

 **FearlessVoyager**

关注

原创

FearlessVoyager

已于 2024-07-16 16:10:35 修改

阅读量555

收藏 12

点赞数 11

分类专栏: Java

文章标签: java 算法 开发语言



Java 专栏收录该内容

1 订阅 20 篇文章

1、限流算法简介

在 **分布式系统** 中，高并发场景下，为了防止系统因突然的流量激增而导致的崩溃，同时保证服务的高可用性和稳定性，限流是最常用的手段。常见主要有：固定窗口算法、滑动窗口算法、漏桶算法、令牌桶算法。

2、固定窗口算法

固定窗口限流算法，也叫计数器限流算法，是最简单的一种限流算法。主要原理是：通过设定一个固定大小的时间窗口，在该时间窗口内允许通过的每个请求进来都会在计数器上加一，当计数器达到设定的阈值时，请求会被拒绝。当超过设置的时间，计数器会重置为0。

```
1 public class FixWindowLimiter {
2     /**
3      * 每秒限制请求数
4      */
5     private final long perSecondLimit = 50;
6     /**
7      * 上一个窗口的开始时间
8      */
9     public long preStartTime = System.currentTimeMillis();
10    /**
11     * 计数器
12     */
13    private int counter;
14
15    public synchronized boolean tryAcquire() {
16        long now = System.currentTimeMillis();
17        // 假设窗口时间位1秒，在窗口期内判断计数器是否超过限制的请求数
18        if (now - preStartTime < 1000) {
19            // 计数器小于限制数时放行，否则拒绝请求
20            if (counter < perSecondLimit) {
21                counter++;
22                return true;
23            } else {
24                return false;
25            }
26        }
27        // 时间窗口过期，重置计数器和时间戳
28        counter = 0;
29        preStartTime = now;
30        return true;
31    }
32 }
```

优点：

- 实现简单，容易理解。
- 适用于突发流量较小的场景。

缺点：

- 无法处理时间窗口的临界突变问题。
- 对于高并发场景，难以保证系统稳定性。
- 无法实现更加精细的限流控制。

3、滑动窗口算法



FearlessVoyager

关注



```
public class SlidingWindowLimiter {
    // 固定时间窗口大小, 单位毫秒
    private long windowSize;
    // 固定窗口拆分的小窗口数
    private int windowNum;
    // 每个窗口允许通过最大请求数
    private int maxRequestCount;
    // 各个窗口内请求计数
    private int[] perWindowCount;
    // 请求总数
    private int totalCount;
    // 当前窗口下标
    private int windowId;
    // 每个小窗口大小, 毫秒
    private long perWindowSize;
    // 窗口右边界
    private long windowRightBorder;

    /**
     * 构造函数
     *
     * @param windowSize 固定时间窗口大小
     * @param windowNum 固定窗口拆分的小窗口数
     * @param maxRequestCount 每个窗口允许通过最大请求数
     */
    public SlidingWindowLimiter(long windowSize, int windowNum, int maxRequestCount) {
        this.windowSize = windowSize;
        this.windowNum = windowNum;
        this.maxRequestCount = maxRequestCount;
        perWindowCount = new int[windowNum];
        perWindowSize = windowSize / windowNum;
        windowRightBorder = System.currentTimeMillis();
    }

    /**
     * 限流方法
     * @return
     */
    public synchronized boolean tryAcquire() {
        long currentTime = System.currentTimeMillis();
        if (currentTime > windowRightBorder){
            do {
                windowId = (++windowId) % windowNum;
                totalCount -= perWindowCount[windowId];
                perWindowCount[windowId]=0;
                windowRightBorder += perWindowSize;
            }while (windowRightBorder < currentTime);
        }
        if (totalCount < maxRequestCount){
            System.out.println("tryAcquire success,{}",windowId);
            perWindowCount[windowId]++;
            totalCount++;
            return true;
        }else{
            System.out.println("tryAcquire fail,{}",windowId);
            return false;
        }
    }

    /**
     * 测试方法
     * @param args
     * @throws InterruptedException
     */
    public static void main(String[] args) throws InterruptedException {
        SlidingWindowLimiter slidingWindowLimiter = new SlidingWindowLimiter(1000,
            TimeUnit.MILLISECONDS.sleep(800);
    }
}
```





```
69     for (int i = 0; i < 15; i++) {
70         boolean acquire = slidingWindowLimiter.tryAcquire();
71         if (acquire){
72             System.out.println("执行任务");
73         }else{
74             System.out.println("被限流");
75         }
76         TimeUnit.MILLISECONDS.sleep(10);
77     }
78 }
```

优点:

- 可以根据业务需求灵活调整窗口的大小和时间间隔, 实现更加精细的限流控制。
- 解决了固定窗口算法的窗口边界问题, 避免突发流量压垮服务器。

缺点:

- 窗口的大小和时间间隔需要根据具体业务场景进行调整, 实现较为复杂。
- 需要统计窗口内的请求次数, 计算较为复杂。

#### 4. 漏桶算法

漏桶算法是一种基于固定速率的流量控制算法。在漏桶算法中, 请求像水一样不断地注入漏桶, 而漏桶会按照固定的速率将水漏掉。如果注入的速率>漏出的速率, 则会出现限流的效果。漏桶算法可以限制请求的速率, 并且可以防止出现过载的情况。如果入口流量过大, 漏桶可能会溢出, 导致数据丢失。

```
1 public class LeakyBucketLimiter {
2     /**
3      * 桶的最大容量
4      */
5     public long capacity = 10;
6     /**
7      * 桶内当前水量
8      */
9     public long count = 0;
10    /**
11     * 漏水速率（每秒5次）
12     */
13    public long rate = 5;
14    /**
15     * 上次漏水时间
16     */
17    public static long lastLeakTime = System.currentTimeMillis();
18
19    /**
20     * 限流方法, 返回true表示通过
21     */
22    public synchronized boolean tryAcquire() {
23        // 调用漏水方法
24        this.leak();
25        // 判断是否超过最大请求数量
26        if (count < capacity) {
27            count++;
28            return true;
29        }
30        return false;
31    }
32
33    /**
34     * 漏水方法, 计算并更新这段时间内漏水量
35     */
36    private void leak() {
37        // 获取系统当前时间
38        long currentTime = System.currentTimeMillis();
39        // 计算这段时间内, 需要流出的水量
40        long leakWater = (currentTime - lastLeakTime) * rate / 1000;
```





```
43     }  
44 }
```

优点:

- 可以限制请求的速率，并且不会出现过载的情况。
- 可以实现较为精细的限流控制。

缺点:

- 如果入口流量过大，超过了桶的容量，那么就需要丢弃部分请求。
- 由于速率是固定的，即使下游能够处理更大的流量，漏桶也不允许突发流量通过。

## 5、令牌桶算法

令牌桶算法是一种基于令牌的流量控制算法。在令牌桶算法中，系统会向令牌桶中不断添加令牌，每个请求会消耗掉一个令牌，如果令牌桶中没有足够的令牌，则请求会被拒绝。令牌桶算法可以限制请求的速率，同时不会出现过载的情况。

```
1 public class TokenBucketRateLimiter {  
2     /**  
3      * 桶的最大容量  
4      */  
5     public long capacity = 10;  
6     /**  
7      * 桶内当前的令牌数量  
8      */  
9     public long count = 0;  
10    /**  
11     * 令牌生成速率（每秒5次）  
12     */  
13    public long tokenRate = 5;  
14    /**  
15     * 上次生成令牌的时间  
16     */  
17    public long lastGenerateTime = System.currentTimeMillis();  
18  
19    /**  
20     * 限流方法，返回true表示通过  
21     */  
22    public boolean limit() {  
23        // 调用生成令牌方法  
24        this.generateTokens();  
25        // 判断桶内是否还有令牌  
26        if (count > 0) {  
27            count--;  
28            return true;  
29        }  
30        return false;  
31    }  
32  
33    /**  
34     * 生成令牌方法，计算并更新这段时间内生成的令牌数量  
35     */  
36    private void generateTokens() {  
37        long currentTime = System.currentTimeMillis();  
38        // 计算这段时间内，需要生成的令牌数量  
39        long tokens = (currentTime - lastGenerateTime) * tokenRate / 1000;  
40        count = Math.min(count + tokens, capacity);  
41        lastGenerateTime = currentTime;  
42    }  
43 }
```

优点:

- 令牌桶算法可以处理突发流量。当桶满时，能够以最大速度处理请求。



FearlessVoyager

关注



- 与漏桶算法相比，令牌桶算法提供了更大的灵活性。

缺点:

- 需要维护令牌桶和令牌生成速度等状态信息，实现较为复杂。
- 当令牌桶溢出时，会导致请求被拒绝，影响用户体验。

文章知识点与官方知识档案匹配，可进一步学习相关知识

算法技能树 首页 概览 63056 人正在系统学习中

实战系列-Java实现限流算法

初级

导语 在Java开发过程中，经常用到的限流算法有两种，一种是令牌桶算法，一种是漏水桶算法，那么下面就来分别看看两种算法的Java实现方式。 文章目录限流算法

常用限流算法的Java实现

Huangjiazhen711的

主要内容为三种限流算法的Java实现获取连接许可的接口。

Java常见限流用法介绍和实现\_java 限流

随着时间的变化,小窗口随之平移,并且重置/舍弃过期的小窗口,每个小窗口的计数器相加,不超过大窗口的限流limit,即限流阈值之内。(2)实现 import java.util.Arrays; import java

Java常见限流方式 java 限流

1、计数限流 2、固定窗口限流 3、滑动窗口限流 4、漏桶算法 5、令牌桶算法 1、计数限流 例如系统能同时处理 100 个请求,保存一个计数器,处理了一个请求,计数器就加一

java的限流算法

qq\_45062447的

\*\* Java限流算法\*\* 概要 在大数据量高并发访问时，经常会出现服务或接口面对暴涨的请求而不可用的情况，甚至引发连锁反映导致整个系统崩溃。此时你需要使用的技术

【Java限流算法详解及实现】 最新发布

Hhzyy99的

令牌桶算法是一个常用于网络流量整形和流量控制的算法。它的工作原理是系统以固定的速率生成令牌，并将其放入令牌桶中。当请求到来时，需要从桶中取出一定数量的

十分钟搞懂Java限流及常见方案 java 限流

白名单就更好理解了,相当于御赐金牌在身,可以自由穿梭在各种限流规则里,畅行无阻。比如某些电商公司会将超大卖家的账号加入白名单,因为这类卖家往往有自己的一套

Java - 深入四大限流算法:原理、实现与应用\_java流控算法

限流算法是一种在分布式系统中广泛使用的技术,用于控制对系统资源的访问速率,以保护系统免受恶意攻击或突发流量导致的过载。 在实际的业务场景中,接口限流策略的

java 滑动窗口算法\_Java 实现滑动时间窗口限流算法的代码

weixin\_34782968的

在网上搜滑动时间窗口限流算法，大都太复杂了，本人实现了个简单的，先上代码： package cn.dijia478.util;import java.time.LocalDateTime;import java.util.LinkedList;impc

java中常见的限流算法详细解析

码农研究僧的

突发的流量请求，系统可能会造成奔溃。可以通过集群多个服务器，所以要加以限流 生活中的比如秒杀订单或者微博热搜条等 在某种容器或者验证上也可以加以限流，具

【Java】四种方案实现限流\_java 限流方案

【Java】四种方案实现限流 本文介绍了四种限流算法:固定窗口限流、滑动窗口限流(基于RedisZSet)、漏桶限流和令牌桶限流(如Guava实现),并分析了各自的适用场景。『

java限流方案 java 限流实现

java限流方案 在Java中,实现限流有多种方式,以下是其中一些常见的方案: 1、计数器限流:使用一个计数器记录请求的数量,当达到设定的阈值时拒绝后续请求,可以通过Atomic

Java限流实现

NULL 博文链接：https://bijian1013.iteye.com/blog/2382409

基于令牌桶算法的Java限流实现

常见的限流算法有多种，如漏桶算法、令牌桶算法、滑动窗口算法等。在Java中，我们可以使用令牌桶算法来实现限流机制。 基于令牌桶算法的Java限流实现是指使用令

基于加权计数器限流算法的java计算限流工具

与市面上开源的限流工具（如谷歌的RateLimiter令牌桶限流、京东HotKey的滑动窗口限流，更关注流量突发缓解，但是过去流量占用资源是否释放不被关注）不同点在于

Java常见100多种算法大全

这份Java算法大全不仅涵盖了基础算法，还可能包含一些高级和特定场景的算法，例如网络流、最优化问题、组合优化等。通过学习和实践这些算法，开发者可以提升解

java实现令牌桶限流

限流是对某一时间窗口内的请求数进行限制，保持...常用的限流算法有令牌桶和漏桶，而Google开源项目Guava中的RateLimiter使用的就是令牌桶控制算法。 在开发高

JAVA近百种算法大全

Java算法大全是一个包含约100种常见算法的资源库，专为Java程序员设计，用于深入理解和实践编程中的各种算法。这些算法涵盖了数据结构、排序、搜索、图论等多

常见的几种限流算法代码实现（JAVA）

weixin\_44991304的

限流算法

一文详解 Java 限流常见的四种限流算法



FearlessVoyager

关注



Java中常用的限流算法

znadongcnahoyi

在Java的系统中，在一些活动日或者是被黑客攻击，导致访问量突然暴增，系统承受不了巨大的流量冲击而崩溃。为了保护我们的系统，在实际开发中有四种常见的限流

java限流

tiny的

```
public class ConcurrencyControl { private static ConcurrencyControl instance = new ConcurrencyControl(); public static ConcurrencyControl getInstance(){ return instanc
```

JAVA限流

qq\_34253002的

1) 限流的方式 漏桶 (leaky bucket) 漏桶则是按照常量固定速率流出请求，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；漏桶限制的

令牌桶限流算法java

令牌桶限流算法是一种常用的流量控制算法，可以用于限制系统的接口请求频率。在Java中，可以使用以下方式实现令牌桶限流算法： ``java public class TokenBucket {

关于我们 招贤纳士 商务合作 寻求报道 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00

公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心 家长监护 网络110报警服务 中国互联网举报中心 Chrome商店下载 账号管理规范 版权与免责声明 版权申诉 出版物许可证 营业执照 ©1999-2024北京创新乐知网络技术有限公司



FearlessVoyager

码龄9年 暂无认证

132 2万+ 1万+ 19万+ 等级  
原创 周排名 总排名 访问

2460 786 1093 25 1604  
积分 粉丝 获赞 评论 收藏



私信

关注

AI圈早知道，每日最新动态  
了解全球AI新鲜事！  
立即参与

大额流量券免费送  
发布一篇就可获得！  
去查看

搜博主文章



热门文章

MyBatis-plus 分页查询 13546

Linux文件搜索命令 12656

Netty框架详解 8108

修改Docker容器中MySQL的用户密码 7718

Maven--settings.xml配置详解 7686

分类专栏

spring cloud	9篇
docker	17篇
Java	20篇
SpringBoot	7篇



FearlessVoyager

关注



最新评论

- Spring--@Transactional解析  
qq\_39159381: nested的场景二，事务是不  
是会失效
- Docker搭建nacos集群  
m0\_56088616: 配置nginx反向代理的那个  
文章链接无效
- Netty框架详解  
2401\_84103549: 这篇文章真是一篇佳作!作  
者运用了生动有趣的语言,将枯燥的理论: ...
- Netty框架详解  
普通网友: 支持一下，细节很到位!【我也  
写了一些相关领域的文章，希望能够得 ...
- Netty框架详解  
普通网友: 支持一下！我也写了一篇获取  
【大厂面试题解析、核心开发学习笔 ...

最新文章

- Resilience4j 实现接口限流
- Docker Compose 使用
- Jenkins--自动化构建和部署SpringBoot项目

- 2024年 38篇      2023年 37篇
- 2022年 27篇      2021年 20篇
- 2020年 10篇

 恒创科技

香港云服务器

Ping <10ms

- 双向CN2
- 快速稳定
- 免备案

查看详情

目录

- 1、限流算法简介
- 2、固定窗口算法



FearlessVoyager

关注





5、令牌桶算法



FearlessVoyager

关注