# 分布式环境下限流方案的实现

* 业务背景介绍   
  对于web应用的限流，光看标题，似乎过于抽象，难以理解，那我们还是以具体的某一个应用场景来引入这个话题吧。   
  在日常生活中，我们肯定收到过不少不少这样的短信，“双11约吗？，千款….”，“您有幸获得唱读卡，赶快戳链接…”。这种类型的短信是属于推广性质的短信。为什么我要说这个呢？听我慢慢道来。   
  一般而言，对于推广营销类短信，它们针对某一群体(譬如注册会员)进行定点推送，有时这个群体的成员量比较大，譬如京东的会员，可以达到千万级别。因此相应的，发送推广短信的量也会增大。然而，要完成这些短信发送，我们是需要调用服务商的接口来完成的。倘若一次发送的量在200万条，而我们的服务商接口每秒能处理的短信发送量有限，只能达到200条每秒。那么这个时候就会产生问题了，我们如何能控制好程序发送短信时的速度昵？于是限流这个功能就得加上了
* 生产环境背景   
  1、服务商接口所能提供的服务上限是400条/s   
  2、业务方调用短信发送接口的速度未知，QPS可能达到800/s，1200/s，或者更高   
  3、当服务商接口访问频率超过400/s时，超过的量将拒绝服务，多出的信息将会丢失   
  4、线上为多节点布置，但调用的是同一个服务商接口
* 需求分析   
  1、鉴于业务方对短信发送接口的调用频率未知，而服务商的接口服务有上限，为保证服务的可用性，业务层需要对接口调用方的流量进行限制—–接口限流
* 需求设计   
  方案一、在提供给业务方的Controller层进行控制。   
  1、使用guava提供工具库里的RateLimiter类(内部采用令牌捅算法实现)进行限流

<!--核心代码片段-->

private RateLimiter rateLimiter = RateLimiter.create(400);//400表示每秒允许处理的量是400

if(rateLimiter.tryAcquire()) {

//短信发送逻辑可以在此处

}

2、使用Java自带delayqueue的延迟队列实现(编码过程相对麻烦，此处省略代码)

3、使用redis实现，存储两个key，一个用于计时，一个用于计数。请求每调用一次，计数器增加1，若在计时器时间内计数器未超过阈值，则可以处理任务

if(!cacheDao.hasKey(API\_WEB\_TIME\_KEY)) { cacheDao.putToValue(API\_WEB\_TIME\_KEY,0,(long)1, TimeUnit.SECONDS);

} if(cacheDao.hasKey(API\_WEB\_TIME\_KEY)&&cacheDao.incrBy(API\_WEB\_COUNTER\_KEY,(long)1) > (long)400) {

LOGGER.info("调用频率过快");

}

//短信发送逻辑

方案二、在短信发送至服务商时做限流处理   
方案三、同时使用方案一和方案二

* 可行性分析   
  最快捷且有效的方式是使用RateLimiter实现，但是这很容易踩到一个坑，单节点模式下，使用RateLimiter进行限流一点问题都没有。但是…线上是分布式系统，布署了多个节点，而且多个节点最终调用的是同一个短信服务商接口。虽然我们对单个节点能做到将QPS限制在400/s，但是多节点条件下，如果每个节点均是400/s，那么到服务商那边的总请求就是***节点*数x400/s**，于是限流效果失效。使用该方案对单节点的阈值控制是难以适应分布式环境的，至少目前我还没想到更为合适的方式。   
  对于第二种，使用delayqueue方式。其实主要存在两个问题，1：短信系统本身就用了一层消息队列，有用kafka，或者rabitmq，如果再加一层延迟队列，从设计上来说是不太合适的。2：实现delayqueue的过程相对较麻烦，耗时可能比较长，而且达不到精准限流的效果   
  对于第三种，使用redis进行限流，其很好地解决了分布式环境下多实例所导致的并发问题。因为使用redis设置的计时器和计数器均是全局唯一的，不管多少个节点，它们使用的都是同样的计时器和计数器，因此可以做到非常精准的流控。同时，这种方案编码并不复杂，可能需要的代码不超过10行。
* 实施方案   
  根据可行性分析可知，整个系统采取redis限流处理是成本最低且最高效的。   
  **具体实现**

1、在Controller层设置两个全局key，一个用于计数，另一个用于计时

private static final String API\_WEB\_TIME\_KEY = "time\_key";

private static final String API\_WEB\_COUNTER\_KEY = "counter\_key";

2、对时间key的存在与否进行判断，并对计数器是否超过阈值进行判断

if(!cacheDao.hasKey(API\_WEB\_TIME\_KEY)) {

cacheDao.putToValue(API\_WEB\_TIME\_KEY,0,(long)1, TimeUnit.SECONDS);

cacheDao.putToValue(API\_WEB\_COUNTER\_KEY,0,(long)2, TimeUnit.SECONDS);//时间到就重新初始化为

}

if(cacheDao.hasKey(API\_WEB\_TIME\_KEY)&&cacheDao.incrBy(API\_WEB\_COUNTER\_KEY,(long)1) > (long)400) {

LOGGER.info("调用频率过快");

}

//短信发送逻辑

**实施结果**   
可以达到非常精准的流控，截图会在后续的过程中贴出来。欢迎有疑问的小伙伴们在评论区提出问题，我看到后尽量抽时间回答的

版权声明：本文为博主原创文章，未经博主允许不得转载。