

38 | 案例分析（一）：高性能限流器Guava RateLimiter

2019-05-25 王宝令

Java并发编程实战

[进入课程 >](#)



讲述：王宝令

时长 09:09 大小 8.40M




从今天开始，我们就进入案例分析模块了。这个模块我们将分析四个经典的开源框架，看看它们是如何处理并发问题的，通过这四个案例的学习，相信你会对如何解决并发问题有个更深入的认识。

首先我们来看看**Guava RateLimiter** 是如何解决高并发场景下的限流问题的。Guava 是 Google 开源的 Java 类库，提供了一个工具类 RateLimiter。我们先来看看 RateLimiter 的使用，让你对限流有个感官的印象。假设我们有一个线程池，它每秒只能处理两个任务，如果提交的任务过快，可能导致系统不稳定，这个时候就需要用到限流。

在下面的示例代码中，我们创建了一个流速为 2 个请求 / 秒的限流器，这里的流速该怎么理解呢？直观地看，2 个请求 / 秒指的是每秒最多允许 2 个请求通过限流器，其实在

Guava 中，流速还有更深一层的意思：是一种匀速的概念，2 个请求 / 秒等价于 1 个请求 / 500 毫秒。

在向线程池提交任务之前，调用 `acquire()` 方法就能起到限流的作用。通过示例代码的执行结果，任务提交到线程池的时间间隔基本上稳定在 500 毫秒。

 复制代码

```
1 // 限流器流速：2 个请求 / 秒
2 RateLimiter limiter =
3     RateLimiter.create(2.0);
4 // 执行任务的线程池
5 ExecutorService es = Executors
6     .newFixedThreadPool(1);
7 // 记录上一次执行时间
8 prev = System.nanoTime();
9 // 测试执行 20 次
10 for (int i=0; i<20; i++){
11     // 限流器限流
12     limiter.acquire();
13     // 提交任务异步执行
14     es.execute(()->{
15         long cur=System.nanoTime();
16         // 打印时间间隔：毫秒
17         System.out.println(
18             (cur-prev)/1000_000);
19         prev = cur;
20     });
21 }
22
23 输出结果：
24 ...
25 500
26 499
27 499
28 500
29 499
```

经典限流算法：令牌桶算法

Guava 的限流器使用上还是很简单的，那它是如何实现的呢？Guava 采用的是**令牌桶算法**，其核心是要想通过限流器，必须拿到令牌。也就是说，只要我们能够限制发放令牌的速率，那么就能控制流速了。令牌桶算法的详细描述如下：

1. 令牌以固定的速率添加到令牌桶中，假设限流的速率是 r / 秒，则令牌每 $1/r$ 秒会添加一个；
2. 假设令牌桶的容量是 b ，如果令牌桶已满，则新的令牌会被丢弃；
3. 请求能够通过限流器的前提是令牌桶中有令牌。

这个算法中，限流的速率 r 还是比较容易理解的，但令牌桶的容量 b 该怎么理解呢？ b 其实是 burst 的简写，意义是**限流器允许的最大突发流量**。比如 $b=10$ ，而且令牌桶中的令牌已满，此时限流器允许 10 个请求同时通过限流器，当然只是突发流量而已，这 10 个请求会带走 10 个令牌，所以后续流量只能按照速率 r 通过限流器。

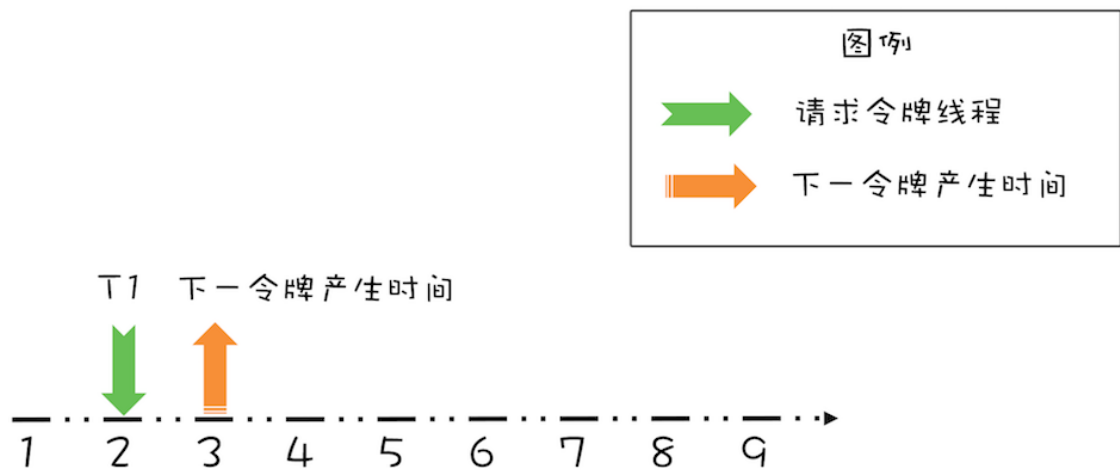
令牌桶这个算法，如何用 Java 实现呢？很可能你的直觉会告诉你生产者 - 消费者模式：一个生产者线程定时向阻塞队列中添加令牌，而试图通过限流器的线程则作为消费者线程，只有从阻塞队列中获取到令牌，才允许通过限流器。

这个算法看上去非常完美，而且实现起来非常简单，如果并发量不大，这个实现并没有什么问题。可实际情况却是使用限流的场景大部分都是高并发场景，而且系统压力已经临近极限了，此时这个实现就有问题了。问题就出在定时器上，在高并发场景下，当系统压力已经临近极限的时候，定时器的精度误差会非常大，同时定时器本身会创建调度线程，也会对系统的性能产生影响。

那还有什么好的实现方式呢？当然有，Guava 的实现就没有使用定时器，下面我们就来看看它是如何实现的。

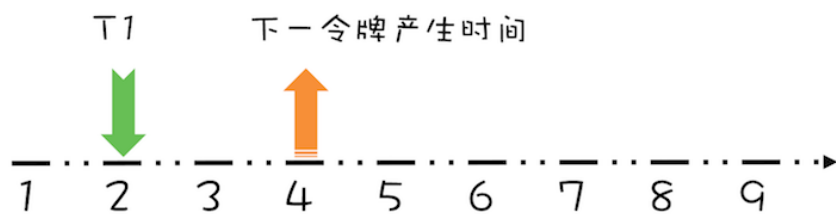
Guava 如何实现令牌桶算法

Guava 实现令牌桶算法，用了一个很简单的办法，其关键是**记录并动态计算下一令牌发放的时间**。下面我们以一个最简单的场景来介绍该算法的执行过程。假设令牌桶的容量为 $b=1$ ，限流速率 $r = 1$ 个请求 / 秒，如下图所示，如果当前令牌桶中没有令牌，下一个令牌的发放时间是在第 3 秒，而在第 2 秒的时候有一个线程 T1 请求令牌，此时该如何处理呢？



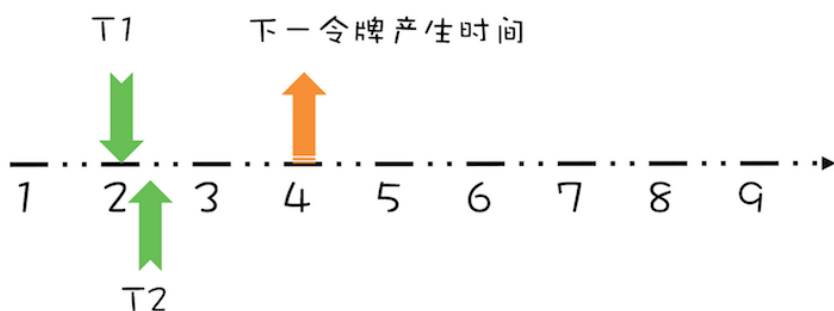
线程 T1 请求令牌示意图

对于这个请求令牌的线程而言，很显然需要等待 1 秒，因为 1 秒以后（第 3 秒）它就能拿到令牌了。此时需要注意的是，下一个令牌发放的时间也要增加 1 秒，为什么呢？因为第 3 秒发放的令牌已经被线程 T1 预占了。处理之后如下图所示。



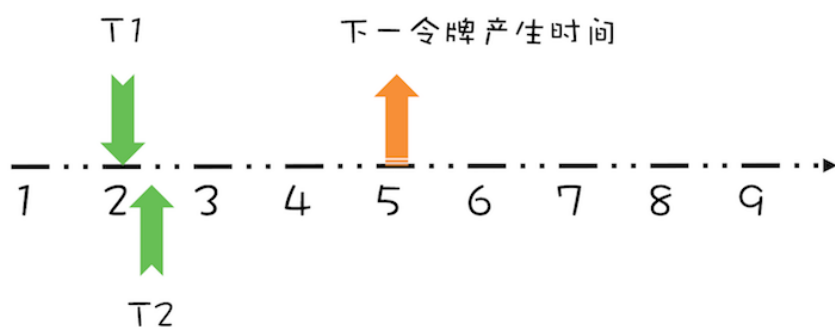
线程 T1 请求结束示意图

假设 T1 在预占了第 3 秒的令牌之后，马上又有一个线程 T2 请求令牌，如下图所示。



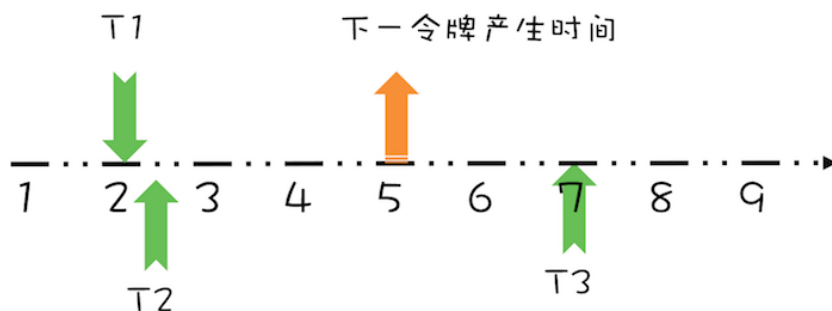
线程 T2 请求令牌示意图

很显然，由于下一个令牌产生的时间是第 4 秒，所以线程 T2 要等待两秒的时间，才能获取到令牌，同时由于 T2 预占了第 4 秒的令牌，所以下一令牌产生时间还要增加 1 秒，完全处理之后，如下图所示。



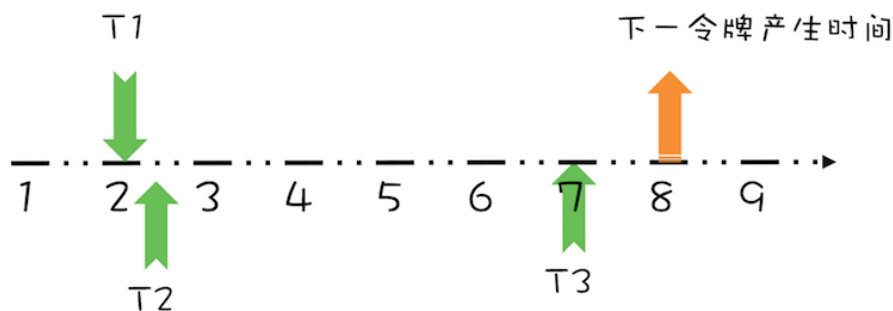
线程 T2 请求结束示意图

上面线程 T1、T2 都是在**下一令牌产生时间之前**请求令牌，如果线程在**下一令牌产生时间之后**请求令牌会如何呢？假设在线程 T1 请求令牌之后的 5 秒，也就是第 7 秒，线程 T3 请求令牌，如下图所示。




线程 T3 请求令牌示意图

由于在第 5 秒已经产生了一个令牌，所以此时线程 T3 可以直接拿到令牌，而无需等待。在第 7 秒，实际上限流器能够产生 3 个令牌，第 5、6、7 秒各产生一个令牌。由于我们假设令牌桶的容量是 1，所以第 6、7 秒产生的令牌就丢弃了，其实等价地你也可以认为是保留的第 7 秒的令牌，丢弃的第 5、6 秒的令牌，也就是说第 7 秒的令牌被线程 T3 占有了，于是下一令牌的产生时间应该是第 8 秒，如下图所示。



线程 T3 请求结束示意图

通过上面简要地分析，你会发现，我们**只需要记录一个下一令牌产生的时间，并动态更新它，就能够轻松完成限流功能**。我们可以将上面的这个算法代码化，示例代码如下所示，依然假设令牌桶的容量是 1。关键是**reserve() 方法**，这个方法会为请求令牌的线程预分配令牌，同时返回该线程能够获取令牌的时间。其实现逻辑就是上面提到的：如果线程请求令牌的时间在下一令牌产生时间之后，那么该线程立刻就能够获取令牌；反之，如果请求时间在下一令牌产生时间之前，那么该线程是在下一令牌产生的时间获取令牌。由于此时下一令牌已经被该线程预占，所以下一令牌产生的时间需要加上 1 秒。

 复制代码


```
1 class SimpleLimiter {
2     // 下一令牌产生时间
3     long next = System.nanoTime();
4     // 发放令牌间隔：纳秒
5     long interval = 1000_000_000;
6     // 预占令牌，返回能够获取令牌的时间
7     synchronized long reserve(long now){
8         // 请求时间在下一令牌产生时间之后
9         // 重新计算下一令牌产生时间
10        if (now > next){
11            // 将下一令牌产生时间重置为当前时间
12            next = now;
13        }
14        // 能够获取令牌的时间
15        long at=next;
16        // 设置下一令牌产生时间
17        next += interval;
18        // 返回线程需要等待的时间
19        return Math.max(at, 0L);
20    }
21    // 申请令牌
22    void acquire() {
23        // 申请令牌时的时间
24        long now = System.nanoTime();
```

```

25     // 预占令牌
26     long at=reserve(now);
27     long waitTime=max(at-now, 0);
28     // 按照条件等待
29     if(waitTime > 0) {
30         try {
31             TimeUnit.NANOSECONDS
32                 .sleep(waitTime);
33         }catch(InterruptedException e){
34             e.printStackTrace();
35         }
36     }
37 }
38 }

```

如果令牌桶的容量大于 1，又该如何处理呢？按照令牌桶算法，令牌要首先从令牌桶中出，所以我们需要按需计算令牌桶中的数量，当有线程请求令牌时，先从令牌桶中出。具体的代码实现如下所示。我们增加了一个**resync() 方法**，在这个方法中，如果线程请求令牌的时间在下一令牌产生时间之后，会重新计算令牌桶中的令牌数，**新产生的令牌的计算公式是： $(now - next) / interval$** ，你可对照上面的示意图来理解。reserve() 方法中，则增加了先从令牌桶中出令牌的逻辑，不过需要注意的是，如果令牌是从令牌桶中出的，那么 next 就无需增加一个 interval 了。

 复制代码

```

1  class SimpleLimiter {
2      // 当前令牌桶中的令牌数量
3      long storedPermits = 0;
4      // 令牌桶的容量
5      long maxPermits = 3;
6      // 下一令牌产生时间
7      long next = System.nanoTime();
8      // 发放令牌间隔：纳秒
9      long interval = 1000_000_000;
10
11     // 请求时间在下一令牌产生时间之后，则
12     // 1. 重新计算令牌桶中的令牌数
13     // 2. 将下一个令牌发放时间重置为当前时间
14     void resync(long now) {
15         if (now > next) {
16             // 新产生的令牌数
17             long newPermits=(now-next)/interval;
18             // 新令牌增加到令牌桶
19             storedPermits=min(maxPermits,
20                 storedPermits + newPermits);
21             // 将下一个令牌发放时间重置为当前时间

```



```
22     next = now;
23 }
24 }
25 // 预占令牌, 返回能够获取令牌的时间
26 synchronized long reserve(long now){
27     resync(now);
28     // 能够获取令牌的时间
29     long at = next;
30     // 令牌桶中能提供的令牌
31     long fb=min(1, storedPermits);
32     // 令牌净需求: 首先减掉令牌桶中的令牌
33     long nr = 1 - fb;
34     // 重新计算下一令牌产生时间
35     next = next + nr*interval;
36     // 重新计算令牌桶中的令牌
37     this.storedPermits -= fb;
38     return at;
39 }
40 // 申请令牌
41 void acquire() {
42     // 申请令牌时的时间
43     long now = System.nanoTime();
44     // 预占令牌
45     long at=reserve(now);
46     long waitTime=max(at-now, 0);
47     // 按照条件等待
48     if(waitTime > 0) {
49         try {
50             TimeUnit.NANOSECONDS
51                 .sleep(waitTime);
52         }catch(InterruptedException e){
53             e.printStackTrace();
54         }
55     }
56 }
57 }
```

总结

经典的限流算法有两个，一个是**令牌桶算法 (Token Bucket)**，另一个是**漏桶算法 (Leaky Bucket)**。令牌桶算法是定时向令牌桶发送令牌，请求能够从令牌桶中拿到令牌，然后才能通过限流器；而漏桶算法里，请求就像水一样注入漏桶，漏桶会按照一定的速率自动将水漏掉，只有漏桶里还能注入水的时候，请求才能通过限流器。令牌桶算法和漏桶算法很像一个硬币的正反面，所以你可以参考令牌桶算法的实现来实现漏桶算法。

上面我们介绍了 Guava 是如何实现令牌桶算法的，我们的示例代码是对 Guava RateLimiter 的简化，Guava RateLimiter 扩展了标准的令牌桶算法，比如还能支持预热功能。对于按需加载的缓存来说，预热后缓存能支持 5 万 TPS 的并发，但是在预热前 5 万 TPS 的并发直接就把缓存击垮了，所以如果需要给该缓存限流，限流器也需要支持预热功能，在初始阶段，限制的流速 r 很小，但是动态增长的。预热功能的实现非常复杂，Guava 构建了一个积分函数来解决这个问题，如果你感兴趣，可以继续深入研究。

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。



Java 并发编程实战

全面系统提升你的并发编程能力

王宝令
资深架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 37 | 设计模式模块热点问题答疑

下一篇 39 | 案例分析（二）：高性能网络应用框架Netty

精选留言 (22)

写留言



zsh0103

2019-05-26

1

老师好，问个问题。文中代码 $b=3$ ， $r=1/s$ 时，如果在next之后同时来了3个请求，应该时都可以获得令牌的对吧。就是说这3个请求都可以执行。那岂不是违背了 $r=1/s$ 的限制吗。

展开 ▾

作者回复: 按照令牌桶算法是这样的，所以b不能搞得太大

◀ ▶



辣椒

2019-06-04



// 令牌净需求：首先减掉令牌桶中的令牌

```
long nr = 1 - fb;
```

```
// 重新计算下一令牌产生时间
```

```
next = next + nr*interval;
```

```
// 重新计算令牌桶中的令牌...
```

展开 ▾



涛哥迷妹

2019-05-30



public static RateLimiter create(double permitsPerSecond) {...} 创建时候并没有 burst 参数啊。请问在哪类里设置的



涛哥迷妹

2019-05-30



guava ratelimiter 容量上限在哪个参数中体现或者在哪设置这个。比如我们设置的流速是2/s,当100s之内都没有请求到来，是不是会往令牌桶中持续放入200个令牌，而这这时候突然来了一波300个并发请求，是不是200个请求可以被调用，剩下100个请求被阻塞慢慢释放。是这样的？

作者回复: burst参数控制

◀ ▶



涛哥迷妹

2019-05-30



```
long interval = 1000_000_000;
```

这是什么写法



涛哥迷妹

2019-05-30



容量上限b怎么设置

展开 ▾



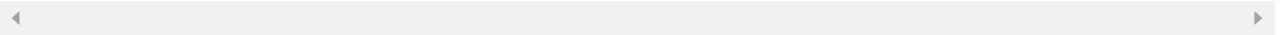
andy

2019-05-29



我有个疑问，这个令牌桶算法，多线程当中不会有问题么？还是我认为的使用场景不对，有点蒙

作者回复: 多线程没问题



曾轼麟

2019-05-28



其次是信号量其实也有限流的方式，比如redisson里面提供的超时信号量，既有信号量的功能也有限流器的功能



曾轼麟

2019-05-28



老师我是这样理解的，从安全角度来说，信号量是要优于限流器的，比如前面的请求还没处理完，信号量不允许新的请求进来，而限流器允许



Sunqc

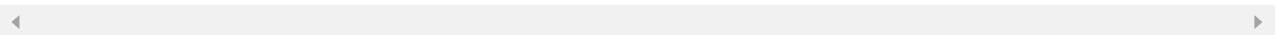
2019-05-28



皮一下，老王，这一节挺深奥的，哈哈哈

展开 ▾

作者回复: 小心暴露智商 😊





Darren

2019-05-26



老师，请教一下，限流器和信号量为什么感觉一样的，那为什么2个还都存在？是因为业务场景不同吗？请老师解惑下

作者回复: 限流器不需要释放操作，信号量没办法控制带时间范围的限流，只能用于非常简单的场景



爱吃回锅肉...

2019-05-26



老师，有没什么资料推荐关于guava预热功能呢？主要网上资料太繁杂，不知道要如何甄别哪些是比较经典的

作者回复: 能把为什么用的是那个积分函数，而不是用其他积分函数讲清楚的，应该比较好的。最好是看guava的代码注释，写的非常详细。



锦

2019-05-25



很精彩！老师应该去讲数据结构与算法:)

展开 ∨

作者回复: 何必难为自己呢，不讲了😁



QQ怪

2019-05-25



读了几遍才基本弄懂，有点深度了哈

展开 ∨



松花皮蛋me

2019-05-25



分布式下整体按服务限流呢？

展开 ▾



张三

2019-05-25



打卡！令牌桶容量大于1的部分没看懂。

展开 ▾



密码123456

2019-05-25



桶容量为1的时候，我能理解。但是桶容量为多个的时候，就不理解了，比如

// 新产生的令牌数

```
long newPermits=(now-next)/interval;
```

这句，不应该1秒生成桶的总容量吗？假设now为2，next为1。interval也为1。那么一个周期也就产生一个令牌啊？

展开 ▾



遇见阳光

2019-05-25



RateLimiter这个限流器和juc包的信号量有啥区别？

展开 ▾



undefined

2019-05-25



对于这个请求令牌的线程而言，很显然需要等待 1 秒，因为 1 秒以后（第 3 秒）它就能拿到令牌了。此时需要注意的是，下一个令牌发放的时间也要增加 1 秒，为什么呢？因为第 3 秒发放的令牌已经被线程 T1 预占了。处理之后如下图所示。

“下一个令牌发放的时间也要增加 1 秒”这句话没懂，下一个令牌是指可以下一次请求...

展开 ▾



搏未来

2019-05-25



当我看到积分函数时，感觉数学也要好好学习了😓

展开 ▾

