弹力设计篇之“幂等性设计”

2018-03-01 陈皓

弹力设计篇之“幂等性设计”

陈皓

- 00:03 / 11:09

所谓幂等性设计，就是说，一次和多次请求某一个资源应该具有同样的副作用。用数学的语言来表达就是：f(x) = f(f(x))。

比如，求绝对值的函数，abs(x) = abs(abs(x))。

为什么我们需要这样的操作？说白了，就是在我们把系统解耦隔离后，服务间的调用可能会有三个状态，一个是成功（Success），一个是失败（Failed），一个是超时（Timeout）。前两者都是明确的状态，而超时则是完全不知道是什么状态。

比如，超时原因是网络传输丢包的问题，可能是请求时就没有请求到，也有可能是请求到了，返回结果时没有返回到等情况。于是我们完不知道下游系统是否收到了请求，而收到了请求是否处理了，成功/失败的状态在返回时是否遇到了网络问题。总之，请求方完全不知道是怎么回事。

举几个例子：

- 订单创建接口，第一次调用超时了，然后调用方重试了一次。是否会多创建一笔订单？
- 订单创建时，我们需要去扣减库存，这时接口发生了超时，调用方重试了一次。是否会多扣一次库存？
- 当这笔订单开始支付，在支付请求发出之后，在服务端发生了扣钱操作，接口响应超时了，调用方重试了一次。是否会多扣一次钱？

因为系统超时，而调用户方重试一下，会给我们的系统带来不一致的副作用。

在这种情况下，一般有两种处理方式。

- 一种是需要下游系统提供相应的查询接口。上游系统在timeout后去查询一下。如果查到了，就表明已经做了，成功了就不用做了，失败了就走失败流程。
- 另一种是通过幂等性的方式。也就是说，把这个查询操作交给下游系统，我上游系统只管重试，下游系统保证一次和多次的请求结果是一样的。

对于第一种方式，需要对方提供一个查询接口来做配合。而第二种方式则需要下游的系统提供支持幂等性的交易接口。

全局ID

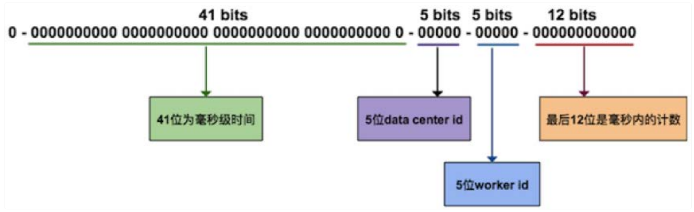
要做到幂等性的交易接口，需要有一个唯一的标识，来标志交易是同一笔交易。而这个交易ID由谁来分配是一件比较头疼的事。因为这个标识要能做到全局唯一。

如果由一个中心系统来分配，那么每一次交易都需要找那个中心系统来。 这样增加了程序的性能开销。如果由上游系统来分配，则可能会导致可能会出现分配ID重复了的问题。因为上游系统可能会是一个集群，它们同时承担相同的工作。

为了不产生分配冲突，我们需要使用一个不会冲突的算法，比如使用UUID这样冲突非常小的算法。但UUID的问题是，它的字符串占用的空间比较大，索引的效率非常低，生成的ID太过于随机，完全不是人读的，而且没有递增，如果要按前后顺序排序的话，基本不可能。

在全局唯一ID的算法中，这里介绍一个Twitter 的开源项目 Snowflake。它是一个分布式ID的生成算法。其核心思想是，产生一个long型的ID，其中：

- 41bits作为毫秒数。大概可以用69.7年。
- 10bits作为机器编号（5bits是数据中心，5bits的机器ID），支持1024个实例。
- 12bits作为毫秒内的序列号。一毫秒可以生成4096个序号。

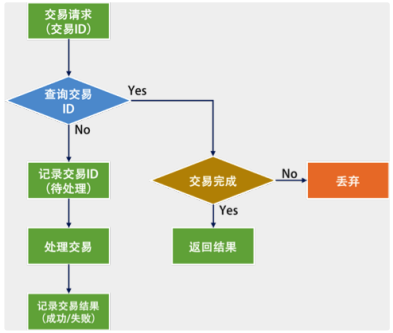


其他的像Redis或MongoDB的全局ID生成都和这个算法大同小异。我在这里就不多说了。你可以根据实际情况加上业务的编号。

处理流程

对于幂等性的处理流程来说，说白了就是要过滤一下已经收到的交易。要做到这个事，我们需要一个存储来记录收到的交易。

于是，当收到交易请求的时候，我们就会到这个存储中去查询。如果查找到了，那么就不再做查询了，并把上次做的结果返回。如果没有查到，那么我们就记录下来。



但是，上面这个流程有个问题。因为绝大多数请求应该都不会是重新发过来的，所以让100%的请求都到这个存储里去查一下，这会导致处理流程可能会很慢。

所以，最好是当这个存储出现冲突的时候会报错。也就是说，我们收到交易请求后，直接去存储里记录这个ID（相对于数据的Insert操作），如果出现ID冲突了的异常，那么我们就知道这个之前已经有人发过来了，所以就不用再做了。比如，数据库中你可以使用 `insert into … values … on DUPLICATE KEY UPDATE …` 这样的操作。

对于更新的场景来说，如果只是状态更新，可以使用如下的方式。如果出错，要么是非法操作，要么是已被更新，要么是状态不对，总之多次调用是不会有副作用的。

```
update table set status = “paid” where id = xxx and status = “unpaid”;
```

当然，网上还有MVCC通过使用版本号的方式，等等方式，我觉得这些都不标准，我们希望我们有一个标准的方式来做这个事，所以，最好还是用一个ID。

因为我们的幂等性服务也是分布式的，所以，需要这个存储也是共享的。这样每个服务就变成没有状态的了。但是，这个存储就成了一个非常关键的依赖，其扩展性和可用性也成了非常关键的指标。

你可以使用关系型数据库，或是key-value的NoSQL（如MongoDB）来构建这个存储系统。

HTTP的幂等性

HTTP GET方法用于获取资源，不应有副作用，所以是幂等的。比如：GET `http://www.bank.com/account/123456`，不会改变资源的状态，不论调用一次还是N次都没有副作用。请注意，这里强调的是一次和N次具有相同的副作用，而不是每次GET的结果相同。GET `http://www.news.com/latest-news`这个HTTP请求可能会每次得到不同的结果，但它本身并没有产生任何副作用，因而是满足幂等性的。

HTTP HEAD 和GET本质是一样的，区别在于HEAD不含有呈现数据，而仅仅是HTTP头信息，不应用有副作用，也是幂等的。有的人可能觉得这个方法没什么用，其实不是这样的。想象一个业务情景：欲判断某个资源是否存在，我们通常使用GET，但这里用HEAD则意义更加明确。也就是说，HEAD方法可以用来做探活使用。

HTTP OPTIONS 主要用于获取当前URL所支持的方法，所以也是幂等的。若请求成功，则它会在HTTP头中包含一个名为“Allow”的头，值是所支持的方法，如“GET, POST”。

HTTP DELETE方法用于删除资源，有副作用，但它应该满足幂等性。比如：DELETE `http://www.forum.com/article/4231`，调用一次和N次对系统产生的副作用是相同的，即删掉ID为4231的帖子。因此，调用者可以多次调用或刷新页面而不必担心引起错误。

HTTP POST方法用于创建资源，所对应的URI并非创建的资源本身，而是去执行创建动作的操作者，有副作用，不满足幂等性。比如：POST `http://www.forum.com/articles`的语义是在`http://www.forum.com/articles`下创建一篇帖子，HTTP响应中应包含帖子的创建状态以及帖子的URI。两次相同的POST请求会在服务器端创建两份资源，它们具有不同的URI；所以，POST方法不具备幂等性。

HTTP PUT方法用于创建或更新操作，所对应的URI是要创建或更新的资源本身，有副作用，它应该满足幂等性。比如：PUT `http://www.forum/articles/4231`的语义是创建或更新ID为4231的帖子。对同一URI进行多次PUT的副作用和一次PUT是相同的；因此，PUT方法具有幂等性。

所以，对于POST的方式，很可能会出现多次提交的问题，就好比，我们在论坛中发贴时，所遇到的有时候网络有问题对同一篇贴子出现多次提交的情况。对此的一般的幂等性的设计如下。

- 首先，在表单中需要隐藏一个token，这个token可以是前端生成的一个唯一的ID。用于防止用户多次点击了表单提交按钮，而导致后端收到了多次请求，却不能分辨是否是重复的提交。这个token是表单的唯一标识。（这种情况其实是通过前端生成ID把POST变成了PUT。）
- 然后，当用户点击提交后，后端会把用户提示的数据和这个token保存在数据库中。如果有重复提交，那么数据库中的token会做排它限制，从而做到幂等性。
- 当然，更为稳妥的做法是，后端成功后向前端返回302跳转，把用户的前端页跳转到GET请求，把刚刚POST的数据给展示出来。如果是Web上的最好还把之前的表单设置成过期，这样用户不能通过浏览器后退按钮来重新提交。这个模式又叫做 PRG模式（Post/Redirect/Get）。

小结

好了，我们来总结一下今天分享的主要内容。首先，幂等性的含义是，一个调用被发送多次所产生的总的副作用和被发送一次所产生的副作用是一样的。而服务调用有三种结果：成功、失败和超时，其中超时是我们需要解决的问题。

解决手段可以是超时后查询调用结果，也可以是在被调用的服务中实现幂等性。为了在分布式系统中实现幂等性，我们需要实现全局ID。Twitter的Snowflake就是一个比较好用的全局ID实现。最后，我给出了幂等性接口的处理流程。

下篇文章中，我们讲述服务的状态。希望对你有帮助。

也欢迎你分享一下你的分布式服务中所有交易接口是否都实现了幂等性？你所使用的全局ID算法又是什么呢？

文末给出了《分布式系统设计模式》系列文章的目录，希望你能在这个列表里找到自己感兴趣的内容。

- 弹力设计篇
  - 认识故障和弹力设计
  - 隔离设计Bulkheads
  - 异步通讯设计Asynchronous
  - 幂等性设计Idempotency
  - 服务的状态State
  - 补偿事务Compensating Transaction
  - 重试设计Retry
  - 熔断设计Circuit Breaker
  - 限流设计Throttle
  - 降级设计degradation
  - 弹力设计总结
- 管理设计篇
  - 分布式锁Distributed Lock
  - 配置中心Configuration Management
  - 边车模式Sidecar
  - 服务网格Service Mesh
  - 网关模式Gateway
  - 部署升级策略
- 性能设计篇
  - 缓存Cache
  - 异步处理Asynchronous
  - 数据库扩展
  - 秒杀Flash Sales
  - 边缘计算Edge Computing

macworks

2018-03-01

等幂性讲的很清楚

halfamonk

2018-03-11

私以为f(x) = f(f(x)) 这个数学公式表达幂等性不太对。因为f(f(x))应该是代表把f(x)的"结果"当作参数重新传入f()。这和文字的表述还是有区别的

作者回复

2018-03-11

谢谢回复，我理解你的意思。不过数学上的幂等的确是这样描述的。参看：https://en.wikipedia.org/wiki/Idempotence

幻想

2018-03-05

皓哥，这个专题能顺便说下分布式锁吗？我最近刚用db实现一个分布式锁。感觉不太满意。能否大概介绍一下这个主题？

邓呵呵

2018-04-27

以前对重复提交总觉得应该放前端实现，原来后端处理就是幂等性，受教了

酱了个油

2018-03-25

皓哥，由客户端如何生存唯一id呀，感觉twitter的算法适合服务器，有1024个服务器限制，可以给点提示吗

作者回复

2018-03-27

你什么集群？1024个不够？如果实在不够加几个bit给机器id吧

小倬

2018-06-16

没有理解，感觉应该用业务内容做id，比如参数，要不然下游超时，上游再次发送请求，生成的id是不同的

sonnyching

2018-05-09

我们现在的系统设计的时候都没考虑到这些��比如做幂等性接口的时候，下游每次收到订单都先查询一次，的确有点慢了。果然需要学习的地方还有很多呀。付费学习是值得的。

刘波3S

2018-04-13

这篇文章ID生成的讲解，解开了我一个长久的疑问，就这一段，付费199我也是愿意的。

作者回复

2018-04-19

谢谢

AlphaGo

2018-03-13

问题：上游（upstream）和下游（downstream）两个词是不是用反了？如果不是，这两个术语的在这篇文章上下文中的具体意思是什么？

作者回复

2018-03-22

有可能是我用错了。我的“上游”偏请求方，“下游”偏响应方。

YY

2018-03-06

重复交易过来后，返回上次交易信息，这个上次交易信息是不是需要存储起来，这个返回信息怎样存储比较好，是否有必要把所有的返回信息都存储起来

thomas

2018-03-02

你这里说的副作用是指什么？

作者回复

2018-03-02

你问的是哪句话的“副作用”？

特约嘉宾

2018-03-01

受教了，��

tiger

2018-07-10

我使用的幂等性ID是业务的组合键，类似数据库的联合唯一索引

---

neohope

2018-06-21

说到全局id的必要性，感触很深，比如皓哥在本文章里说到的幂等性；比如有了全局id，就能很便利的监控数据流向；再比如有了全局id，查看数据日志和排错会很方便，没有的话排错就很难。

在实际业务中，遇到了很多客户端重复提交的问题，造成的问题其实也很大。举一个很极端的例子，在一个项目中我们的系统替换另一个厂商的系统，新旧两个系统同时在线，有服务不断从旧系统搬数据到新系统。有一天系统性能下降比较严重，新系统变慢，用户就做了一件事情，把同一批数据，同时上传到了新系统和旧系统中，然后新系统提交的时候，产生了很多数据库提交主键冲突的问题，最终导致了丢失了一些数据，搞了一个通宵才完全恢复。然后才有精力开始排查性能下降的问题。

现在回头看一下，虽然问题比较极端，很多环节上出现了不该出现的操作，但当时我们的设计从来没有考虑去应对过这种极端情况，问题真的很大。

Jeff

2018-06-20

我们系统中公众号发布文章，就出现过幂等性的问题：运营人员在后台页面发布文章后，没有收到被调用方的回应，随后点击按钮发布了几次，结果客户端收到了几条同样的文章。后来是通过提前产生一个全局id，避免重复请求，来达成幂等性

颜忒妥

2018-06-09

使用snowflake 的话要配置machine id ，那如果用auto scale 的时候怎么自动配置呢？

作者回复

2018-06-11

一方面，你需要一个控制系统（这个系统是跑不掉的（想想CMDB），可以由它来分配），另一方面，可以用一些机器标识，如Mac地址，IP地址什么的。

Winter

2018-06-04

皓哥请教一下，对于创建 (create) 一类的动作 ，如果server侧发现资源已经存在，在幂等 性的设计里，是返回异常，还是正常返回已创 建资源的信息通常有特殊考虑么。有个例子是 上游已知订单号，要给订单创建一个关联的支 付交易(transaction)，交易可过期，可被替换。这种场景感觉总是返回当前可用的transaction信息对上游比较方便一点，不管这个transaction是本次请求创建的，还是之前已存在的未过期的。

张峰华

2018-05-19

有两个疑问，希望皓哥解答，1 前边说道“100% 的请求都到这个存储里去查一下，这会导致处理流程可能会很慢。” 后边说的标准的处理方式还是要存一个全局的id，这样的话每个请求还是要去查一下这个id是不是已经存在啊。
2 因为是下游服务存的全局id，当上游第一次提交超时，第二次再次提交的时候下游怎么判断是不是重复提交？ 第一次超时也不能够返回上游啊

小鱼儿

2018-05-15

有些疑惑，被调接口做幂等性处理，接口是不是要做aop包裹？

鱼的记忆

2018-05-14

请问一下，第一次请求因超时失败了，然后再次请求，怎么做到全局id是一样的？因为两次请求的时间点变化了。

作者回复

2018-05-14

重试的时候不用重新获取新的id，用上次的就好了。

Hua

2018-04-24

get不应有副作用所以幂等。我可以理解为因为没有副作用所以幂等。相当于有副作用所以不幂等？和你的delete和put解释有抵触。希望解答一下。看了文章还是没搞懂副作用和幂等的关系，或者说他们没有关系？

天真有邪

2018-03-16

处理流程那节我觉得有问题，首先你插曲如果存在报错，只能说明你收到了，并不能说明处理成功了，那如果出现存在，但是处理未成功，返回丢失了，你下次重试的时候怎么判断状态呢

作者回复

2018-03-22

有点没看懂

---

yunfeng

2018-03-06

之前碰到过类似的问题：前端多次调用后端接口。导致第一次是入库更新操作成功，后面几次由于前一次将调上游系统成功之后清除redis缓存了，导致后续的请求再次操失败。给用户提示不是很好，优化就是：不删除缓存，设置一个时效，用ID查，有则直接返回结果，否则继续后续的步骤。

---

null

2018-03-06

"小结"往上第二段：
然后，当用户点击提交后，后端会把用户提示[提交？]的数据和这个 token 保存在数据库中。

北极点

2018-03-05

HTTP的幂等设计讲的非常清楚，我之前一直分不清put 和 post方法的区别！在项目中一直就没用put方法！

曹林华

2018-03-03

如果进行重试请求，刚好上一次请求正在处理。我们的重试请求就会被丢弃掉，但是刚好上一次请求处理过程中遇到异常没有处理成功，请问还有什么补偿机制吗？

---

fishcat

2018-03-03

一次和多次请求某一个资源应该具有同样的副作用
这里说的副作用是什么意思？

作者回复

2018-03-08

所谓副作用，就是对这个资源
的变更所带来的连锁反应和影响。

whhbbq

2018-03-03

抱歉，我理解有误。我们的做法是相当于在上游防止重复提交。

作者回复

2018-03-08

上游防重，那需要在请求超时后做查询了。文章中也讲了这种情况

---

whhbbq

2018-03-02

通过请求头中带token和拦截器做防重，token不用存数据库

作者回复

2018-03-02

不太清楚你说的拦截器是什么东西？另外，如果不记录，怎么知道做没做？

姚利虎

2018-03-01

平时设计的时候很难会对一个服务考虑到这么细致，领教了！