【Java 专题系列】「回顾 RateLimiter 限流器的入门到精通（含实战和算法原理介

作者：洛神灬殇

2021-09-17
本文字数：5867 字 阅读完需约 19 分钟

并发编程的三剑客

> 在开发高并发系统时有三剑客：缓存、降级和限流。

缓存 缓存的目的是提升系统访问速度和增大系统处理容量。
降级 降级是当服务出现问题或者影响到核心流程时，需要暂时屏蔽掉，待高峰或者问题解决后再打开。
限流 限流的目的是通过对并发访问/请求进行限速，或者对一个时间窗口内的请求进行限速来保护系统，一旦达到限制速率则可以拒绝服务、排队或等待、降级等处理。

限流的思想

溢出思想：

> 就是用一个固定大小的队列。比如设置限流为 5qps，1s 可以接受 5 个请求；那我们就造一个大小为 5 的队列，如果队列为满了，就拒绝请求；如果队列未满，就往队列添加请求。

速度控制

> **令牌听起来挺酷的。以固定的速率往桶里发放令牌。然后消费者每次要取到令牌(acquire)才可以响应请求**，控制速率呢，我们通过控制消费者的消费速率是 5qps，1s 消费 5 个即可。

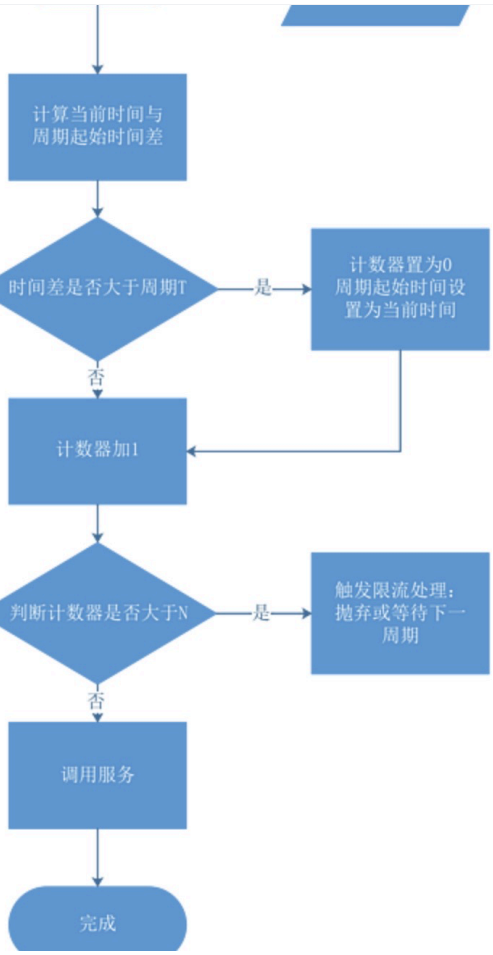
限流的算法

> 常用的限流算法有两种：**漏桶算法和令牌桶算法**及滑动窗口(计数器)算法等。

计数限流算法

> **无论固定窗口还是滑动窗口核心均是对请求进行计数，区别仅仅在于对于计数时间区间的处理。**

固定窗口计数



创作场景
记录自己日常工作的实践
发表对生活和职场的感悟
针对感兴趣的事件发表随
从0到1详细介绍你掌握的
技术，或者一个兴趣、爱
或者，就直接把你的个人
接搬到这里

实现原理

固定窗口计数法思想比较简单，只需要确定两个参数：计数周期 T 及周期内最大访问（调用）数 N。请求到达时使用以下流程进行操作：

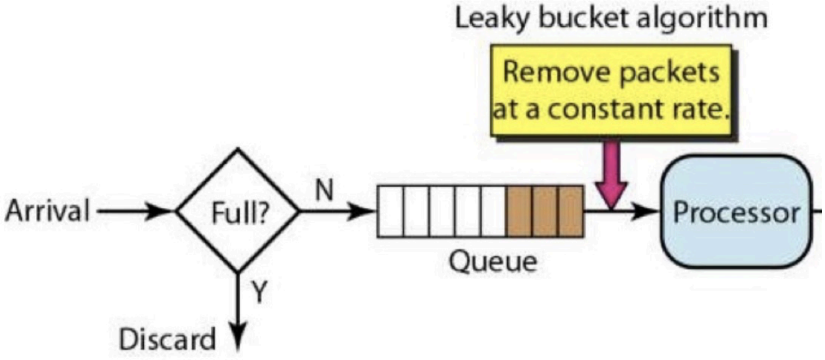
固定窗口计数实现简单，并且只需要记录上一个周期起始时间与周期内访问总数，几乎不消耗额外的存储空间。

算法缺陷

> **固定窗口计数缺点也非常明显，在进行周期切换时，上一个周期的访问总数会立即置为 0，这可能导致在进行周期切换时可能出现流量突发**

令牌桶算法

> 令牌桶算法的原理：系统会以一个恒定的速度往桶里放入令牌，而如果请求需要被处理，则需要先从桶里获取一个令牌，当桶里没有令牌可取时，则拒绝服务。



桶中存放的令牌数有最大上限，超出之后就被丢弃或者拒绝。

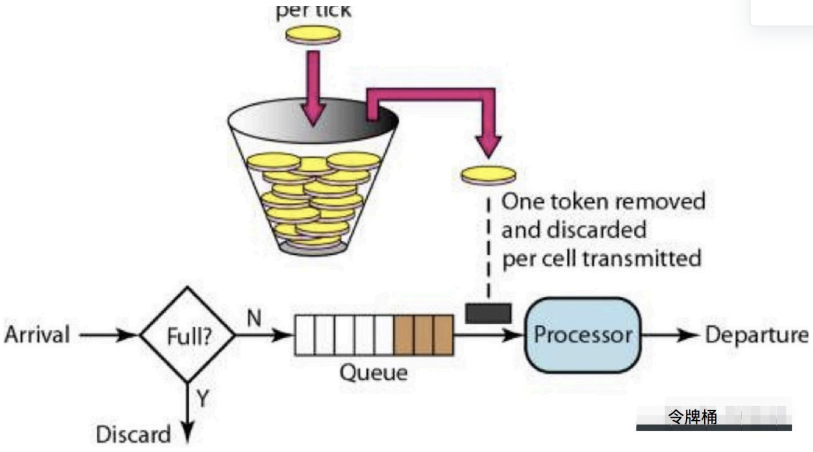
当流量或者网络请求到达时，每个请求都要获取一个令牌，如果能够获取到，则直接处理，并
牌。

如果获取不到，该请求就要被限流，要么直接丢弃，要么在缓冲区等待。

创作场景
记录自己日常工作的实践
发表对生活和职场的感悟
针对感兴趣的事件发表随
从0到1详细介绍你掌握的
技术，或者一个兴趣、爱
或者，就直接把你的个人
接搬到这里

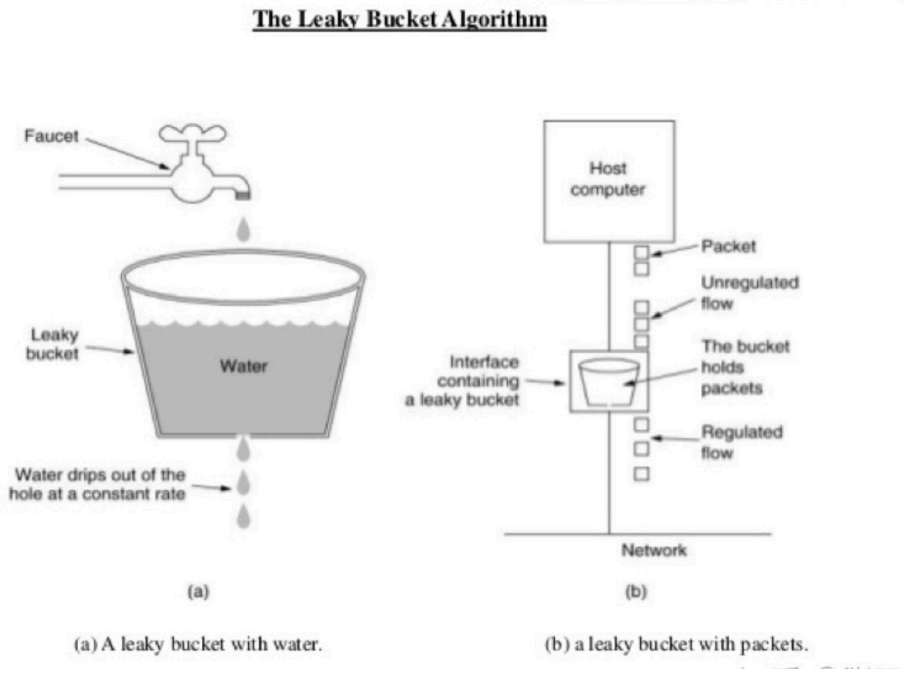
优点

> 由于令牌是固定间隔发放的，假设还是 5qps，如果我有 1s 内没有请求，我的令牌桶就满了
> 个请求（一次过取 5 个令牌），也就是可以应对瞬时流量。



漏桶算法

漏桶算法思路很简单，水（请求）先进入到漏桶里，漏桶以一定的速度出水，当水流入速度过大会直接溢出，可以看出漏桶算法能强行限制数据的传输速率。



如上图就像一个漏斗一样，进来的水量就好像访问流量一样，而出去的水量就像是我们的系统处理请求一样。

当访问流量过大时，这个漏斗中就会积水，如果水太多了就会溢出。

> 漏桶算法的实现往往依赖于队列，请求到达如果队列未满则直接放入队列，然后有一个处理器按照固定频率从队列头取出请求进行处理。如果请求量大，则会导致队列满，那么新来的请求就会被抛弃。

令牌桶是按照固定速率往桶中添加令牌，请求是否被处理需要看桶中令牌是否足够，当令牌数
求；漏桶则是按照常量固定速率流出请求，流入请求速率任意，当流入的请求数累积到漏桶容
被拒绝；
令牌桶限制的是平均流入速率，允许突发请求，只要有令牌就可以处理，支持一次拿 3 个令牌
漏桶限制的是常量流出速率，即流出速率是一个固定常量值，比如都是 1 的速率流出，而不能
2，从而平滑突发流入速率；
令牌桶允许一定程度的突发，而漏桶主要目的是平滑流出速率；

> **除了要求能够限制数据的平均传输速率外，还要求允许某种程度的突发传输。这时候漏桶算法**
> **牌桶算法更为适合。**

信号量的应用

操作系统的信号量是个很重要的概念，Java 并发库 的 Semaphore 可以很轻松完成信号量控制，Semaphore 可以控制某个资源可被同时访问的个数，通过 acquire() 获取一个许可，如果没有就等待，而 release() 释放一个许可。
信号量的本质是控制某个资源可被同时访问的个数，在一定程度上可以控制某资源的访问频率，但不能精确控制。

限流的思想

Guava 中的 RateLimiter 可以限制单进程中某个方法的速率，本文主要介绍如何使用，实现原理请参考文档：推荐：超详细的 Guava RateLimiter 限流原理解析和推荐：RateLimiter 源码分析(Guava 和 Sentinel 实现)。

Guava RateLimiter

> **Google 开源工具包 Guava 提供了限流工具类 RateLimiter，该类基于令牌桶算法实现流量限制，使用十分方便。**

原理：Guava RateLimiter 基于令牌桶算法，

RateLimiter 系统限制 QPS 是多少，那么 RateLimiter 将以这个速度往桶里面放入令牌。
然后请求的时候，通过 tryAcquire()方法向 RateLimiter 获取许可（令牌）。

Guava RateLimiter 控制操作

Guava RateLimiter 限速手段

RateLimiter 从概念上来讲，速率限制器会在可配置的速率下分配许可证。如果必要的话，每个 acquire() 会阻塞当前线程直到许可证可用后获取该许可证。一旦获取到许可证，不需要再释放许可证。
RateLimiter 通过限制后面请求的等待时间，来支持一定程度的突发请求(预消费)。

Maven 配置

复制代码

```
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>31.0-jre</version>
</dependency>

```

Java 简单案例

复制代码

```
public class RateLimiterService {
```

创作场景
记录自己日常工作的实践
发表对生活和职场的感悟
针对感兴趣的事件发表随
从0到1详细介绍你掌握的技术，或者一个兴趣、爱
或者，就直接把你的个人接搬到这里

```
     */
    public boolean tryAcquire() {
        return rateLimiter.tryAcquire();
    }
  public  void acquire() {
        rateLimiter.acquire();
    }
   public static void main(String[] args){
        if (accessLimitService.tryAcquire()) {
            log.info("start");
            // 模拟业务执行500毫秒
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            return "access success [" + LocalDateTime.now() + "]";
        } else {
            //log.warn("限流");
            return "access limit [" + LocalDateTime.now() + "]";
        }
      }
}

public void testMethod(){
  ExecutorService pool = Executors.newFixedThreadPool(10);
        RateLimiter rateLimiter = RateLimiter.create(5); // rate is "5 permits per second"
        IntStream.range(0, 10).forEach(i -> pool.submit(() -> {
            if (rateLimiter.tryAcquire()) {
                try {
                    log.info("start");
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                }
            } else {
                log.warn("限流");
            }
        }));

public void testMethod2(){
  ExecutorService pool = Executors.newFixedThreadPool(10);
        RateLimiter rateLimiter = RateLimiter.create(5); // rate is "5 permits per second"
        IntStream.range(0, 10).forEach(i -> pool.submit(() -> {
            rateLimiter.acquire();
            log.info("start");
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }));
        pool.shutdown();
      }
  }
}
```

复制代码

```
public class GuavaRateLimiter {
    public static ConcurrentHashMap<String, RateLimiter> resourceRateLimiter = new ConcurrentHashMa
    //初始化限流工具RateLimiter
    static {
        createResourceRateLimiter("order", 50);
    }
    public static void createResourceRateLimiter(String resource, double qps) {
        if (resourceRateLimiter.contains(resource)) {
            resourceRateLimiter.get(resource).setRate(qps);
        } else {
            //创建限流工具，每秒发出50个令牌指令
            RateLimiter rateLimiter = RateLimiter.create(qps);
            resourceRateLimiter.putIfAbsent(resource, rateLimiter);
```

创作场景
记录自己日常工作的实践
发表对生活和职场的感悟
针对感兴趣的事件发表随
从0到1详细介绍你掌握的技术，或者一个兴趣、爱
或者，就直接把你的个人接搬到这里

```
18              new Thread(new Runnable() {
19                  @Override
20                  public void run() {
21                      //如果获得令牌指令，则执行业务逻辑
22                      if (resourceRateLimiter.get("order").tryAcquire(10, TimeUnit.
23                          System.out.println("执行业务逻辑");
24                      } else {
25                          System.out.println("限流");
26                      }
27                  }
28              }).start();
29          }
30      }
31  }
32
```

创作场景
记录自己日常工作的实践
发表对生活和职场的感悟
针对感兴趣的事件发表随
从0到1详细介绍你掌握的技术，或者一个兴趣、爱
或者，就直接把你的个人接搬到这里

方法摘要

限流及创建方法

修饰符和类型	方法和描述
static RateLimiter	create(double permitsPerSecond) 根据指定的稳定吞吐率创建RateLimiter，这里的吞吐率是指每秒多少许可数（通常是指QPS，每秒多少查询）
static RateLimiter	create(double permitsPerSecond, long warmupPeriod, TimeUnit unit) 根据指定的稳定吞吐率和预热期来创建RateLimiter，这里的吞吐率是指每秒多少许可数（通常是指QPS，每秒多少个请求量），在这段预热时间内，RateLimiter每秒分配的许可数会平稳地增长直到预热期结束时达到其最大速率。（只要存在足够请求数来使其饱和）

create 方法

复制代码

```
1 public static RateLimiter create(double permitsPerSecond)
2
```

> **根据指定的稳定吞吐率创建 RateLimiter，这里的吞吐率是指每秒多少许可数（通常是指 QPS，每秒多少查询）。**

> The returned RateLimiter ensures that on average no more than permitsPerSecond are issued during any given second, with sustained requests being smoothly spread over each second. When the incoming request rate exceeds permitsPerSecond the rate limiter will release one permit every (1.0 / permitsPerSecond) seconds. When the rate limiter is unused, bursts of up to permitsPerSecond permits will be allowed, with subsequent requests being smoothly limited at the stable rate of permitsPerSecond.

返回的 RateLimiter

确保了在平均情况下，每秒发布的许可数不会超过 permitsPerSecond，每秒钟会持续发送请求。
当传入请求速率超过 permitsPerSecond，速率限制器会每秒释放一个许可(1.0 / permitsPerSecond 这里是指设定了 permitsPerSecond 为 1.0)。
当速率限制器闲置时，允许许可数暴增到 permitsPerSecond，随后的请求会被平滑地限制在稳定速率 permitsPerSecond 中。

permitsPerSecond – 返回的 RateLimiter 的速率，意味着每秒有多少个许可变成有效。
抛出:
IllegalArgumentException – 如果 permitsPerSecond 为负数或者为 0

```
1 public static RateLimiter create(double permitsPerSecond,long warmupPeriod,TimeUn
2
```

> 根据指定的稳定吞吐率和预热期来创建 RateLimiter，这里的吞吐率是指每秒多少许可数（通
> 多少查询），在这段预热时间内，RateLimiter 每秒分配的许可数会平稳地增长直到预热期结束时达到其最大速率
> （只要存在足够请求数来使其饱和）。同样地，如果 RateLimiter 在 warmupPeriod 时间内闲置不用，它将会
> 逐步地返回冷却状态。也就是说，它会像它第一次被创建般经历同样的预热期。返回的 RateLimiter 主要用于那
> 些需要预热期的资源，这些资源实际上满足了请求（比如一个远程服务），而不是在稳定（最大）的速率下可以立
> 即被访问的资源。返回的 RateLimiter 在冷却状态下启动（即预热期将会紧跟着发生），并且如果被长期闲置不
> 用，它将回到冷却状态。

参数:

permitsPerSecond – 返回的 RateLimiter 的速率，意味着每秒有多少个许可变成有效。
warmupPeriod – 在这段时间内 RateLimiter 会增加它的速率，在抵达它的稳定速率或者最大速率之前
unit – 参数 warmupPeriod 的时间单位

抛出:

IllegalArgumentException – 如果 permitsPerSecond 为负数或者为 0

限流及阻塞方法

acquire

修饰符和类型	方法和描述
boolean	tryAcquire() 从RateLimiter 获取许可，如果该许可可以在无延迟下的情况下立即获取得到的话
boolean	tryAcquire(int permits) 从RateLimiter 获取许可数，如果该许可数可以在无延迟下的情况下立即获取得到的话
boolean	tryAcquire(int permits, long timeout, TimeUnit unit) 从RateLimiter 获取指定许可数如果该许可数可以在不超过timeout的时间内获取得到的话，或者如果无法在timeout 过期之前获取得到许可数的话，那么立即返回false （无需等待）
boolean	tryAcquire(long timeout, TimeUnit unit) 从RateLimiter 获取许可如果该许可可以在不超过timeout的时间内获取得到的话，或者如果无法在timeout 过期之前获取得到许可的话，那么立即返回false（无需等待）
double	acquire() 从RateLimiter获取一个许可，该方法会被阻塞直到获取到请求
double	acquire(int permits) 从RateLimiter获取指定许可数，该方法会被阻塞直到获取到请求

复制代码

```
1 public double acquire()
```

创作场景
记录自己日常工作的实践
发表对生活和职场的感悟
针对感兴趣的事件发表随
从0到1详细介绍你掌握的技术，或者一个兴趣、爱
或者，就直接把你的个人接搬到这里

> 从 RateLimiter 获取一个许可，该方法会被阻塞直到获取到请求。如果存在等待的情况的话，该请求所需要的睡眠时间。该方法等同于 acquire(1)。

返回:

> time spent sleeping to enforce rate, in seconds; 0.0 if not rate-limited 执行速率的单位为妙；如果没有则返回 0

acquire

复制代码

```
1 public double acquire(int permits)
2
```

> 从 RateLimiter 获取指定许可数，该方法会被阻塞直到获取到请求数。如果存在等待的情况的话，告诉调用者获取到这些请求数所需要的睡眠时间。

参数:

permits – 需要获取的许可数

返回:

执行速率的所需要的睡眠时间，单位为妙；如果没有则返回 0

抛出:

IllegalArgumentException – 如果请求的许可数为负数或者为 0

tryAcquire

复制代码

```
1 public boolean tryAcquire(long timeout,TimeUnit unit)
2
```

> 从 RateLimiter 获取许可如果该许可可以在不超过 timeout 的时间内获取得到的话，或者如果无法在 timeout 过期之前获取得到许可的话，那么立即返回 false（无需等待）。该方法等同于 tryAcquire(1, timeout, unit)。

参数:

timeout – 等待许可的最大时间，负数以 0 处理
unit – 参数 timeout 的时间单位

返回:

true 表示获取到许可，反之则是 false

抛出:

IllegalArgumentException – 如果请求的许可数为负数或者为 0

tryAcquire

创作场景
记录自己日常工作的实践
发表对生活和职场的感悟
针对感兴趣的事件发表随
从0到1详细介绍你掌握的技术，或者一个兴趣、爱
或者，就直接把你的个人接搬到这里

> 从 RateLimiter 获取指定许可数如果该许可数可以在不超过 timeout 的时间内获取得到的话，meout 过期之前获取得到许可数的话，那么立即返回 false （无需等待）。

参数:

permits – 需要获取的许可数
timeout – 等待许可数的最大时间，负数以 0 处理
unit – 参数 timeout 的时间单位

返回:

true 表示获取到许可，反之则是 false

限流及状态设置

修饰符和类型	方法和描述
double	getRate() 返回RateLimiter 配置中的稳定速率，该速率单位是每秒多少许可数
void	setRate(double permitsPerSecond) 更新RateLimite的稳定速率，参数permitsPerSecond 由构造RateLimiter的工厂方法提供。

复制代码

```
1 public final void setRate(double permitsPerSecond)
2
```

> 更新 RateLimite 的稳定速率，参数 permitsPerSecond 由构造 RateLimiter 的工厂方法提供。调用该方法后，当前限制线程不会被唤醒，因此他们不会注意到最新的速率；只有接下来的请求才会。需要注意的是，由于每次请求偿还了（通过等待，如果需要的话）上一次请求的开销，这意味着紧紧跟着的下一个请求不会被最新的速率影响到，在调用了 setRate 之后；它会偿还上一次请求的开销，这个开销依赖于之前的速率。RateLimiter 的行为在任何方式下都不会被改变，比如如果 RateLimiter 有 20 秒的预热期配置，在此方法被调用后它还是会进行 20 秒的预热。

参数:

permitsPerSecond – RateLimiter 的新的稳定速率

抛出:

IllegalArgumentException – 如果 permitsPerSecond 为负数或者为 0

复制代码

```
1 public final double getRate()
2
```

> 返回 RateLimiter 配置中的稳定速率，该速率单位是每秒多少许可数。它的初始值相当于构造这个 RateLimiter 的工厂方法中的参数 permitsPerSecond ，并且只有在调用 setRate(double)后才会被更新。

发布于: 2021-09-17 | 阅读数: 792

版权声明: 本文为 InfoQ 作者【洛神灬殇】的原创文章。
原文链接:【https://xie.infoq.cn/article/371ef8ee16df6a58b97e0e315】。文章转载请联系作

限流算法 Guava 9月日更 Gatelimitor

洛神灬殇
InfoQ写作平台-签约作者 2020-03-25 加入
前优酷资深工程师，具有高洞察力的理性自律小i人 — INTJ 个人著作《深入浅出Java虚拟机—JVM原理与实战》 10年开发经验，参与过多个大型互联网项目，定期分享技术干货和项目经验

点赞 | 收藏 | 微信 | 微博 | 部落 | 举报

评论

快抢沙发！虚位以待 发布

暂无评论

创作场景
记录自己日常工作的实践
发表对生活和职场的感悟
针对感兴趣的事件发表随
从0到1详细介绍你掌握的技术，或者一个兴趣、爱
或者，就直接把你的个人接搬到这里

Copyright © 2024, Geekbang Technology Ltd. All rights reserved. 极客邦控股（北京）有限公司 | 京 ICP 备 16027448 号 - 5 京公网安备 11010502039052号 | 产品资质