# 一、问题描述

　　某天A君突然发现自己的接口请求量突然涨到之前的10倍，没多久该接口几乎不可使用，并引发连锁反应导致整个系统崩溃。如何应对这种情况呢？生活给了我们答案：比如老式电闸都安装了保险丝，一旦有人使用超大功率的设备，保险丝就会烧断以保护各个电器不被强电流给烧坏。同理我们的接口也需要安装上“保险丝”，以防止非预期的请求对系统压力过大而引起的系统瘫痪，当流量过大时，可以采取拒绝或者引流等机制。

# 二、常用的限流算法

      常用的限流算法有两种：漏桶算法和令牌桶算法。

      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。

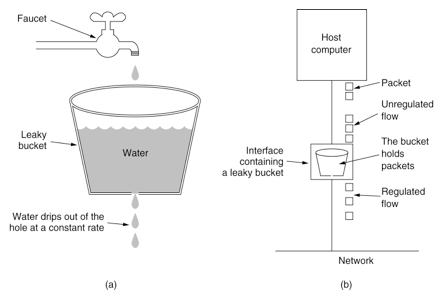


图1 漏桶算法示意图

      对于很多应用场景来说，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。这时候漏桶算法可能就不合适了，令牌桶算法更为适合。如图2所示，令牌桶算法的原理是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。

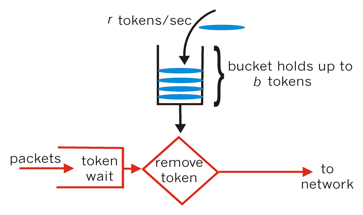


图2 令牌桶算法示意图

# 三、限流工具类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来完成限流，非常易于使用。RateLimiter类的接口描述请参考：[RateLimiter接口描述](http://docs.guava-libraries.googlecode.com/git-history/master/javadoc/com/google/common/util/concurrent/RateLimiter.html)，具体使用请参考：[RateLimiter使用实践](http://java.dzone.com/articles/ratelimiter-discovering-google)。

      下面是主要源码：

[复制代码](javascript:void(0);)

public double acquire() {

return acquire(1);

}

public double acquire(int permits) {

checkPermits(permits); //检查参数是否合法（是否大于0）

long microsToWait;

synchronized (mutex) { //应对并发情况需要同步

microsToWait = reserveNextTicket(permits, readSafeMicros()); //获得需要等待的时间

}

ticker.sleepMicrosUninterruptibly(microsToWait); //等待，当未达到限制时，microsToWait为0

return 1.0 \* microsToWait / TimeUnit.SECONDS.toMicros(1L);

}

private long reserveNextTicket(double requiredPermits, long nowMicros) {

resync(nowMicros); //补充令牌

long microsToNextFreeTicket = nextFreeTicketMicros - nowMicros;

double storedPermitsToSpend = Math.min(requiredPermits, this.storedPermits); //获取这次请求消耗的令牌数目

double freshPermits = requiredPermits - storedPermitsToSpend;

long waitMicros = storedPermitsToWaitTime(this.storedPermits, storedPermitsToSpend)

+ (long) (freshPermits \* stableIntervalMicros);

this.nextFreeTicketMicros = nextFreeTicketMicros + waitMicros;

this.storedPermits -= storedPermitsToSpend; // 减去消耗的令牌

return microsToNextFreeTicket;

}

private void resync(long nowMicros) {

// if nextFreeTicket is in the past, resync to now

if (nowMicros > nextFreeTicketMicros) {

storedPermits = Math.min(maxPermits,

storedPermits + (nowMicros - nextFreeTicketMicros) / stableIntervalMicros);

nextFreeTicketMicros = nowMicros;

}

}

[复制代码](javascript:void(0);)

转载请标明源地址：http://www.cnblogs.com/LBSer