分布式限流设计与实现

人员：李斌
部门：酒店研发部
时间：2018-05-14

京东

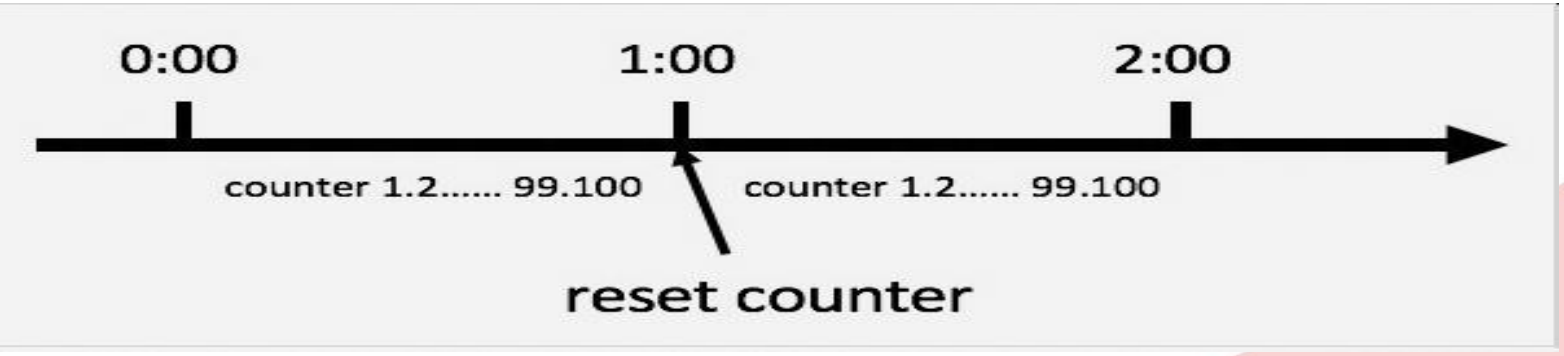
目录

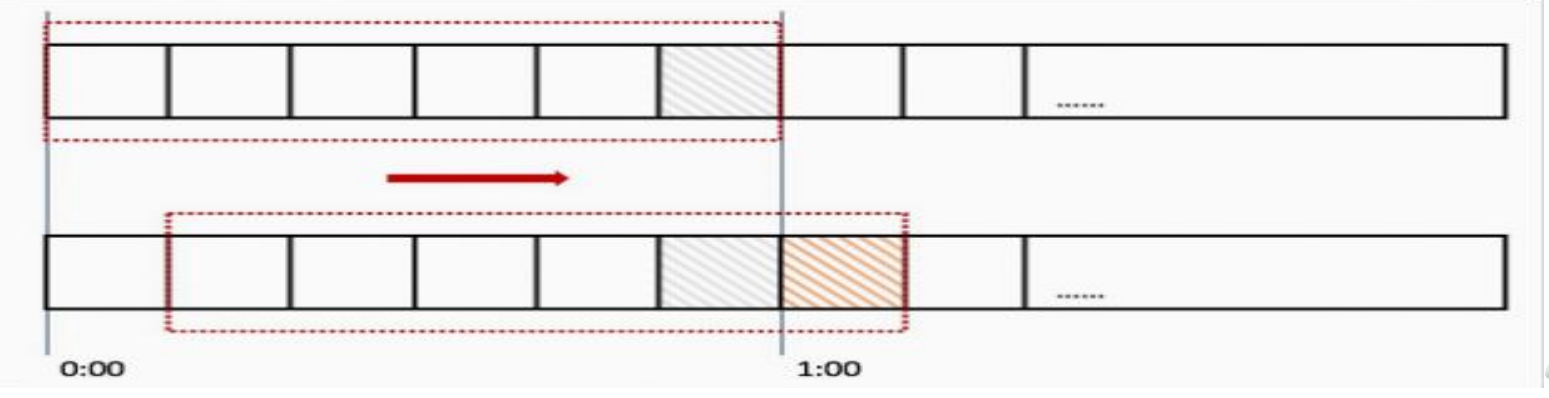
CONTENTS

京东

现有限流算法介绍

- **计数器：**主要用来限制总并发数，比如数据库连接池、线程池、秒杀的并发数；只要全局总请求数或者一定时间段的总请求数设定的阀值则进行限流，是简单粗暴的总数量限流，而不是平均速率限流

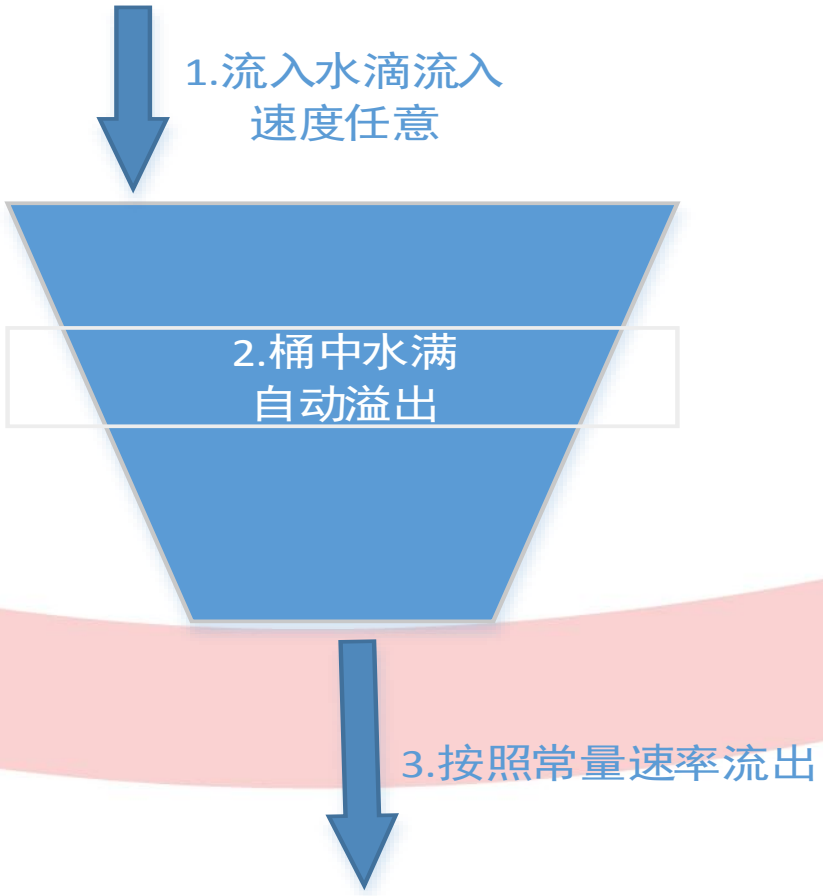




京东

现有限流算法介绍

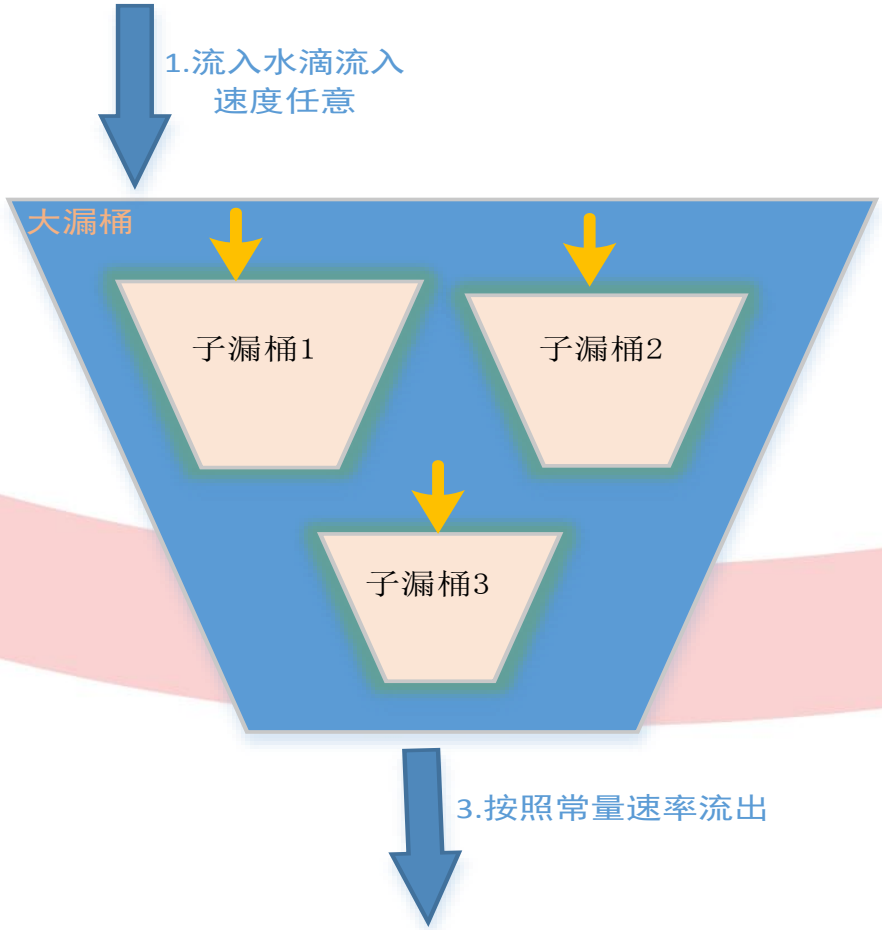
* 漏桶（Leaky Bucket）：水（请求）先进入到漏桶里，漏桶以一定的速度出水（接口有响应速率），当水流入速度过大会直接溢出（请求拒绝）



京东

现有限流算法介绍

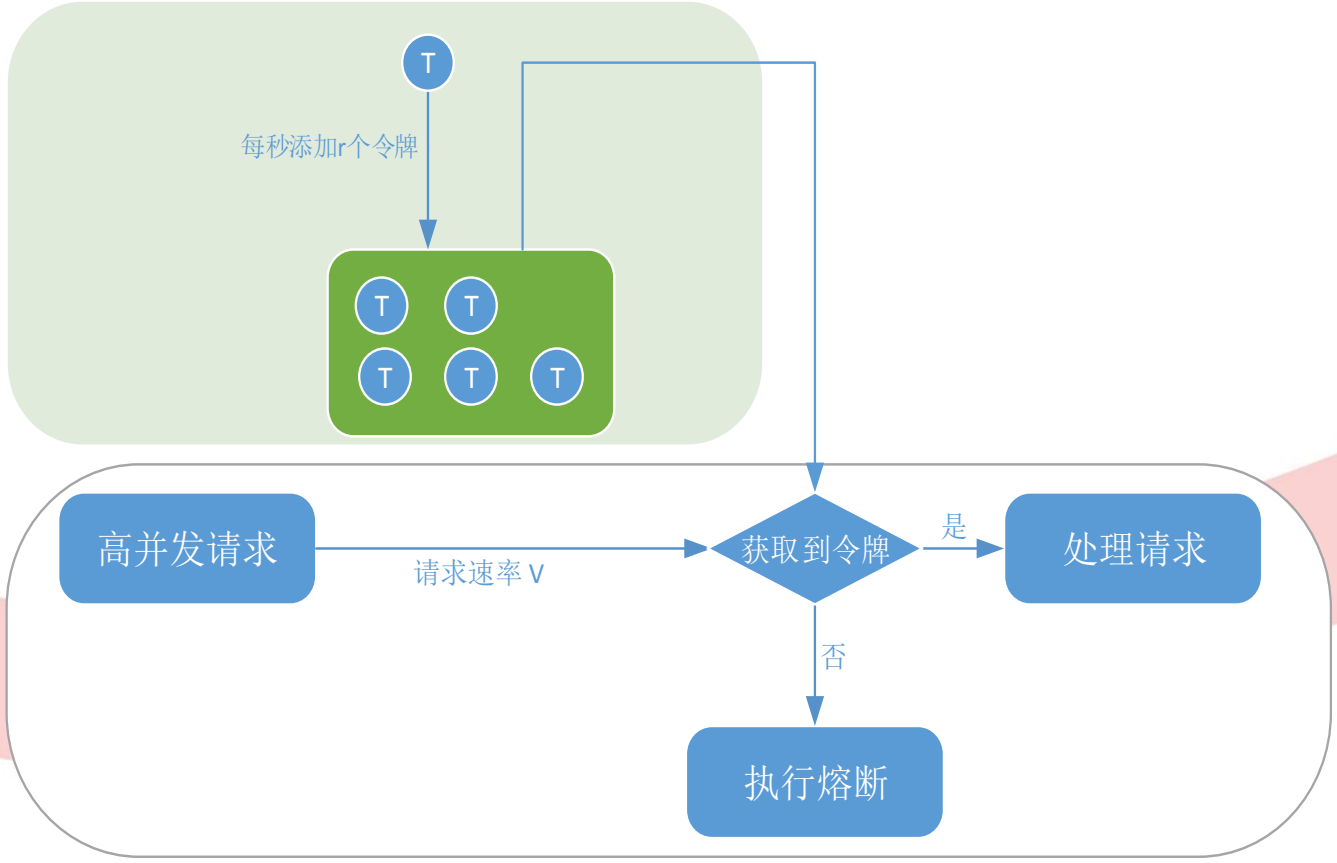
- **多漏桶（漏桶改进版）**：同一漏桶但具有不同权重优先级的一组请求(报文)进行限速的时候，将总漏桶按照权重优先级的个数及比例参数分割为多个子漏桶(报文权重优先级与子漏桶权重优先级相对应)，各个子漏桶突发度总和等于大漏桶的突发度且各个子漏桶水流流出速率总和等于大漏桶流出速率



京东

现有限流算法介绍

- **令牌桶（Token Bucket）：**和漏洞漏水相反，令牌桶是系统以恒定速率往桶中加令牌（桶满了则溢出）新请求到达直接从桶中拿走需要的令牌数，如果桶中剩余令牌数不够则拒绝请求



京东

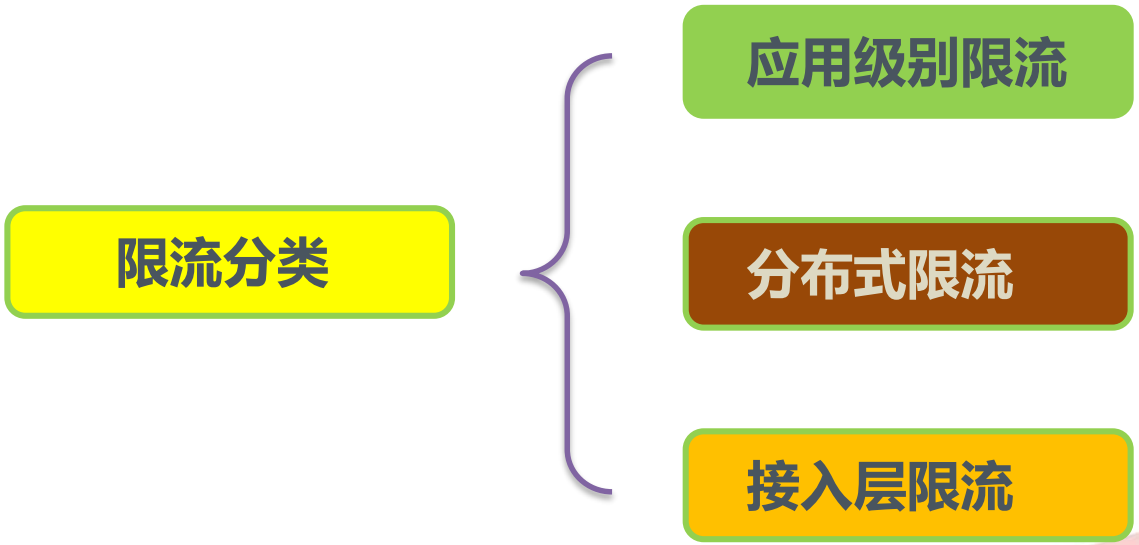
现有限流算法介绍

* 算法对比:

	令牌桶	漏桶
机制	按照固定速率往桶中添加令牌（限制流入速度），请求是否被处理需要看桶中令牌是否足够，当令牌数减为零时则拒绝新的请求；	按照常量固定速率流出请求（限制流出速度），流入请求速率任意，当流入的请求数累积到漏桶容量时，则新流入的请求被拒绝；
目的	容忍一定程度的突发	平滑流入速率
	两个算法实现可以一样，但是方向是相反的，对于相同的参数得到的限流效果一样（均不存在临界问题）	

京东

现有限流方案介绍



京东

现有限流方案介绍

应用级限流	
限流总（并发/连接/请求/资源）数	系统TPS/QPS限制，Web容器总的连接数、连接池、线程池等资源限制 例如：Tomcat Connector配置： acceptCount、maxConnections、maxThreads等参数配置

限流某个接口的总并发/请求数或时间窗内的请求数	1. AtomicLong（原子操作）： atomicLong.incrementAndGet(); //*增*
	atomicLong.decrementAndGet(); //*减*
	2. Semaphore(信号量) : semaphore.acquire();
	semaphore.release();
	3. Guava Cache 缓存计数器

平滑限流某个接口的请求数（限速率）	Guava RateLimiter (平滑突发限流(SmoothBursty：令牌生成速度恒定(令牌桶))和平滑预热限流(SmoothWarmingUp：令牌生成速度缓慢提升直到维持在一个稳定值 (漏桶)))

京东

现有限流方案介绍

```
try {
    if(atomic.incrementAndGet() > rateLimit) {
        //熔断
    }
    //处理请求
} finally {
    atomic.decrementAndGet();
}
```

```
/**限制每秒提交两个任务*/
final RateLimiter rateLimiter = RateLimiter.create(2.0);
void submitTasks(List<Runnable> tasks) {
    for (Runnable task : tasks) {
        rateLimiter.acquire(); // 也许需要等待
        //执行任务
    }
}
```

```
/**限制每秒的调用次数*/
/**初始化 cache 存储结构：<当前时间， 调用次数>*/
LoadingCache<Long, AtomicLong> counter = CacheBuilder.newBuilder()
                .expireAfterWrite(2, TimeUnit.SECONDS) //过期时间设置为2S
                .build(new CacheLoader<Long, AtomicLong>() {
                    @Override
                    public AtomicLong load(Long seconds) throws Exception {
                        return new AtomicLong(0);
                    }
                });
/*调用*/
while(true) {
    if(counter.get(System.currentTimeMillis() / 1000).incrementAndGet() > rateLimit) {
        //执行熔断
        continue;
    }
    //业务处理
}
```

京东

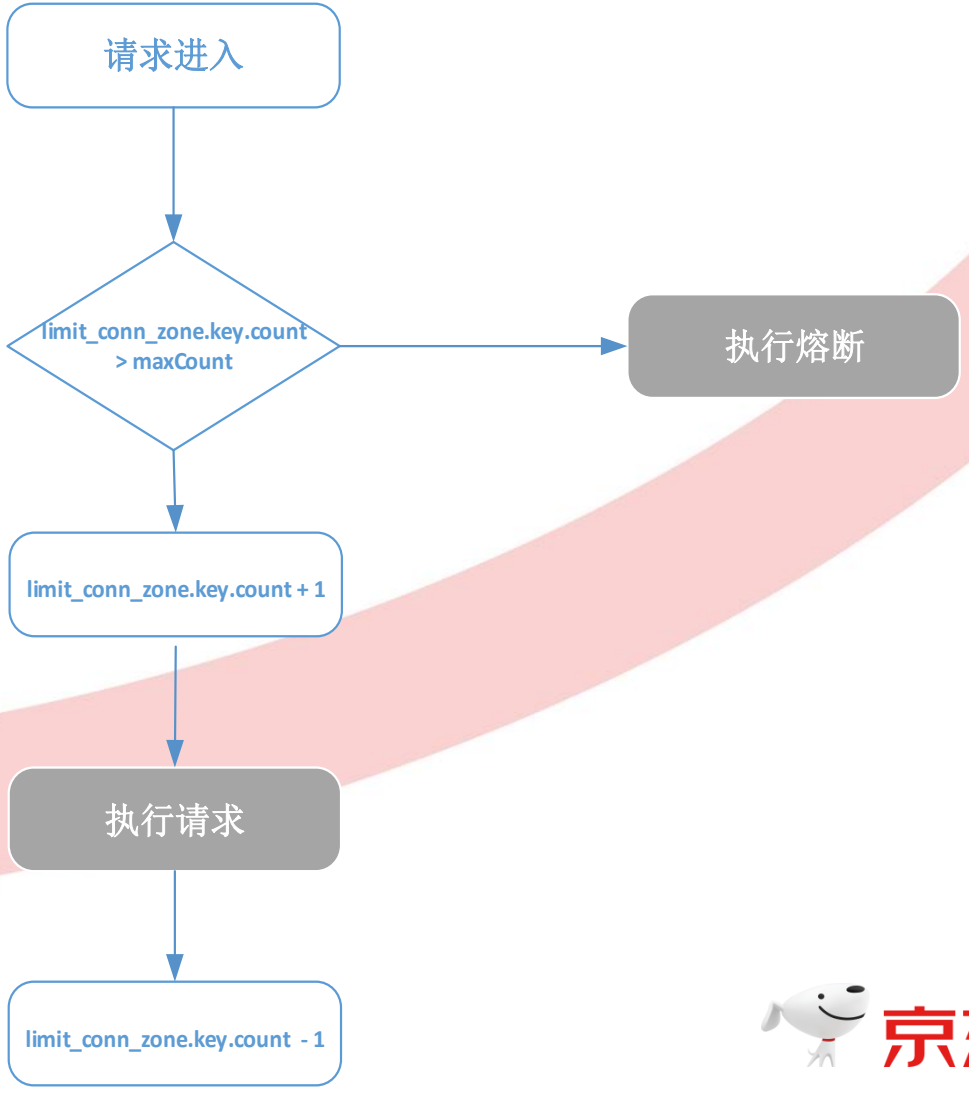
现有限流方案介绍

接入层限流（流量入口限流）	
目的	负载均衡、非法请求过滤、请求聚合、缓存、降级、限流、A/B测试、服务质量监控等等
流程方案	基于Nginx实现： ngx_http_limit_conn_module：限制总的连接数 ngx_http_limit_req_module： 限制请求速率（漏桶算法） lua-resty-limit-traffic ： 灵活实现限流算法 （基于OpenResty提供lua限流模块）

京东

现有限流方案介绍

```
http {
    limit_conn_zone$binary_remote_addr zone=perip:10m;
    #配置限流KEY及存放KEY对应信息的共享内存区域大小；此处KEY表示IP地址
    # limit_conn_zone $ server_name zone=perserver:10m;
    limit_conn_log_level error; #配置记录被限流后的日志级别，默认error级别
    limit_conn_status 503; #配置被限流后返回的状态码，默认返回503
    ...
    server {
    ...
    location /limit {
        limit_conn perip/perserver  1; # 限制ip或者域名最大并发连接数
    }
```

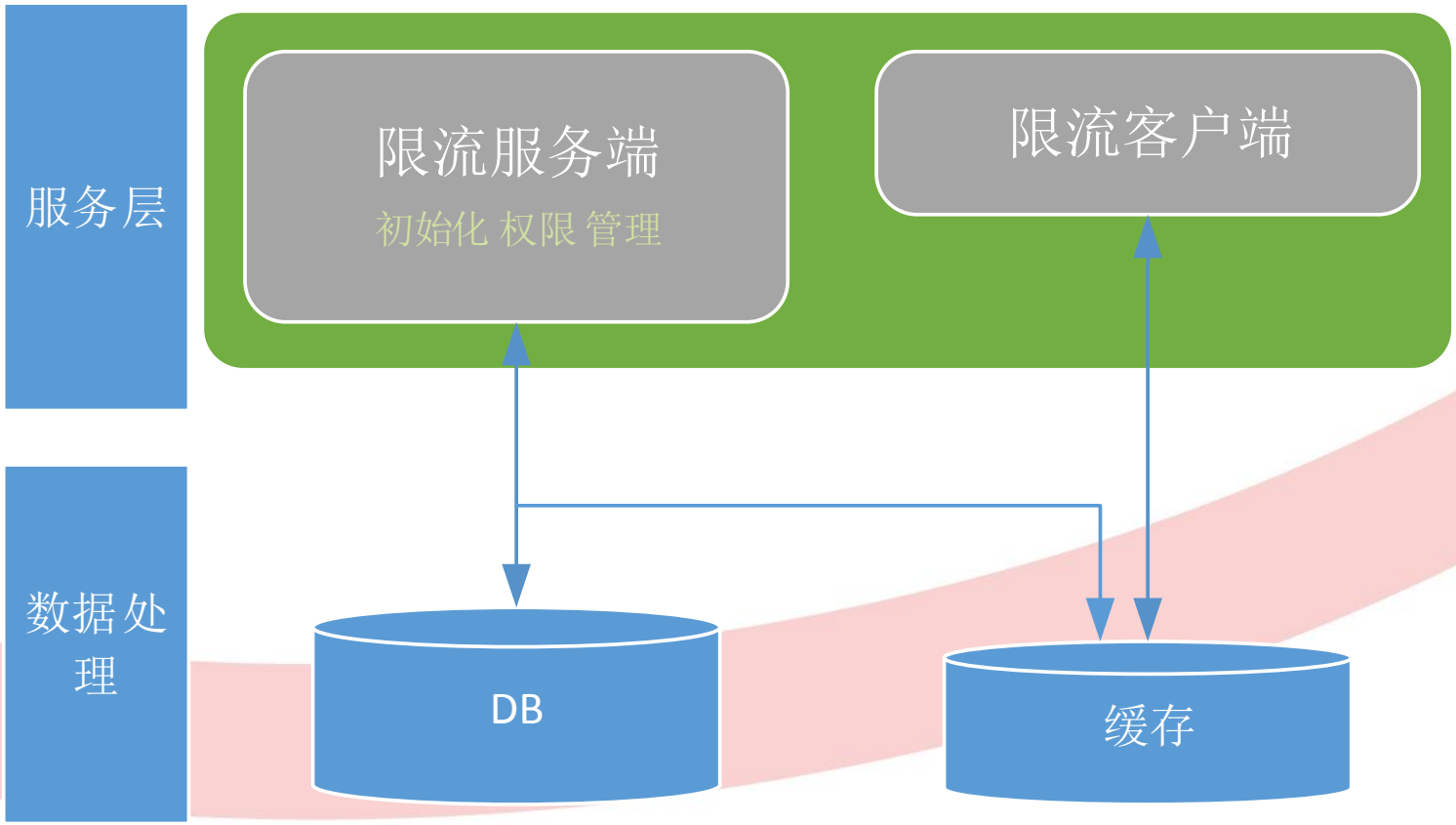


现有限流方案介绍

分布式限流	
问题	应用被部署到多台机器，应用级限流方式只是单应用内的请求限流，不能进行全局限流，怎样保证多系统之间限流操作一致性？
解决方式	将限流服务做成原子化
目前流程方案	基于缓存 + lua 或者 Nginx + lua 实现 （利用lua脚本执行的原子性特征）

京东

现有限流方案介绍



京东

分布式优化可切入点

基于单漏桶或者令牌桶实现的分布式限流机制一般适用于一个类别的业务限流

适用面太窄

目前流行的双/多漏桶算法计算复杂，桶维护繁琐

复杂度问题

分布式系统限流和普通宽带限流不同，需要依据应用平台的不同，考虑不同影响因素

系统性能影响

京东

分布式限流优化方式

01

可扩展性

基于令牌桶引入优先级策略，基于应用设置应用内不同类型请求的优先级，扩展适用场景

02

实时性

基于单令牌桶实现双/多漏桶的功能。基于分布式系统特点，将基于任务将一些复杂计算逻辑分离出去

03

稳定性

不再只是依据初始化设置做流速控制，同时需要考虑系统实时性能，外部突发变动等因素实现一种自动动态调整的流速控制机制

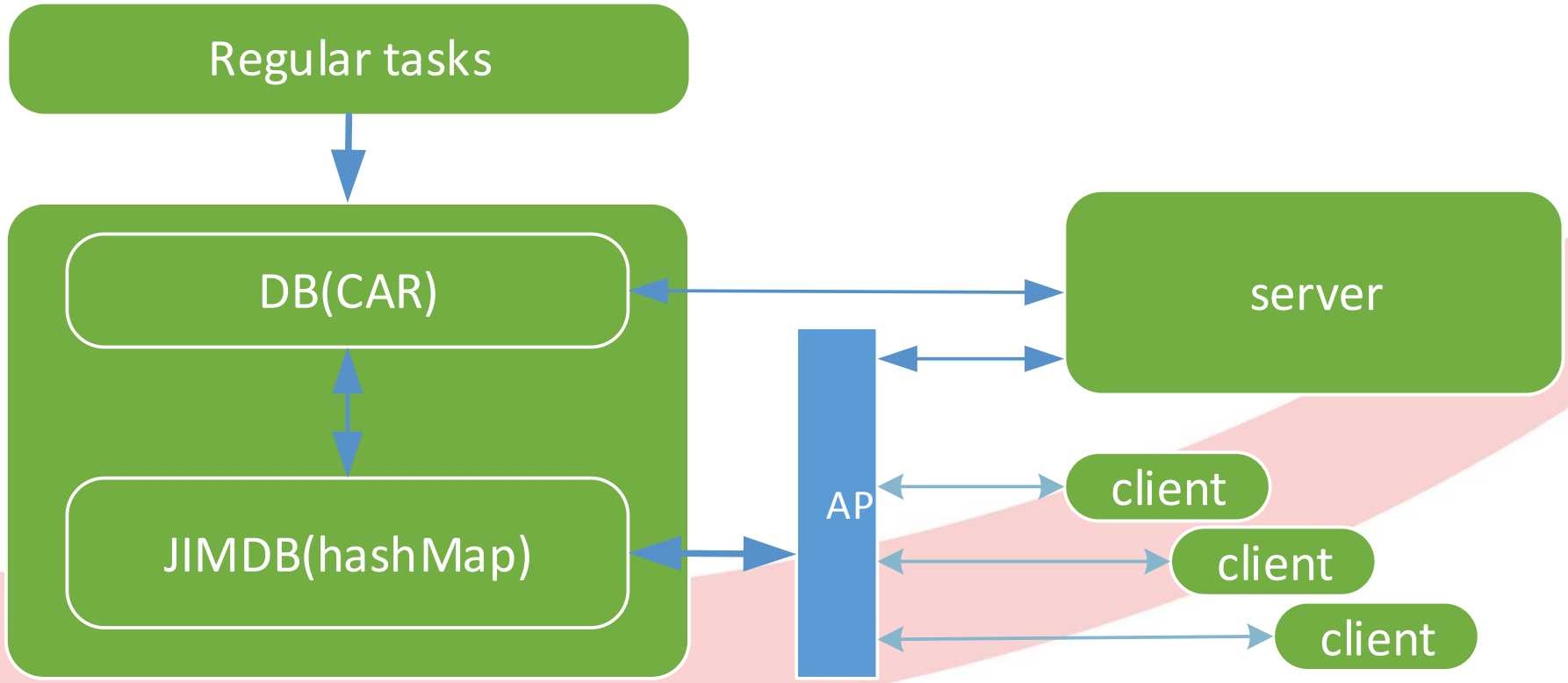
04

操作性

尽可能降低人为操作复杂度和调整次数

京东

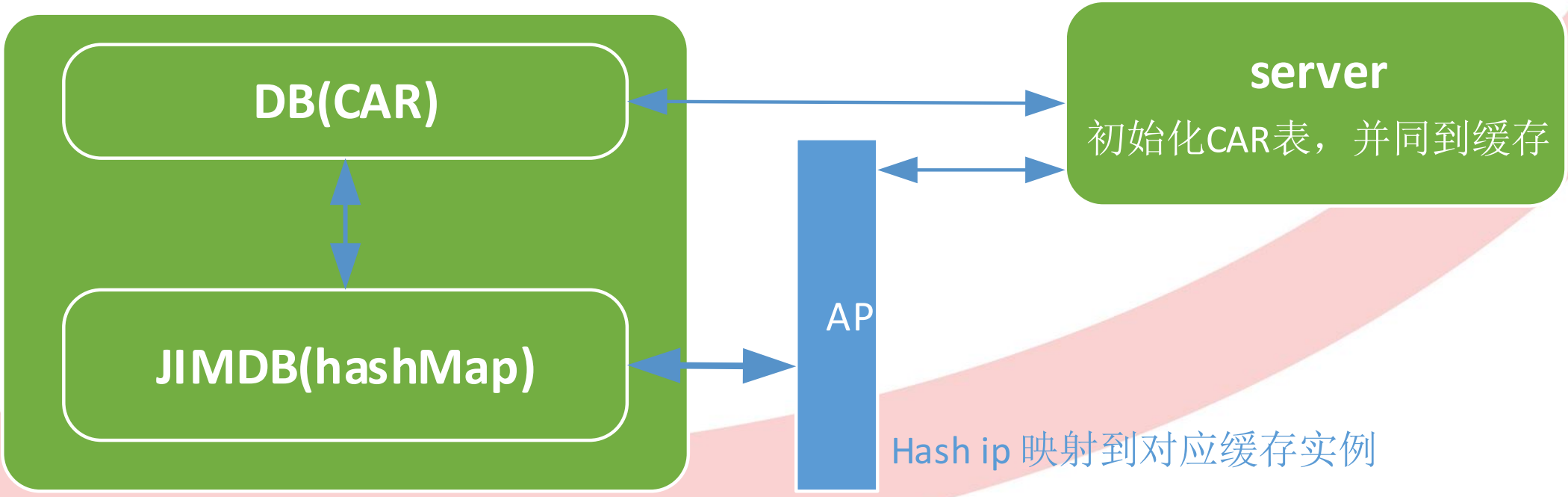
基于令牌桶的分布式限流优化算法



京东

分布式限流优化算法

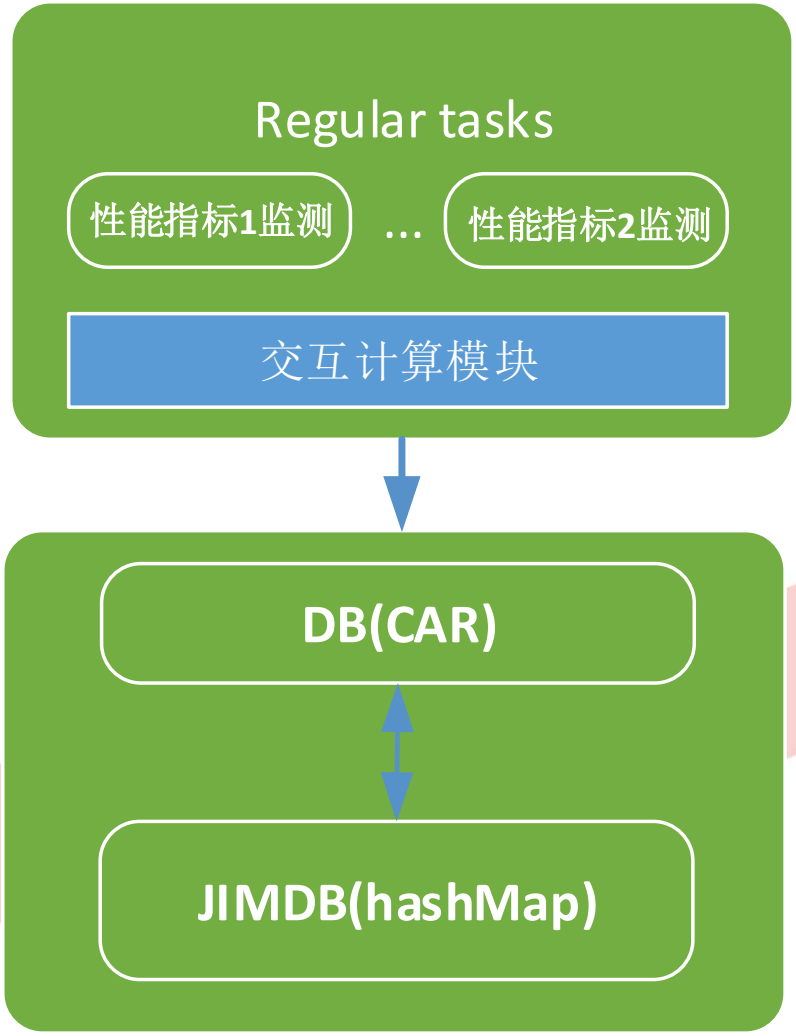
1. ratelimiter-server



京东

分布式限流优化算法

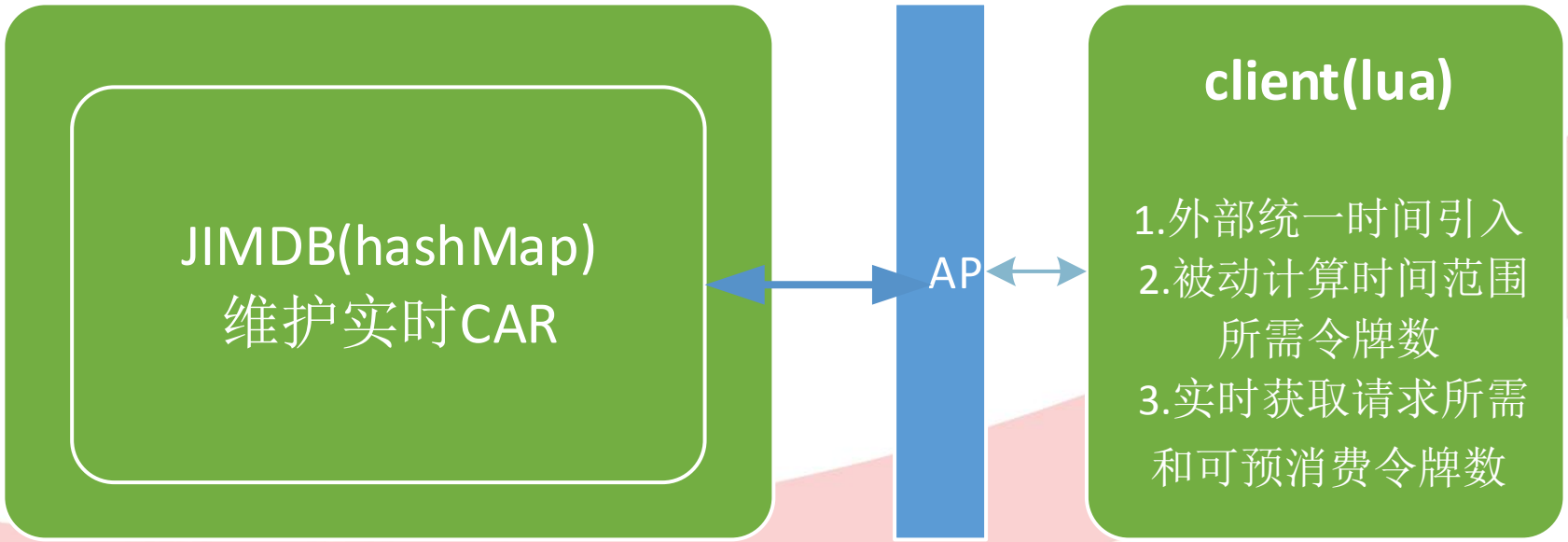
2. task



京东

分布式限流优化算法

3. ratelimiter-client



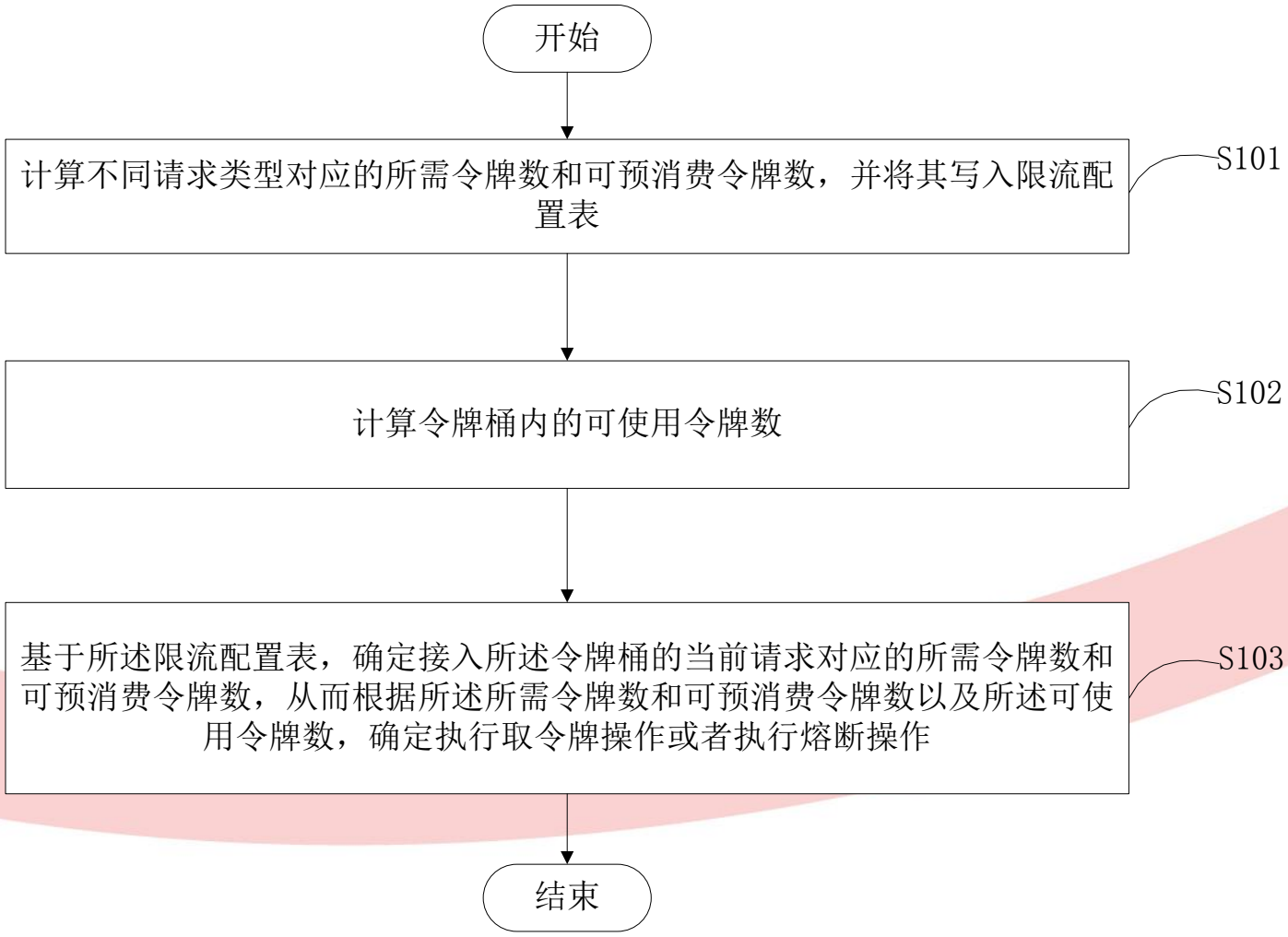
京东

分布式限流优化算法

CAR(Committed Access Rate) 表	
app	请求所属应用标识
key	请求类别标识
rate	令牌添加速率
max_permits	桶突发度
need_permits	对应key请求所需令牌数
pre_permits	对应key可预消费令牌数
G	对应key请求优先级
U	调节因子
extras	备选字段

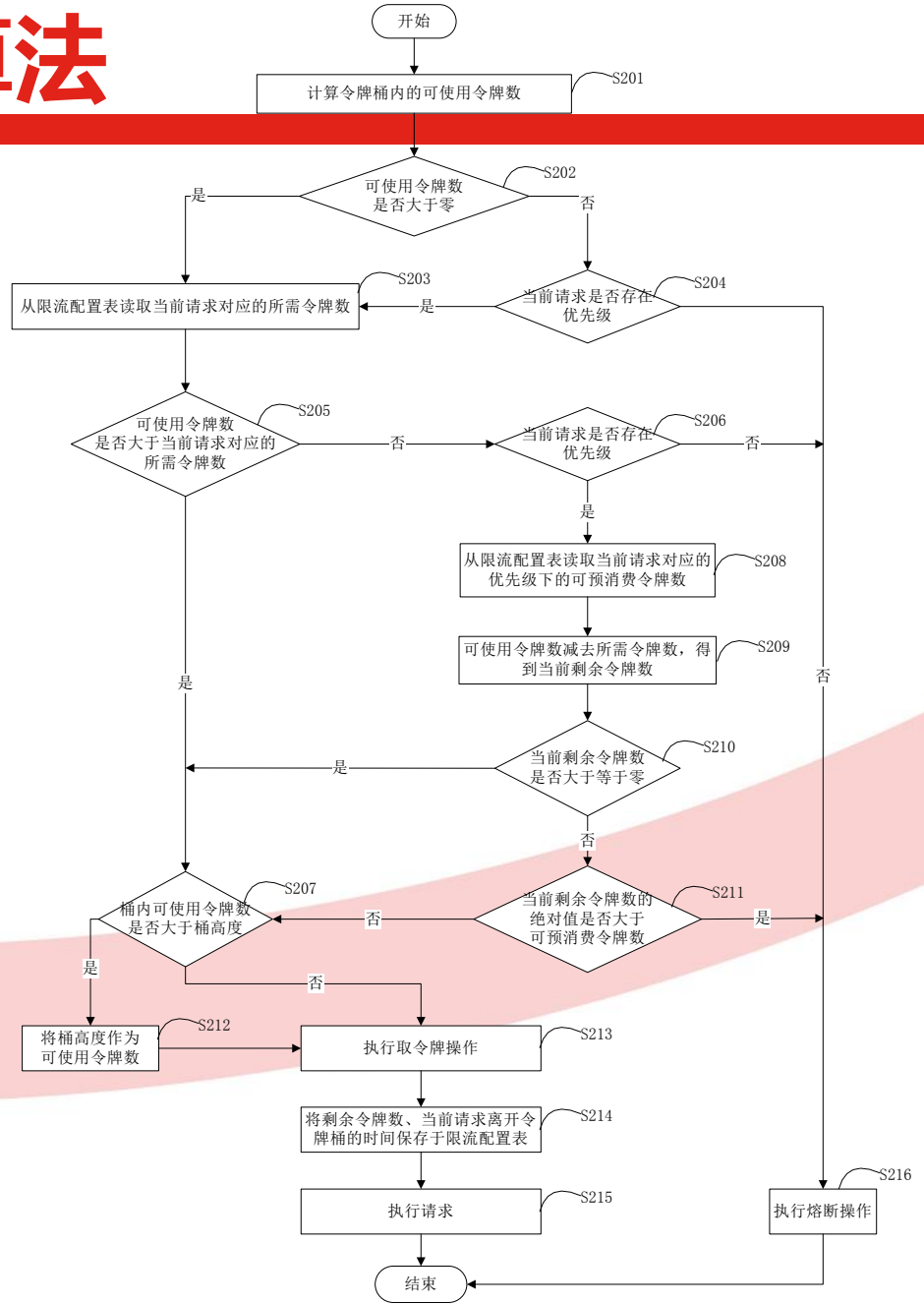
京东

分布式限流优化算法



京东

分布式限流优化算法

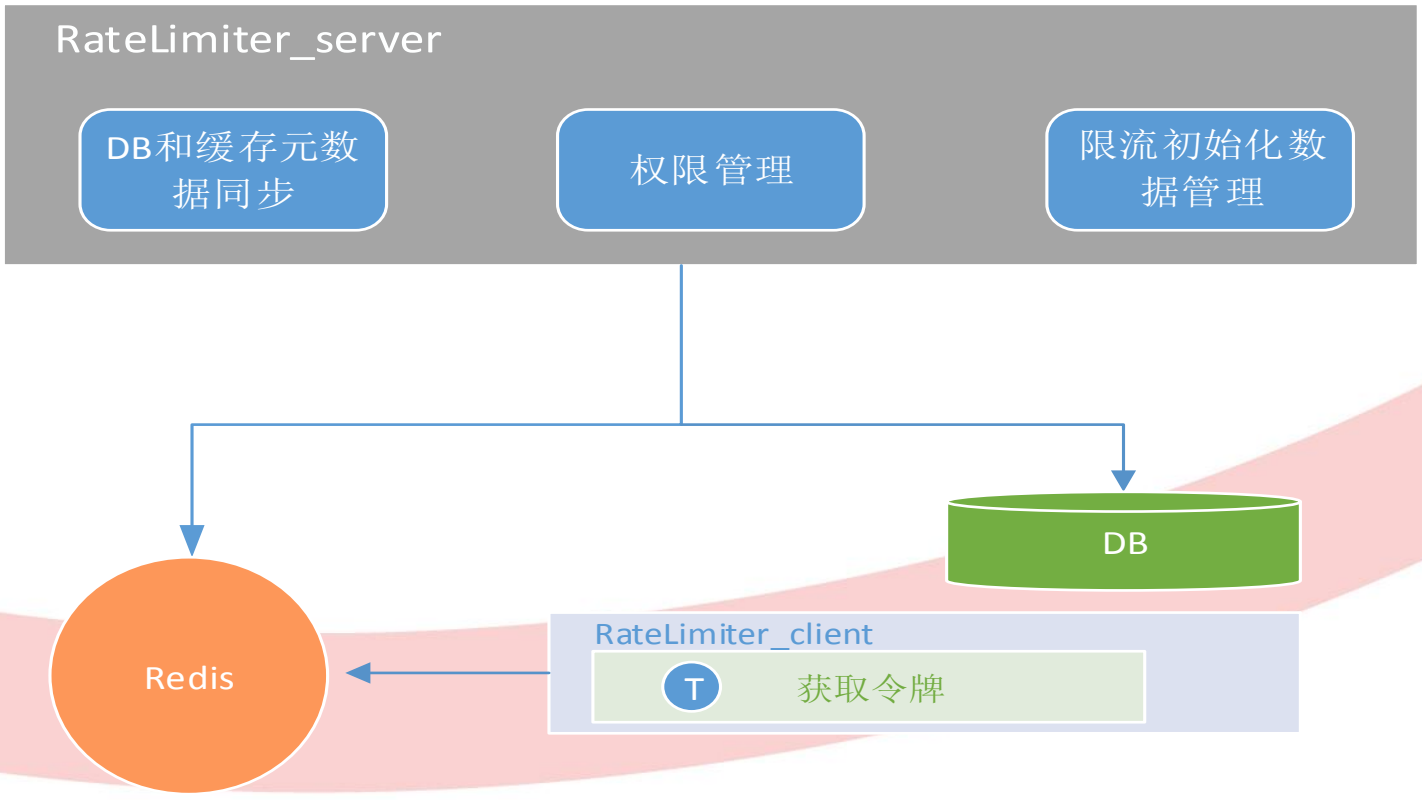

开始
计算令牌桶内的可使用令牌数
S201
可使用令牌数是否大于零
S202
是
否
从限流配置表读取当前请求对应的所需令牌数
S203
是
当前请求是否存在优先级
S204
可使用令牌数是否大于当前请求对应的所需令牌数
S205
否
当前请求是否存在优先级
S206
否
是
从限流配置表读取当前请求对应的优先级下的可预消费令牌数
S208
可使用令牌数减去所需令牌数，得到当前剩余令牌数
S209
是
否
当前剩余令牌数是否大于等于零
S210
是
否
桶内可使用令牌数是否大于桶高度
S207
否
当前剩余令牌数的绝对值是否大于可预消费令牌数
S211
是
是
否
将桶高度作为可使用令牌数
S212
执行取令牌操作
S213
将剩余令牌数、当前请求离开令牌桶的时间保存于限流配置表
S214
执行请求
S215
执行熔断操作
S216
结束

分布式限流优化算法

算法说明	
时间	分布式环境获取当前系统时间由外部引入。可以统一从单点系统获取或者从缓存获取（如果缓存也是分布式的，可以基于Hash将对应请求Key映射到指定的缓存实例上）
令牌添加	采用触发式添加令牌的方式，每次获取令牌之前被动计算对应时间段内所需添加的令牌数
所需令牌数计算	need_permits(i) = ⌊S(i) * V(i) * U⌋ S(i) = c(i)/ ∑ c(k) S(i) : 表示对应请求i的平均耗时比率 V(i): 表示对应请求i的令牌桶添加速度 U : 表示调节因子（与对应key优先级成正比, 默认1）
可预消费令牌数计算	Pre_count(i) = ⌊need_permits(i) * G(i)⌋

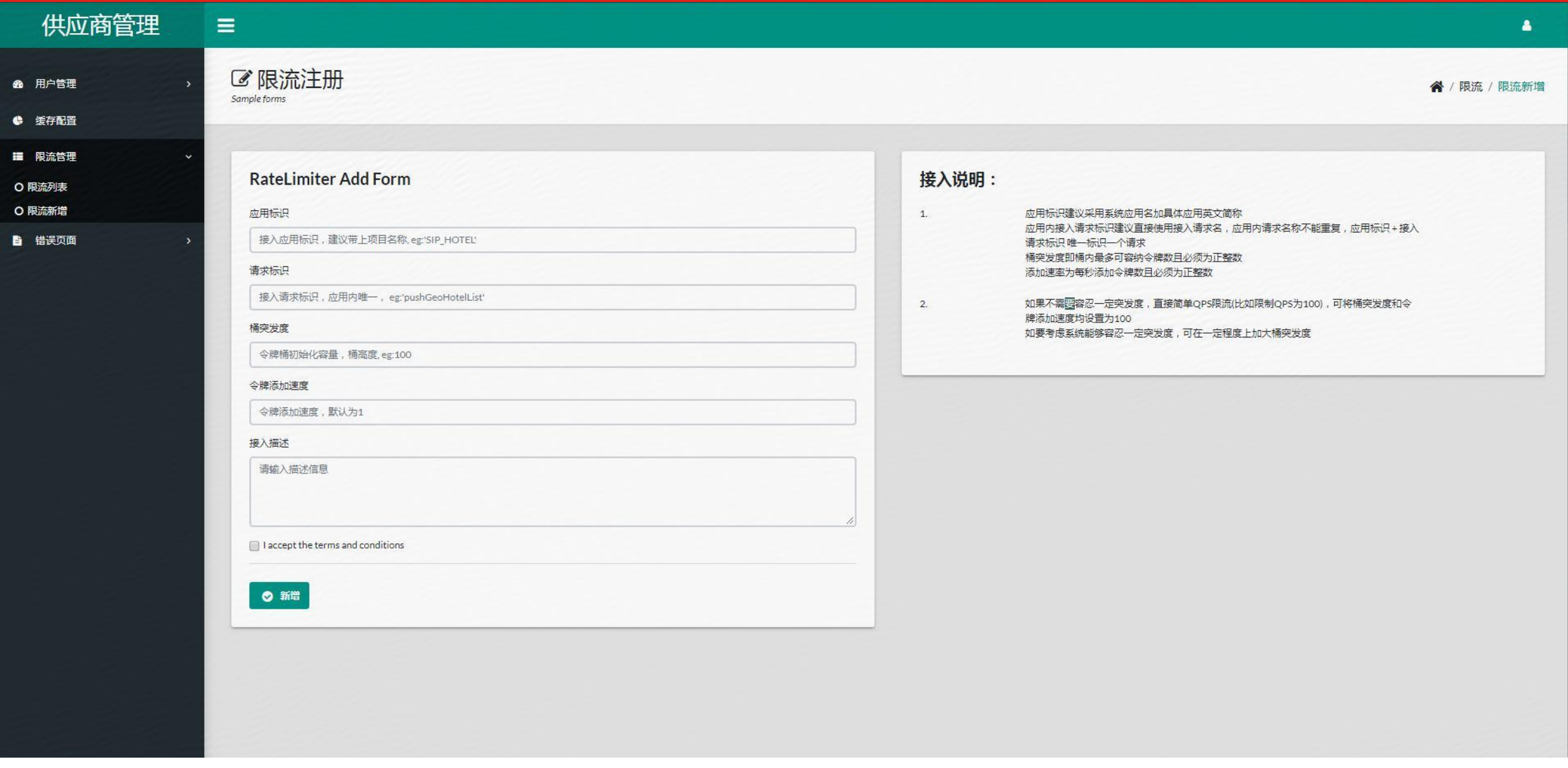
京东

供应商限流系统设计与实现

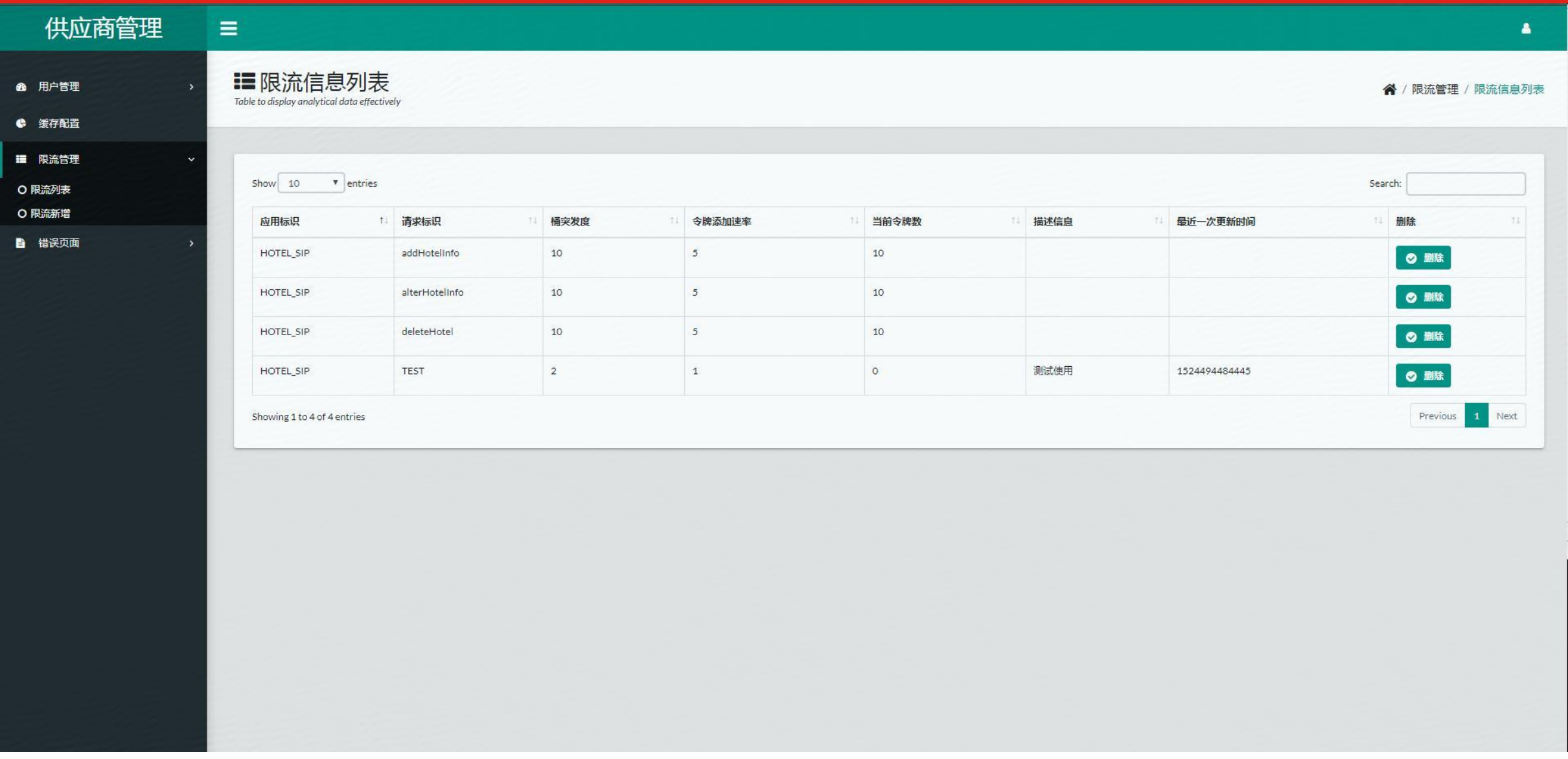


京东

供应商限流系统设计与实现



供应商限流系统设计与实现



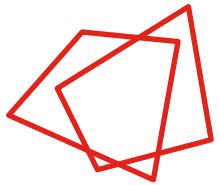
供应商限流系统设计与实现

```java
@RequestMapping(value = "/test", method = RequestMethod.GET)
@RateLimiter(app = "HOTEL_SIP", name = "TEST", needPermits = "1", desc = "测试获取令牌")
public void testAspect(@RequestParam String id, HttpServletResponse response) throws Exception {
    //Poems For you
    GLogger.info("执行业务方法");
    response.getWriter().write("SUCCESS");
}
```

3. 触发熔断返回模板(目前主要提供供应商接入限流熔断返回模板，其他待开发)：

```
{
  "result": "FAILURE",
  "errorMessage": {
          "code": 2000,
          "desc": "请求过于频繁，请稍后重试"
  }
}
```

京东



未来规划

* 熔断可定制化
  - 引入Hystrix(豪猪), 应用介绍：
    https://tech.meituan.com/service-fault-tolerant-pattern.html

* 初始化过程进一步简化

* 系统性能监测轻量化实现

京东

感谢聆听

THANKS

京东