# 分布式限流

## 固定窗口限流

将时间划分为多个窗口；

在每个窗口内每有一次请求就将计数器加一；

如果计数器超过了限制数量，则本窗口内所有的请求都被丢弃当时间到达下一个窗口时，计数器重置。

固定窗口计数器是最为简单的算法，但这个算法有时会让通过请求量允许为限制的两倍。考虑如下情况：限制 1 秒内最多通过 5 个请求，在第一个窗口的最后半秒内通过了 5 个请求，第二个窗口的前半秒内又通过了 5 个请求。这样看来就是在 1 秒内通过了 10 个请求。

## 滑动窗口限流

将时间划分为多个区间；

在每个区间内每有一次请求就将计数器加一维持一个时间窗口，占据多个区间；

每经过一个区间的时间，则抛弃最老的一个区间，并纳入最新的一个区间

如果当前窗口内区间的请求计数总和超过了限制数量，则本窗口内所有的请求都被丢弃。

滑动窗口计数器是通过将窗口再细分，并且按照时间"滑动"，这种算法避免了固定窗口计数器带来的双倍突发请求，但时间区间的精度越高，算法所需的空间容量就越大。

## 桶漏

将每个请求视作"水滴"放入"漏桶"进行存储；

“漏桶"以固定速率向外"漏"出请求来执行如果"漏桶"空了则停止"漏水”；

如果"漏桶"满了则多余的"水滴"会被直接丢弃。

漏桶算法多使用队列实现，服务的请求会存到队列中，服务的提供方则按照固定的速率从队列中取出请求并执行，过多的请求则放在队列中排队或直接拒绝。

漏桶算法的缺陷也很明显，当短时间内有大量的突发请求时，即便此时服务器没有任何负载，每个请求也都得在队列中等待一段时间才能被响应。

这个东西根本就不能在请求的场景用啊, 你这个就是把消息放到队列里面,然后按照固定速率流出.

## 令牌桶

https://github.com/go-redis/redis\_rate