93 | 项目实战二：设计实现一个通用的接口幂等框架（分析）

王争 · 设计模式之美

上三节课，我带你分析、设计、实现了一个接口限流框架。在分析阶段，我们讲到需求分析的两大方面，功能性需求分析和非功能性需求分析。在设计阶段，我们讲了如何通过合理的设计，在实功能性需求的前提下，满足易用、易扩展、灵活、高性能、高容错等非功能性需求。在实现阶段，我们讲了如何利用设计思想、原则、模式、编码规范等，编写可读、可扩展等高质量的代码实现。

从今天开始，我们来实战一个新的项目，开发一个通用的接口幂等框架。跟限流框架一样，我们还是分为分析、设计、实现三个部分，对应三节课来讲解。

话不多说，让我们正式开始今天的学习吧！

需求场景

我们先来看下幂等框架的需求场景。

还记得之前讲到的限流框架的项目背景吗？为了复用代码，我们把通用的功能设计成了公共服务平台。公司内部的其他金融产品的后台系统，会调用公共服务平台的服务，不需要完全从零开始开发。公共服务平台提供的是 RESTful 接口。为了简化开发，调用方一般使用 Feign 框架（一个 HTTP 框架）来访问公共服务平台的接口。

调用方访问公共服务平台的接口，会有三种可能的结果：成功、失败和超时。前两种结果非常明确，调用方可以自己决定收到结果之后如何处理。结果为“成功”，万事大吉。结果为“失败”，一般情况下，调用方会将失败的结果，反馈给用户（移动端 App），让用户自行决定是否重试。

但是，当接口请求超时时，处理起来就没那么容易了。有可能业务逻辑已经执行成功了，只是公共服务平台返回结果给调用方的时候超时了，但也有可能业务逻辑没有执行成功，比如，因为数据库当时存在集中写入，导致部分数据写入超时。总之，超时对应的执行结果是未决的。那调用方调用接口超时时（基于 Feign 框架开发的话，一般是收到 Timeout 异常），该如何处理呢？

如果接口只包含查询、删除、更新这些操作，那接口天然是幂等的。所以，超时之后，重新再执行一次，也没有任何副作用。不过，这里有两点需要特殊说明一下。

删除操作需要当心 ABA 问题。删除操作超时了，又触发一次删除，但在这次删除之前，又有一次新的插入。后一次删除操作删除了新插入的数据，而新插入的数据本不应该删除。不过，大部分业务都可以容忍 ABA 问题。对于少数不能容忍的业务场景，我们可以针对性的特殊处理。

除此之外，细究起来，update x = x+delta 这样格式的更新操作并非幂等，只有 update x=y 这样格式的更新操作才是幂等的。不过，后者也存在跟删除同样的 ABA 问题。

如果接口包含修改操作（插入操作、update x=x+delta 更新操作），多次重复执行有可能会导致业务上的错误，这是不能接受的。如果插入的数据包含数据库唯一键，可以利用数据库唯一键的排他性，保证不会重复插入数据。除此之外，一般我会建议调用方按照这样几种方式来处理。

第一种处理方式是，调用方访问公共服务平台接口超时时，返回清晰明确的提醒给用户，告知执行结果未知，让用户自己判断是否重试。不过，你可能会说，如果用户看到了超时提醒，但还是重新发起了操作，比如重新发起了转账、充值等操作，那该怎么办呢？实际上，对这种情况，技术是无能为力的。因为两次操作都是用户主动发起的，我们无法判断第二次的转账、充值是新的操作，还是基于上一次超时的重试行为。

第二种处理方式是，调用方调用其他接口，来查询超时操作的结果，明确超时操作对应的业务，是执行成功了还是失败了，然后再基于明确的结果做处理。但是这种处理方法存在一个问题，那就是并不是所有的业务操作，都方便查询操作结果。

第三种处理方式是，调用方在遇到接口超时之后，直接发起重试操作。这样就需要接口支持幂等。我们可以选择在业务代码中触发重试，也可以将重试的操作放到 Feign 框架中完成。因为偶尔发生的超时，在正常的业务逻辑中编写一大坨补救代码，这样做会影响到代码的可读性，有点划不来。当然，如果项目中需要支持超时重试的业务不多，那对于仅有几个业务，特殊处理一下也未尝不可。但是，如果项目中需要支持超时重试的业务比较多，我们最好是把超时重试这些非业务相关的逻辑，统一在框架层面解决。

对响应时间敏感的调用方来说，它们服务的是移动端的用户，过长的等待时间，还不如直接返回超时给用户。所以，这种情况下，第一种处理方式是比较推荐的。但是，对响应时间不敏感的调用方来说，比如 Job 类的调用方，我推荐选择后两种处理方式，能够提高处理的成功率。而第二种处理方法，本身有一定的局限性，因为并不是所有业务操作都方便查询是否执行成功。第三种保证接口幂等的处理方式，是比较通用的解决方案。所以，我们针对这种处理方式，抽象出一套统一的幂等框架，简化幂等接口的开发。

需求分析

刚刚我们介绍了幂等框架的需求背景：超时重试需要接口幂等的支持。接下来，我们再对需求进行更加详细的分析和整理，这其中就包括功能性需求和非功能性需求。

不过，在此之前，我们需要先搞清楚一个重要的概念：幂等号。

前面多次提到“幂等”，那“幂等”到底是什么意思呢？放到接口调用的这个场景里，幂等的意思是，针对同一个接口，多次发起同一个业务请求，必须保证业务只执行一次。那如何判定两次

接口请求是同一个业务请求呢？也就是说，如何判断两次接口请求是重试关系？而非独立的两个业务请求？比如，两次调用转账接口，尽管转账用户、金额等参数都一样，但我们也无法判断这两个转账请求就是重试关系。

实际上，要确定重试关系，我们就需要给同一业务请求一个唯一标识，也就是“幂等号”！如果两个接口请求，带有相同的幂等号，那我们就判断它们是重试关系，是同一个业务请求，不要重复执行。

幂等号需要保证全局唯一性。它可以有业务含义，比如，用户手机号码是唯一的，对于用户注册接口来说，我们可以拿它作为幂等号。不过，这样就会导致幂等框架的实现，无法完全脱离具体的业务。所以，我们更加倾向于，通过某种算法来随机生成没有业务含义的幂等号。

幂等号的概念搞清楚了，我们再来看下框架的功能性需求。

前面也介绍了一些需求分析整理方法，比如画线框图、写用户用例、基于测试驱动开发等。跟限流框架类似，这里我们也借助用户用例和测试驱动开发的思想，先去思考，如果框架最终被开发出来之后，它会如何被使用。我写了一个框架使用的 Demo 示例，如下所示。

复制代码

```
//////// 使用方式一: 在业务代码中处理幂等 ////////////
// 接口调用方
Idempotence idempotence = new Idempotence();
String idempotenceId = idempotence.createId();
Order order = createOrderWithIdempotence(..., idempotenceId);

// 接口实现方
public class OrderController {
  private Idempotence idempontence; // 依赖注入

  public Order createOrderWithIdempotence(..., String idempotenceId) {
    // 前置操作
    boolean existed = idempotence.check(idempotenceId);
    if (existed) {
      // 两种处理方式:
      // 1. 查询order，并且返回;
      // 2. 返回duplication operation Exception
    }
    idempotence.record(idempotenceId);

```

```
    //...执行正常业务逻辑
  }

  public Order createOrder(...) {
    //...
  }
}

//////////// 使用方式二：在框架层面处理幂等 ///////////////
// 接口调用方
Idempotence idempotence = new Idempotence();
String idempotenceId = idempotence.createId();
//...通过feign框架将幂等号添加到http header中...

// 接口实现方
public class OrderController {
  @IdempotenceRequired
  public Order createOrder(...) {
    //...
  }
}

// 在AOP切面中处理幂等
@Aspect
public class IdempotenceSupportAdvice {
  @Autowired
  private Idempotence idempotence;

@Pointcut("@annotation(com.xzg.cd.idempotence.annotation.IdempotenceRequired)")
  public void controllerPointcut() {
  }

  @Around(value = "controllerPointcut()")
  public Object around(ProceedingJoinPoint joinPoint) throws Throwable {
    // 从HTTP header中获取幂等号idempotenceId

    // 前置操作
    boolean existed = idempotence.check(idempotenceId);
    if (existed) {
      // 两种处理方式：
      // 1. 查询order，并且返回；
      // 2. 返回duplication operation Exception
    }
    idempotence.record(idempotenceId)

    Object result = joinPoint.proceed();
    return result;
  }
}
```

结合刚刚的 Demo，从使用的角度来说，幂等框架的主要处理流程是这样的。接口调用方生成幂等号，并且跟随接口请求，将幂等号传递给接口实现方。接口实现方接收到接口请求之后，按照约定，从 HTTP Header 或者接口参数中，解析出幂等号，然后通过幂等号查询幂等框架。如果幂等号已经存在，说明业务已经执行或正在执行，则直接返回；如果幂等号不存在，说明业务没有执行过，则记录幂等号，继续执行业务。

对于幂等框架，我们再来看下，它都有哪些非功能性需求。

在易用性方面，我们希望框架接入简单方便，学习成本低。只需编写简单的配置以及少许代码，就能完成接入。除此之外，框架最好对业务代码低侵入松耦合，在统一的地方（比如 Spring AOP 中）接入幂等框架，而不是将它耦合在业务代码中。

在性能方面，针对每个幂等接口，在正式处理业务逻辑之前，我们都要添加保证幂等的处理逻辑。这或多或少地会增加接口请求的响应时间。而对于响应时间比较敏感的接口服务来说，我们要让幂等框架尽可能低延迟，尽可能减少对接口请求本身响应时间的影响。

在容错性方面，跟限流框架相同，不能因为幂等框架本身的异常，导致接口响应异常，影响服务本身的可用性。所以，幂等框架要有高度的容错性。比如，存储幂等号的外部存储器挂掉了，幂等逻辑无法正常运行，这个时候业务接口也要能正常服务才行。

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。

今天我们介绍了幂等框架的一个需求场景，那就是接口超时重试。大部分情况下，如果接口只包含查询、删除、更新这些操作，那接口天然是幂等的。除此之外，如果接口包含修改操作（插入操作或 update x=x+delta 更新操作），保证接口的幂等性就需要做一些额外的工作。

现在开源的东西那么多，但幂等框架非常少见。原因是幂等性的保证是业务强相关的。大部分保证幂等性的方式都是针对具体的业务具体处理，比如利用业务数据中的 ID 唯一性来处理插

入操作的幂等性。但是，针对每个需要幂等的业务逻辑，单独编写代码处理，一方面对程序员的开发能力要求比较高，另一方面开发成本也比较高。

为了简化接口幂等的开发，我们希望开发一套统一的幂等框架，脱离具体的业务，让程序员通过简单的配置和少量代码，就能将非幂等接口改造成幂等接口。

课堂讨论

1. 重试无处不在，比如，Nginx、Dubbo、Feign 等重试机制，你还能想到哪些其他的重试场景吗?
2. 超时重试只是接口幂等的一个需求场景。除此之外，处理消息队列中消息重复的一种常用方法，就是将消息对应的业务逻辑设计成幂等的。因为业务逻辑是幂等的，所以多次接收重复消息不会导致重复执行业务逻辑。除了这些场景，你还知道有哪些其他场景需要用到幂等设计?

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。

AI智能总结

本文介绍了如何设计实现一个通用的接口幂等框架。作者首先回顾了之前讲到的限流框架的项目背景，指出公司内部的其他金融产品的后台系统会调用公共服务平台的服务，而为了简化开发，调用方一般使用Feign框架来访问公共服务平台的接口。接着，文章详细分析了接口请求超时时的处理方式，特别是针对不同类型的接口操作，如查询、删除、更新和修改等，提出了相应的处理建议。作者还强调了对响应时间敏感的调用方和对响应时间不敏感的调用方在处理超时的不同方式。最后，作者提出了抽象出一套统一的幂等框架，简化幂等接口的开发，以解决超时重试这些非业务相关的逻辑。整篇文章通过实际的需求场景和具体的操作方式，为读者提供了设计实现通用接口幂等框架的思路和方法，对于需要处理接口幂等性的开发人员具有一定的指导意义。文章还介绍了幂等号的概念和功能性需求，以及框架的易用性、性能和容错性方面的需求。最后，文章提出了希望开发一套统一的幂等框架，脱离具体的业务，让程序员通过简单的配置和少量代码，就能将非幂等接口改造成幂等接口。

全部留言 (23)

最新　精选

J.Smile
2020-06-05

“如果幂等号已经存在，说明业务已经执行或正在执行，则直接返回；如果幂等号不存在，说明业务没有执行过，则记录幂等号，继续执行业务“
————————————————————————————————
这个判断存在与否也要保证原子性

作者回复: 判断是否存在+业务执行 要保持原子

共 5 条评论> 17

小喵喵
2020-06-05

1 如果存储幂等号的外部存储器里面的数据太多了，会影响查询性能，如何优化?
2 如果存储幂等挂掉了，幂等逻辑无法正常运行，那这个就相当于没有幂等了。这个时候咋搞呢?

作者回复: 1. 定期删一下很老的数据
2. 让接口报错，后面会讲到的

共 2 条评论> 13

skull
2020-06-18

幂等如果外部存储挂掉了，就不能让业务正常使用了吧，否则会出问题

作者回复: 是的，跟限流框架对异常的处理不同。后面两篇有解答。

3

Jxin
2020-06-05

1.当给代码分论别类习惯了。那业务代码和技术代码的耦合就挺扎眼，总想着分离，透明掉技术代码，保护业务代码的干净。

课后题
1.mq消费重试，网络丢包重试。。。
2.技术上的想不到。有重试的地方好像都要。

疑问
容错性这个有点不理解。限流这个不生效还好说。幂等功能不生效？刷数已经在路上。

作者回复: 哈哈，看后面的两篇文章，有解释的。

1

美美
2020-06-15

分布式系统中，有一台机器挂了，请求会打到另一台机器上，这个时候，第二台机器不知道第一台是什么情况，第一台的日志，我认为也没有什么价值了，除非服务有状态，某一个请求一直只打到一台服务器上，但这个设计就很复杂了，老师有别的建议吗？

作者回复: 没看懂你说的第一台机的日志没有价值的意思呢？两台机器大部分都是独立执行任务的吧，日志本来就不需要互通的吧

共 2 条评论>

aoe
2020-09-11

重试场景：
1. 微信发送消息失败，可以手动触发”重发“
2. 网吧电脑卡顿、死机，可以重启
3. 拨打电话无人接听，请稍后再拨
4. 秒杀没有抢到，下一轮再接再厉
5. 游戏通关后升级难度从头开始

需要用到幂等设计场景：
1. 下单时不要出现重复订单
2. 提款时千万别出Bug
3. 火箭发射时，按下点火按钮，只能点火一次
4. 造小宝宝的时候，自然界超强幂等

共 5 条评论>

37

cricket1981
2020-06-26

分布式事务需要用到幂等设计：at-least-once + 幂等 == exactly-once

15

小晏子
2020-06-05

1. RocketMQ支持消息失败定时重试。
2. 用到幂等设计的还有：1）用户短时间内多次点击提交，2）第三方平台接口以为异常多次异步回调。

9

，
2020-06-10

偏个题,最近刚好在做硬件设备的断点重连,场景是这样的:
在现场有wifi,所有的物联网设备通过wifi连接,有的设备在不断的移动过程中,可能移动到wifi范围之外,这个时候就需要支持断点重连,在设备进入wifi范围时重新建立连接,为此,有以下几种方案:
1.提高wifi信号的强度,减少连接断掉的几率
2.当前使用tcp连接,将其修改为udp连接,通过服务不断地向设备发包/设备向服务发包来保证,此时不需要保持连接,做法相对简单
3.定时重连,通过定时任务,每隔一段时间就重新建立连接
方案1不靠谱,毕竟现场的环境相当大,无法保证wifi能覆盖到每一个角落,成本也高
方案2无法使用,通过观察设备厂家提供的sdk,发现只提供tcp连接的调用方式,因此该方案无效
方案3感觉上可行,但是如果设备正在进行业务处理时重连,则必然会有负面影响,业务上有延迟,连接也有可能迟迟建立不起来
在上述思考之后,感觉在失去连接时重新建立连接即可,于是通过学习厂家的sdk,发现确实有提供连接断开之后的回调,于是现在的做法像是方案3的升级版,每当连接断掉后就将其加入重连队列,定时轮询队列,建立连接,连接成功后从队列中弹出,不成功则在下一轮定时任务中处理
　这一版本的断点重连稳定性很好,但是依旧存在问题,有人发现了吗?那么怎么解决这个问题呢?(提醒一下,迭代器)

7

麻薯不是年糕🍢
2021-11-16

我有个疑问。幂等号采用随机数生成，会不会这么个场景，比如下订单过程中由于网络波动，导致用户点了两次按钮 调用了两次接口，这种情况幂等号不是相当于生成两个不同的随机数吗？如果是这样，如果确保接口幂等性呢

3