂了很多，性能也会有所损失。

针对这个问题，我们还有另外一种解决方案。那就是，在存储业务数据的业务数据库（比如MySQL）中，建一张表来记录幂等号。幂等号先存储到业务数据库中，然后再同步给幂等框架的Redis数据库。这样做的好处是，我们不需要引入分布式事务框架，直接利用业务数据库本身的事务属性，保证业务数据和幂等号的写入操作，要么都成功，要么都失败。不过，这个解决方案会导致幂等逻辑，跟业务逻辑没有完全解耦，不符合我们之前讲到的低侵入、松耦合的设计思想。

实际上，做工程不是做理论。对于这种极少发生的异常，在工程中，我们能够做到，在出错时能及时发现问题、能够根据记录的信息人工修复就可以了。虽然看起来解决方案不优雅，不够智能，不够自动化，但是，这比编写一大坨复杂的代码逻辑来解决，要好使得多。所以，我们建议业务系统记录SQL的执行日志，在日志中附加上幂等号。这样我们就能在机器宕机时，根据日志来判断业务执行情况和幂等号的记录是否一致。

幂等框架异常处理

我们前面提到，限流框架本身的异常，不能导致接口响应异常。那对于幂等框架来说，是否也适用这条设计原则呢？

对于限流来说，限流框架执行异常（比如，Redis访问超时或者访问失败），我们可以触发服务降级，让限流功能暂时不起作用，接口还能正常执行。如果大量的限流接口调用异常，在具有完善监控的情况下，这些异常很快就会被运维发现并且修复，所以，短暂的限流失效，也不会对业务系统产生太多影响。毕竟限流只是一个针对突发情况的保护机制，平时并不起作用。如果偶尔的极个别的限流接口调用异常，本不应该被放过的几个接口请求，因为限流的暂时失效被放过了，对于这种情况，绝大部分业务场景都是可以接受的。毕竟限流不可能做到非常精确，多放过一两个接口请求几乎没影响。

对于幂等来说，尽管它应对的也是超时重试等特殊场景，但是，如果本不应该重新执行的业务逻辑，因为幂等功能的暂时失效，被重复执行了，就会导致业务出错（比如，多次执行转账，钱多转了）。对于这种情况，绝大部分业务场景都是无法接受的。所以，在幂等逻辑执行异常时，我们选择让接口请求也失败，相应的业务逻辑就不会被重复执行了。毕竟接口请求失败（比如转钱没转成功），比业务执行出错（比如多转了钱），修复的成本要低很多。

重点回顾

好了，今天的内容到此就讲完了。我们一块来总结回顾一下，你需要重点掌握的内容。

今天，我们讲解了幂等框架的设计思路。在正常情况下，幂等框架的处理流程是比较简单的，调用方生成幂等号，传递给实现方，实现方记录幂等号或者用幂等号判重。但是，幂等框架要处理的异常情况很多，这也是设计的复杂之处和难点之处。

我们针对三种不同类型的异常，讲解了幂等框架的应对思路。

对于业务代码异常，为了让幂等框架尽可能的灵活，低侵入业务逻辑，发生异常（不管是业务异常还是系统异常），是否允许再重试执行业务逻辑，交给开发这块业务的工程师来决定。

对于业务系统宕机，对于这种极少发生的异常，在工程中，我们能够做到，在出错时能及时发现问题、能够根据记录的信息人工修复，就可以了。所以，我们建议业务系统记录SQL的执行日志，在日志中附加上幂等号。这样我们就能在机器宕机时，根据日志来判断业务执行情况和幂等号的记录是否一致。

对于幂等框架异常，跟限流框架异常处理对策不同，在幂等逻辑执行异常时，我们选择让接口请求也失败，相应的业务逻辑就不会被重复执行了，业务就不会出错。毕竟接口请求失败，比业务执行出错，修复的成本要低很多。

虽然幂等框架要处理的异常很多，但考虑到开发成本以及简单易用性，我们对某些异常的处理在工程上做了妥协，交由业务系统或者人工介入处理。这样就大大简化了幂等框架开发的复杂度和难度。

课堂讨论

我常说，异常情况考虑是否全面，处理是否得当，很能体现一个程序员的逻辑思维能力、工程能力。除了我们今天讲到的异常，在幂等框架中，你还能想到有哪些其他异常情况会发生？又该如何应对呢？

欢迎留言和我分享你的想法。如果有收获，也欢迎你把这篇文章分享给你的朋友。

精选留言：

- 有铭 2020-06-08 08:26:22
  幂等框架是宁可错杀，不可放过，放过了（多执行）修复难度太大，错杀了无非是再执行一次 [4赞]

- Jxin 2020-06-08 13:25:46
  1.这教的不止是设计模式。还包含了设计过程中的权衡取舍。看栏主的专栏，就像个小迷弟一样，每篇都是666打call。

  2.针对今天的专栏延伸一个问题。我们知道rpc接口一般都是result返回的模式。然后系统的异常可以分为业务异常和技术异常，业务异常一般重试解决不了，需要主动告警人工介入，技术异常则往往需要重试，自动化解决。这就导致result中的code往往有多个值，用于区分业务异常和技术异常走相应的逻辑。

  但是我觉得这很难受，我希望result的code只有0和1，仅用于表示是否发生了技术异常，甚至我希望没有result这种返回模式（rpc框架层面处理技术异常），毕竟rpc技术就是让开发者像调用本地接口一样调用远程接口。但是如果code只表示技术异常，那么属于业务异常的标记和异常消息就只能放在data中，这又让接口返回数据和业务异常耦合了。当然，我们也可以不捕捉业务异常，让它在调用侧抛出，这更贴合像调用本地方法一样调用远程方法的理念，但这样站在服务方的角度，最外层的接口都没有处理异常，又

显得不合适了。

所以，请问栏主和各位同学，rpc接口的返回应该怎么设计合理，你们又是怎么实践的？

3.回答课后题。比如某个策略依赖配置表，由于配置信息缓存延迟，发生了业务异常。这时候异常是业务异常，但引起异常的原因是缓存延迟这个技术问题。重试可以走通流程，但需要人工介入刷新缓存，或则等待缓存刷新。 [2赞]

- Heaven 2020-06-08 11:08:33
  框架本身在设计的时候,尽可能的考虑到相关的异常问题,一般都够触发项目的异常的,还是启动时候因为配置文件设置错误导致的异常问题,这就需要对配置文件的正确性进行校验
  而且,对于幂等号,我们是否可以设置过期时间,来方便项目组件宕机后重启,直接使用,不用手动删除幂等号了 [1赞]