15 | 限流器设计：如何避免超预期的高并发压力压垮系统？

2022-03-23 李智慧

讲述：李智慧

时长 16:00 大小 14.66M

你好，我是李智慧。

在互联网高可用架构设计中，限流是一种经典的高可用架构模式。因为某些原因，大量用户突然访问我们的系统时，或者有黑客恶意用 DoS（Denial of Service，拒绝服务）方式攻击我们的系统时，这种未曾预期的高并发访问对系统产生的负载压力可能会导致系统崩溃。

解决这种问题的一个主要手段就是限流，即拒绝部分访问请求，使访问负载压力降低到一个系统可以承受的程度。这样虽然有部分用户访问失败，但是整个系统依然是可用的，依然能对外提供服务，而不是因为负载压力太大而崩溃，导致所有用户都不能访问。

为此，我们准备开发一个限流器，产品名称为“Diana”。

需求分析

我们将 Diana 定位为一个限流器组件，即 Diana 的主要应用场景是部署在微服务网关或者其他 HTTP 服务器入口，以过滤器的方式对请求进行过滤，对超过限流规则的请求返回“服务不可用”HTTP 响应。

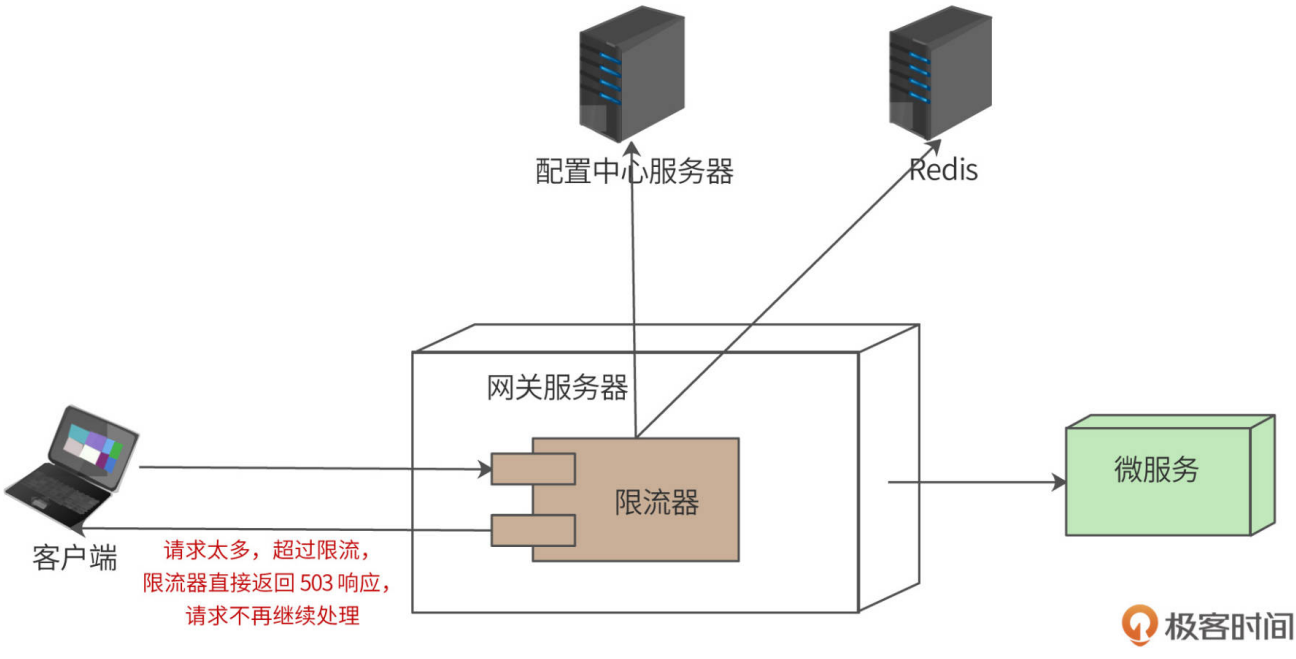
Diana 的限流规则可通过配置文件获取，并需要支持本地配置和远程配置两种方式，远程配置优先于本地配置。限流方式包括：

- 全局限流：针对所有请求进行限流，即保证整个系统处理的请求总数满足限流配置。
- 账号限流：针对账号进行限流，即对单个账号发送的请求进行限流。
- 设备限流：针对设备进行限流，即对单个客户端设备发送的请求进行限流。
- 资源限流：针对某个资源（即某个 URL）进行限流，即保证访问该资源的请求总数满足限流配置。

并且 Diana 设计应遵循开闭原则，能够支持灵活的限流规则功能扩展，即未来在不修改现有代码和兼容现有配置文件的情况下，支持新的配置规则。

概要设计

Diana 的设计目标是一个限流器组件，即 Diana 并不是一个独立的系统，不可以独立部署进行限流，而是部署在系统网关（或者其他 HTTP 服务器上），作为网关的一个组件进行限流，部署模型如下：



用户请求（通过负载均衡服务器）到达网关服务器。网关服务器本质也是一个 HTTP 服务器，限流器是部署在网关中的一个过滤器（filter）组件，和网关中的签名校验过滤器、用户权限过滤器等配置在同一个过滤器责任链（Chain of Responsibility）上。限流器应该配置在整个过滤器责任链的前端，也就是说，如果请求超过了限流，请求不需要再进入其他过滤器，直接被限流器拒绝。

用户请求进入限流器后，根据限流策略，判断该请求是否已经超过限流，如果超过，限流器直接返回状态码为 503（Too Many Requests）的响应；如果没有超过限流，请求继续向下处理（经过其他网关过滤器），并最终调用微服务完成处理。

限流器的策略可以在本地配置，也可以通过远程的配置中心服务器加载，即远程配置。远程配置优先于本地配置。

限流模式设计

请求是否超过限流，主要就是判断单位时间请求数量是否超过配置的请求限流数量。单位时间请求数量，可以本地记录，也可以远程记录。方便起见，本地记录称作本地限流，远程记录称作远程限流（也叫分布式限流）。

本地限流意味着，每个网关服务器需要根据本地记录的单位时间请求数量进行限流。假设限流配置为每秒限流 50 请求，如果该网关服务器本地记录的当前一秒内接受请求数量达到 50，那么这一秒内的后续请求都返回 503 响应。如果整个系统部署了 100 台网关服务器，每个网关配置本地限流为每秒 50，那么，整个系统每秒最多可以处理 5000 个请求。

远程限流意味着，所有网关共享同一个限流数量，每个网关服务器收到请求后，从远程服务器中获取单位时间内已处理请求数，如果超过限流，就返回 503 响应。也就是说，可能某个网关服务器一段时间内根本就没有请求到达，但是远程的已处理请求数已经达到了限流上限，那么这台网关服务器也必须拒绝请求。我们使用 Redis 作为记录单位时间请求数量的远程服务器。

高可用设计

为了保证配置中心服务器和 Redis 服务器宕机时，限流器组件的高可用。限流器应具有自动降级功能，即配置中心不可用，则使用本地配置；Redis 服务器不可用，则降级为本地限流。

详细设计

常用的限流算法有 4 种，固定窗口（Window）限流算法，滑动窗口（Sliding Window）限流算法，漏桶（Leaky Bucket）限流算法，令牌桶（Token Bucket）限流算法。我们将详细讨论这四种算法的实现。

此外，限流器运行期需要通过配置文件获取对哪些 URL 路径进行限流；本地限流还是分布式限流；对用户限流还是对设备限流，还是对所有请求限流；限流的阈值是多少；阈值的时间单位是什么；具体使用哪种限流算法。因此，我们需要先看下配置文件的设计。

配置文件设计

Diana 限流器使用 YAML 进行配置，配置文件举例如下：

```
Url:/
rules:
 - actor:device
   unit:second
   rpu:10
   algo:TB
   scope:global
 - actor:all
   unit:second
   rpu:50
   algo:W
   scope:local
```

配置文件的配置项有 7 种，分别说明如下：

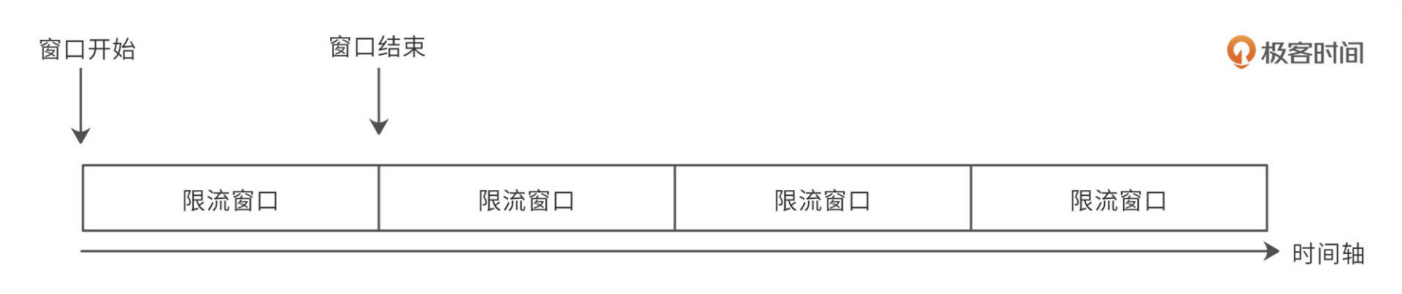
1. Url 记录限流的资源地址，"/“表示所有请求，配置文件中的路径可以互相包含，比如“/”包含“/sample”，限流器要先匹配“/”的限流规则，如果“/”的限流规则还没有触发（即访问”/"的流量，也就是单位时间所有的请求总和没有达到限流规则），则再匹配“/sample”。
2. 每个 Url 可以配置多个规则 rules，每个规则包括 actor，unit，rpu，algo，scope
3. actor 为限流对象，可以是账号（actor），设备（device），全部（all）
4. unit 为限流时间单位，可以是秒（second），分（minute），时（hour），天（day）
5. rpu 为单位时间限流请求数（request per unit），即上面 unit 定义的单位时间内允许通过的请求数目，如 unit 为 second，rpu 为 100，表示每秒允许通过 100 个请求，每秒超过 100 个请求就进行限流，返回 503 响应

6. scope 为 rpu 生效范围，可以是本地（local），也可以是全局（global），scope 也决定了单位时间请求数量是记录在本地还是远程，local 记录在本地，global 记录在远程。
7. algo 限流算法，可以是 window，sliding window，leaky bucket，token bucket 。

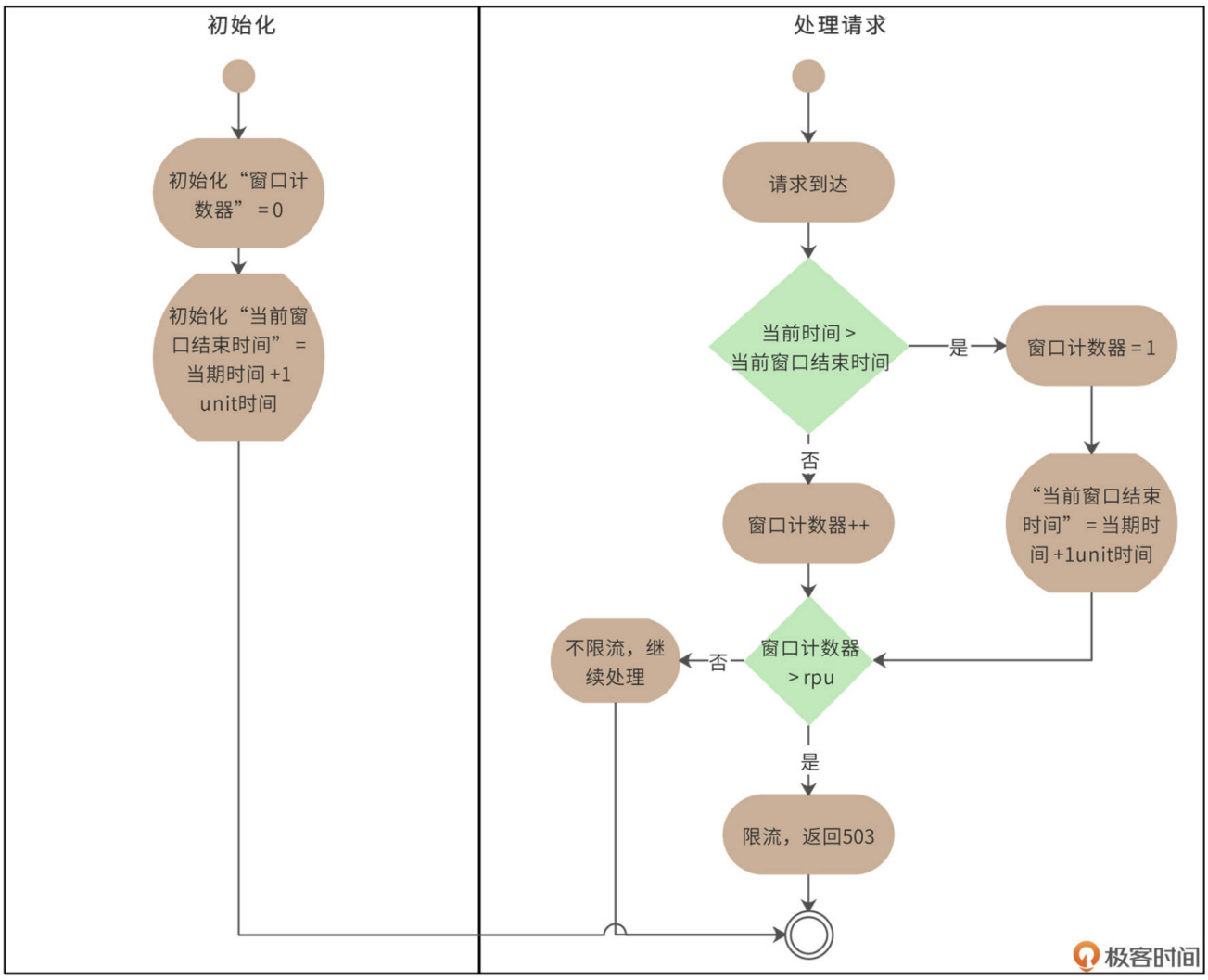
Diana 支持配置 4 种限流算法，使用者可以根据自己的需求场景，为不同资源地址配置不同的限流算法，下面详细描述这四种算法实现。

固定窗口（Window）限流算法

固定窗口限流算法就是将配置文件中的时间单位 unit 作为一个时间窗口，每个窗口仅允许限制流量内的请求通过，如图。



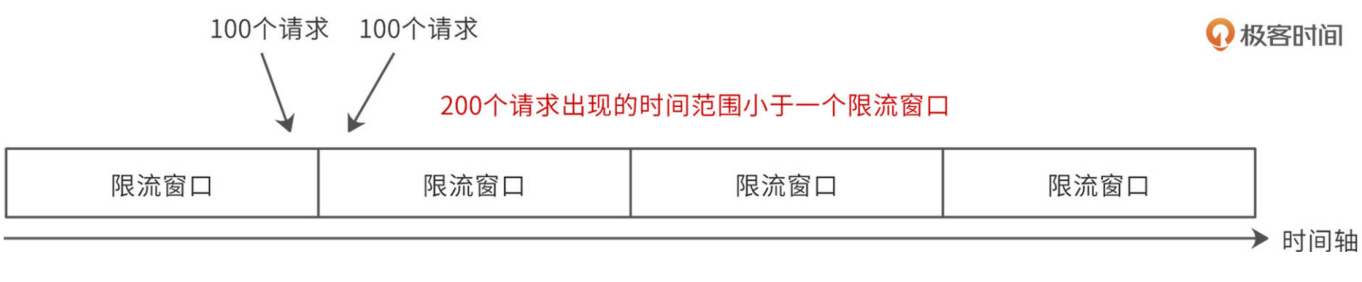
我们将时间轴切分成一个一个的限流窗口，每个限流窗口有一个窗口开始时间和一个窗口结束时间，窗口开始时，计数器清零，每进入一个请求，计数器就记录 +1。如果请求数目超过 rpu 配置的限流请求数，就拒绝服务，返回 503 响应。当前限流窗口结束后，就进入下个限流窗口，计数器再次清零，重新开始。处理流程活动图如下。



上图包括“初始化”和“处理流程”两个泳道。初始化的时候，设置“窗口计数器”和“当前窗口结束时间”两个变量。处理请求的时候，判断当前时间是否大于“当前窗口结束时间”，如果大于，那么重置“窗口计数器”和“当前窗口结束时间”两个变量；如果没有，窗口计数器 +1，并判断计数器是否大于配置的限流请求数 rpu，根据结果决定是否进行限流。

这里的“窗口计数器”可以本地记录，也可以远程记录，也就是配置中的 local 和 global。固定窗口算法在配置文件中 algo 项可配置“window”或者缩写“W”。

固定窗口实现比较容易，但是如果使用这种限流算法，在一个限流时间单位内，通过的请求数可能是 rpu 的两倍，无法达到限流的目的，如下图。



假设单位时间请求限流数 rpu 为 100，在第一个限流窗口快要到结束时间的时候，突然进来 100 个请求，因为这个请求量在限流范围内，所以没有触发限流，请求全部通过。然后进入第二个限流窗口，限流计数器清零。这时又忽然进入 100 个请求，因为已经进入第二个限流窗口，所以也没触发限流。在短时间内，通过了 200 个请求，这样可能会给系统造成巨大的负载压力。

滑动窗口（Sliding Window）限流算法

改进固定窗口缺陷的方法是采用滑动窗口限流算法，如下图。

滑动窗口就是将限流窗口内部切分成一些更小的时间片，然后在时间轴上滑动，每次滑动，滑过一个小时间片，就形成一个新的限流窗口，即滑动窗口。然后在这个滑动窗口内执行固定窗口算法即可。

滑动窗口可以避免固定窗口出现的放过两倍请求的问题，因为一个短时间内出现的所有请求必然在一个滑动窗口内，所以一定会被滑动窗口限流。

滑动窗口的算法实现基本和固定窗口一致，只要改动重置“窗口计数器”和“当前窗口结束时间”的逻辑就可以。固定窗口算法重置为窗口结束时间 +1 unit 时间，滑动窗口算法重置为窗口结束时间 +1 个时间片。但是固定窗口算法重置后，窗口计数器为 0，而滑动窗口需要将窗口计数器设置为当前窗口已经经过的时间片的请求总数，比如上图里，一个滑动窗口被分为 5 个时间片，滑动窗口 2 的浅蓝色部分就是已经经过了 4 个时间片。

滑动窗口算法在配置文件中 algo 项可配置“sliding window”或者缩写“SW”。

漏桶（Leaky Bucket）限流算法

漏桶限流算法是模拟水流过一个有漏洞的桶进而限流的思路，如图。

水龙头的水先流入漏桶，再通过漏桶底部的孔流出。如果流入的水量太大，底部的孔来不及流出，就会导致水桶太满溢出去。

限流器利用漏桶的这个原理设计漏桶限流算法，用户请求先流入到一个特定大小的漏桶中，系统以特定的速率从漏桶中获取请求并处理。如果用户请求超过限流，就会导致漏桶被请求数据填满，请求溢出，返回 503 响应。

所以漏桶算法不仅可以限流，当流量超过限制的时候会拒绝处理，直接返回 503 响应，还能控制请求的处理速度。

实践中，可以采用队列当做漏桶。如图。

构建一个特定长度的队列 queue 作为漏桶，开始的时候，队列为空，用户请求到达后从队列尾部写入队列，而应用程序从队列头部以特定速率读取请求。当读取速度低于写入速度的时候，一段时间后，队列会被写满，这时候写入队列操作失败。写入失败的请求直接构造 503 响应返回。

但是使用队列这种方式，实际上是把请求处理异步化了（写入请求的线程和获取请求的线程不是同一个线程），并不适合我们目前同步网关的场景（如果使用前面设计过的 Flower 框架开发的异步网关就可以用这种队列方式）。

因此 Diana 实现漏桶限流算法并不使用消息队列，而是阻塞等待。根据限流配置文件计算每个请求之间的间隔时间，例如：限流每秒 10 个请求，那么每两个请求的间隔时间就必须 >=100ms。用户请求到达限流器后，根据当前最近一个请求处理的时间和阻塞的请求线程数目，计算当前请求线程的 sleep 时间。每个请求线程的 sleep 时间不同，最后就可以实现每隔 100ms 唤醒一个请求线程去处理，从而达到漏桶限流的效果。

计算请求线程 sleep 时间的伪代码如下：

```
初始化 :
间隔时间 = 100ms;
阻塞线程数 = 0;
最近请求处理时间戳  = 0;

long sleep时间(){
  //最近没有请求，不阻塞
  if((now - 最近请求处理时间戳） >= 间隔时间 and 阻塞线程数 <= 0）{
    最近请求处理时间戳 = now;
    return 0; //不阻塞
  }
  //排队请求太多，漏桶溢出
  if(阻塞线程数 > 最大溢出线程数) {
    return MAX_TIME;//MAX_TIME表示阻塞时间无穷大，当前请求被限流
  }
  //请求在排队，阻塞等待
    阻塞线程数++;
    return 间隔时间 * 阻塞线程数 - (now - 最近请求处理时间戳) ;
}
```

请求线程 sleep 时间结束，继续执行的时候，修改阻塞线程数：

```
最近请求处理时间戳 = now;
阻塞线程数--;
```

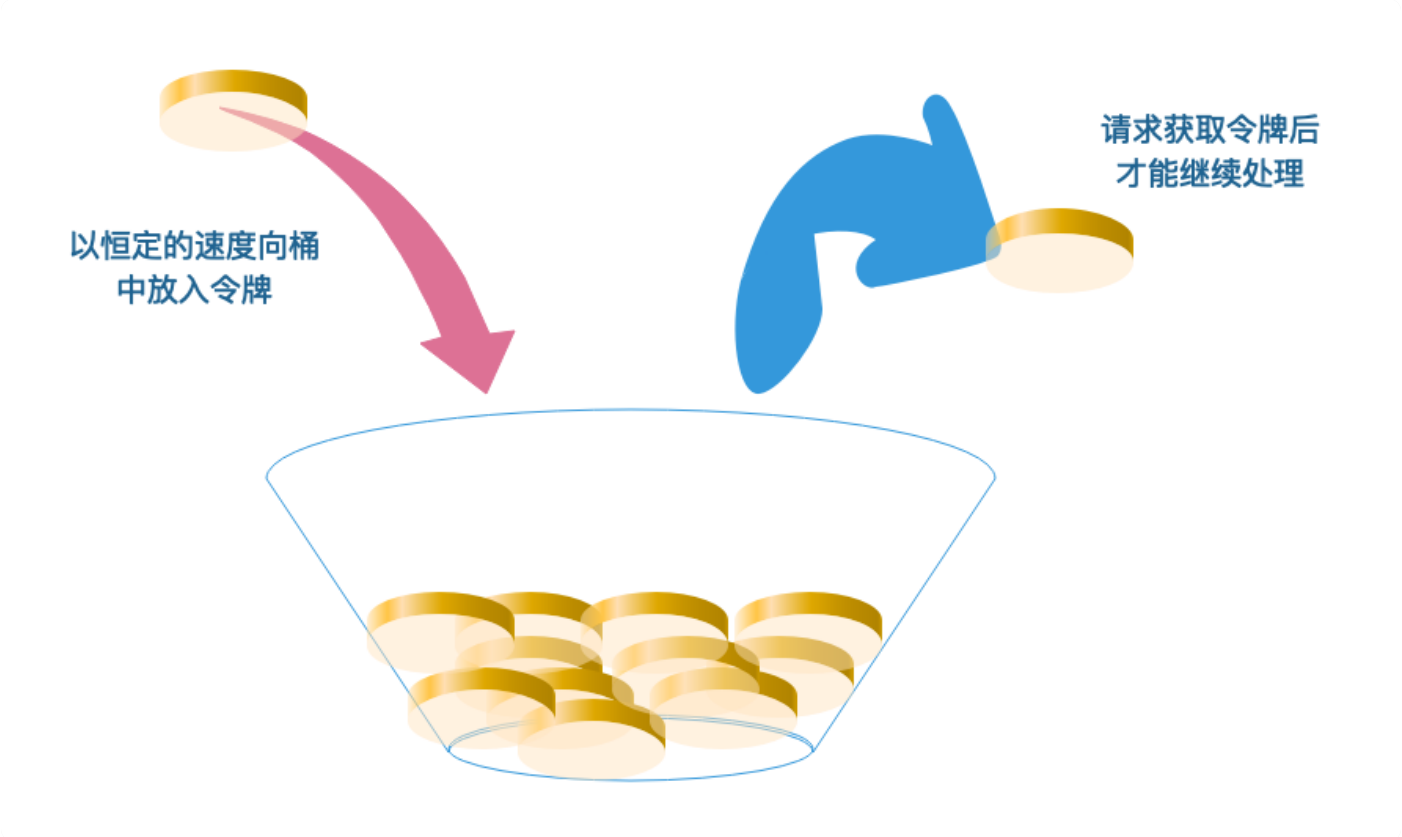
注意，以上代码多线程并发执行，需要进行加锁操作。

使用漏桶限流算法，即使系统资源很空闲，多个请求同时到达时，漏桶也是慢慢地一个接一个地去处理请求，这其实并不符合人们的期望，因为这样就是在浪费计算资源。因此除非有特别的场景需求，否则不推荐使用该算法。

漏桶算法的 algo 配置项名称为“leaky bucket”或者“LB”。

令牌桶（Token Bucket）限流算法

令牌桶是另一种桶限流算法，模拟一个特定大小的桶，然后向桶中以特定的速度放入令牌（token），请求到达后，必须从桶中取出一个令牌才能继续处理。如果桶中已经没有令牌了，那么当前请求就被限流，返回 503 响应。如果桶中的令牌放满了，令牌桶也会溢出。



上面的算法描述似乎需要有一个专门线程生成令牌，还需要一个数据结构模拟桶。实际上，令牌桶的实现，只需要在请求获取令牌的时候，通过时间计算，就可以算出令牌桶中的总令牌数。伪代码如下：

复制代码

```
初始化 :
最近生成令牌时间戳 = 0;
总令牌数 = 0;
令牌生成时间间隔 = 100ms;

boolean 获取令牌(){
  //令牌桶中有令牌，直接取令牌即可
  if(总令牌数 >= 1){
    总令牌数--;
    return true;
  }
```

```
    //令牌桶中没有令牌了，重算现在令牌桶中的总令牌数，可能算出的总令牌数依然为0
    总令牌数 = min(令牌数上限值，总令牌数 +
    (now - 最近生成令牌时间戳) / 令牌生成时间间隔);
    if(总令牌数 >= 1){
      总令牌数--;
      最近生成令牌时间戳 = now; //有令牌了，才能重设时间
      return true;
    }
    return false;
  }
```

令牌桶限流算法综合效果比较好，能在最大程度利用系统资源处理请求的基础上，实现限流的目标，建议通常场景中优先使用该算法，Diana 的缺省配置算法也是令牌桶。令牌桶算法的 algo 配置项名称为“token bucket”或“TB”。

小结

限流器是一个典型的技术中间件，使用者是应用系统开发工程师，他们在自己的应用系统中使用限流器，通过配置文件来实现满足自己业务场景的限流需求。这里隐含了一个问题：大家都是开发者，这些应用系统开发工程师为什么要用你开发的中间件？事实上，技术中间件天然会受到更多的挑剔，架构师在设计技术组件的时候要格外考虑**易用性和扩展性**，开发出来的技术中间件要能经得起同行的审视和挑战。

这篇设计文档中，包含了很多伪代码，这些伪代码是限流算法实现的核心逻辑。架构师一方面需要思考宏观的技术决策，一方面要思考微观的核心代码。这里两方面的能力支撑起架构师的技术影响力，既要能上得厅堂，在老板、客户等外部相关方面前侃侃而谈，保障自己和团队能掌控自己的技术方向；也要能下得厨房，搞定最有难度的代码实现，让团队成员相信跟着你混，没有迈不过去的技术坎。

思考题

滑动窗口算法中，如何管理时间片，以及如何计算滑动过程中的一个窗口内各个时间片的窗口计数器之和？用什么样的数据结构和算法比较合适？

欢迎在评论区分享你的思考，我们共同进步。

Ta单独购买本课程，你将得 20 元

生成海报并分享

赞 0　提建议

上一篇 14 | 百科应用系统设计：机房被火烧了系统还能访问吗？

下一篇 16 | 高可用架构的十种武器：怎么度量系统的可用性？

精选留言 (12)

写留言

Geek_7347cf
2022-03-24

个人觉得可以使用redis的的hash 通过 对时间进行取模分片

作者回复: 很赞

1

周文重
2022-03-23

Sentinel 是用 Atomic 的 Array 存储，每个数组元素为一个 bucket，存储的时候用时间戳和数组的 length 取模。算累计时间就只需要遍历数组，算每一个 bucket 里面的时间戳和当前时间戳是否在一个时间限制里面。

作者回复: 很赞。相当于用时间戳计算Hash值，key就是时间片，所以其实不用遍历数组，取模就得到数组下标。

1

2022-03-23

我可能会使用定长队列存储每一个时间窗。时间窗只统计最近的若干个，符合先进先出的要求。额外用一个变量存储总计数器。

新的请求进来，如果没有切换新的时间窗，将最后的时间窗计数器+1。如果涉及时间窗切换，则需要先做出队操作。并且总计数器扣除出队的时间窗计数器。然后入队新的时间窗，时间窗计数器为1

如果，中间请求间隔过长，涉及跳时间窗，则根据实际情况，入队若干个0时间窗进行填充。

共 1 条评论>

1

食指可爱多
2022-04-06

我之前做过一个分布式限流组建，就是用令牌桶算法。redis做存储，脚本逻辑用lua脚本实现，算法逻辑参考Google Guava中的实现。

作者回复: 赞

dao
2022-04-02

固定窗口是滑动窗口的一个特例。滑动窗口算法：
1）在同一时间片内累加计数；
2）在同一时间窗口内，切换时间片时计数会移交下一时间片；
3）窗口切换时，重置时间片序号和计算器。
固定窗口没有2），并且1）是指同一时间窗口，3）没有时间片序号重置。
个人看法，对于同一限流计数器，不需要特殊的数据结构，使用一般对象就可以。对象可能包含的信息如下，

{ 时间片序号，截止时间戳，计数 }（附：这个对象是复用的，在计数、时间片切换、窗口切换时更新）
多种限流计数器之间使用 hashmap 。

作者回复: 赞

Lance-Yanh
2022-03-28

本地用concurrentSkipListMap，远程用redis zset，辛苦老师点评

作者回复: 嗯嗯，具体如何实现呢，key value如何设计？时间片内请求如何计数？

javaadu
2022-03-26

（1）使用队列管理时间片，每次窗口移动就是一个出队和入队操作

（2）使用hashmap管理各个时间片的计数器key，如果是本地模式则value存在内存中，如果是分布式模式则value存放在redis中

作者回复: 非常赞

Geek_7347cf
2022-03-23

对于固定窗口或滑动窗口 如果所有请求都集中在第一个1/4区段触发阈值，那么后面3/4区段将被闲置，无法处理用户请求，系统不可用时间>系统可用时间，针对这种情况有没有比较好的解决方案

作者回复: 1 这种情况可以用更小的窗口和更小的时间片改善
2 后3/4并没有闲置，前面集中进来的请求还在服务器中处理呢，这就是限流器的目的。通常也不存在系统不可用时间>系统可用时间，如果是集中在前1/4进来的，就意味着后3/4请求很少；如果前后的请求都很多，意味着正常请求就有这么多，那就是阈值设置不合理或者服务器部署太少了。

Geek_7347cf

2022-03-23

重置窗口时间的逻辑 在高并发场景下 是不是应该考虑锁的问题

作者回复: 是的


奔浪

2022-03-23

首先:

能想到的是链表带头尾指针呢种，新来的从头部插入，而检查是否过期直接从尾部检查【前提是进入的元素要带有时间否则没法判断】

第二种:

基于上面进行变形，上面检查是否过期是有个一线程定时轮询，或者在准备进入新元素的时候被动检查是否能弹出过期元素这样的我们也可以将此工作封入链表结构本身，在插入元素自动检查尾部元素是否过期如果过期自动弹出。

当然以上当检查到尾部有过期的那么接着迭代检查下一个元素


丫丫

2022-03-23

谢谢老师的专题！

请问老师，在分布式限流中，本地限流器如何知道其他限流器的计数，工程中用什么同步算法或者协议？谢谢老师。

作者回复: Redis进行全局计数

共 3 条评论>

peter

2022-03-23

请教老师几个问题啊：

Q1：限流器可以和网关相互独立吗？

文中提到的限流器是网关的一个组件，限流器不是独立的。SpringCloud中的sentinel和网关是独立部署的。限流器应该可以和网关相互独立吧。

Q2：Nginx可以作为限流器使用吗？

Q3：固定窗口方法中的时间结束是通过定时器实现的吗？

作者回复: 1 快速看了下sentinel文档，sentinel的控制台是独立部署，sentinel流控组件也需要和网关或者应用部署在一起。

2 可以

3 不是，请求进入的时候判断，文中有讲