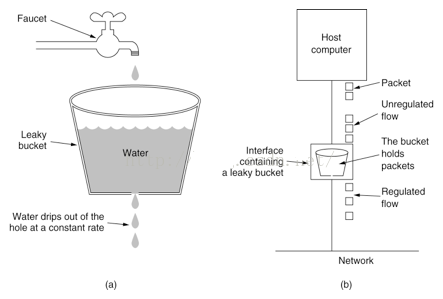
# 限流算法

常用的限流算法有两个：漏桶算法和令牌桶算法

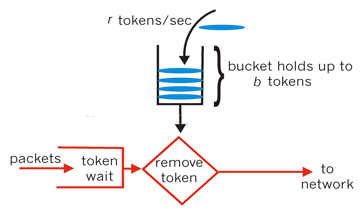
# 漏桶算法

漏桶算法原理比较简单，请求进入到漏桶中，漏桶以一定的速率漏水。当请求过多时，水直接溢出，可以看出，漏桶算法可以强制限制数据的传输速度，如下图：



# 令牌桶算法

令牌桶算法的原理是以一定速率向桶中放入令牌，如果有请求时，请求会从桶中取出令牌，如果能取到令牌，则可以继续完成请求，否则等待或者拒绝服务。这种算法可以应对突发程度的请求，因此比漏桶算法好，如下图：



令牌桶算法如下：

* 每秒会有r个令牌放入桶中，或者每过1/r秒桶中增加一个令牌
* 桶中最多存放b个令牌，如果桶满了，新放入的令牌会被丢弃
* 当一个n字节的数据包到达时，消耗n个令牌，然后发送该数据包
* 如果桶中可用令牌小于n，则该数据包将缓存或者丢弃

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。使用示例如下：

1. 添加guava依赖

*<dependency>*

*<groupId>com.google.guava</groupId>*

*<artifactId>guava</artifactId>*

*<version>23.0</version>*

*</dependency>*

1. 测试程序

*RateLimiter rateLimiter = RateLimiter.create(2);*

*//每秒QPS为2，如果不超过1000，则设置为RateLimiter.create(1000)*

*System.out.println(rateLimiter.acquire(5));*

*System.out.println(rateLimiter.acquire(2));*

*System.out.println(rateLimiter.acquire(1));*

程序中，每秒只能产生2个令牌，第一次取出5个，但是第二个再去取令牌的时候需要等2.5秒，也就是第一次令牌取完之后，需要等2.5秒才能取到令牌。同样，第三次取1个令牌的时候，也需要等待第二次的1秒时间。总之取的速率可以超过令牌产生的速率，但是下一次取的时候，需要阻塞等待。

http://ifeve.com/guava-ratelimiter/

https://blog.csdn.net/charleslei/article/details/53152883

https://segmentfault.com/a/1190000012875897

http://www.cnblogs.com/exceptioneye/p/4824394.html