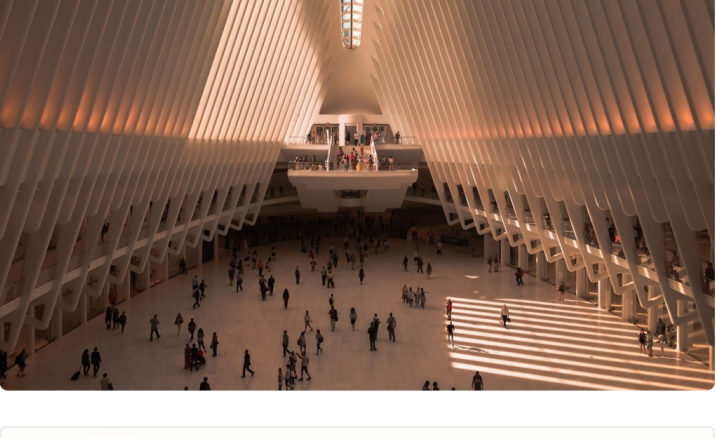


# 91 | 项目实战一：设计实现一个支持各种算法的限流框架（设计）

文章 2020-06-01



00:00

1.0x

讲述：冯永吉

大小：11.20M

时长：12:13

上一节课，我们介绍了限流框架产生的项目背景，并且对需求做了分析，这其中包括功能性需求和非功能性需求，算是在正式开始设计之前的一个铺垫。

前面提到，我们把项目实战分为分析、设计、实现三部分来讲解。其中，分析环节跟之前讲过的面向对象分析很相似，都是做需求的梳理。但是，项目实战中的设计和实现，跟面向对象设计和实现就不是一回事儿了。这里的“设计”指的是系统设计，主要是划分模块，对模块进行设计。这里的“实现”实际上等于面向对象设计加实现。因为我们前面讲到，面向对象设计与实现是聚焦在代码层面的，主要产出的是类的设计和实现。

今天，我们分限流规则、限流算法、限流模式、集成使用这 4 个模块，来讲解限流框架的设计思路。上节课我们提到，限流框架的基本功能非常简单，复杂在于它的非功能性需求，所以，我们今天讲解的重点是，看如何通过合理的设计，实现一个满足易用、易扩展、灵活、低延时、高容错等非功能性需求的限流框架。

话不多说，让我们正式开始今天的学习吧！

## 限流规则

框架需要定义限流规则的语法规式，包括调用方、接口、限流阈值、时间粒度这几个元素。框架用户按照这个语法规式来配置限流规则。我举了一个例子来说明一下，如下所示。其中，unit 表示限流时间粒度，默认情况下是 1 秒。limit 表示在 unit 时间粒度内最大允许的请求次数。拿第一条规则来举例，它表示的意思就是：调用方 app-1 对接口 /v1/user 每分钟的最大请求次数不能超过 100 次。

复制代码

```
1 configs:
2   - appId: app-1
3     limits:
4       - api: /v1/user
5         limit: 100
6         unit: 60
7       - api: /v1/order
8         limit: 50
9   - appId: app-2
10    limits:
11      - api: /v1/user
12        limit: 50
13      - api: /v1/order
14        limit: 50
```

对于限流时间粒度的选择，我们既可以选择限制 1 秒钟内不超过 1000 次，也可以选择限制 10 毫秒内不超过 10 次，还可以选择限制 1 分钟内不超过 6 万次。虽然看起来这几种限流规则是等价的，但过大的时间粒度会达不到限流的效果。比如，有可能 6 万次请求集中在 1 秒中到达，限制 1 分钟不超过 6 万次，就起不到保护的作用；相反，因为接口访问在细时间粒度上随机性很大，并不会很均匀。过小的时间粒度，会误杀很多本不应该限流的请求。所以，尽管越细的时间粒度限流整形效果越好，流量曲线越平滑，但也并不是时间粒度越小越合适。

我们知道，Spring 框架支持各种格式的配置文件，比如 XML、YAML、Porperties 等。除此之外，基于约定优于配置原则，Spring 框架用户只需要将配置文件按照约定来命名，并且放置到约定的路径下，Spring 框架就能按照约定自动查找和加载配置文件。

大部分 Java 程序员已经习惯了 Spring 的配置方式，基于我们前面讲的最小惊奇原则，在限流框架中，我们也延续 Spring 的配置方式，支持 XML、YAML、Properties 等几种配置文件格式，同时，约定默认的配置文件名名为 ratelimiter-rule.yaml，默认放置在 classpath 路径中。

除此之外，为了提高框架的兼容性、易用性，除了刚刚讲的本地文件的配置方式之外，我们还希望兼容从其他数据源获取配置的方式，比如 Zookeeper 或者自研的配置中心。

## 限流算法

常见的限流算法有：固定时间窗口限流算法、滑动时间窗口限流算法、令牌桶限流算法、漏桶限流算法。其中，固定时间窗口限流算法最简单。我们只需要选定一个起始时间起点，之后每来一个接口请求，我们都给计数器（记录当前时间窗口内的访问次数）加一，如果在当前时间窗口内，根据限流规则（比如每秒钟最大允许 100 次接口请求），累加访问次数超过限流值（比如 100 次），就触发限流熔断，拒绝接口请求。当进入下一个时间窗口之后，计数器清零重新计数。

不过，固定时间窗口的限流算法的缺点也很明显。这种算法的限流策略过于粗略，无法应对两个时间窗口临界时间内的突发流量。我们来举一个例子。假设我们限流规则为每秒钟不超过 100 次接口请求。第一个 1 秒时间窗口内，100 次接口请求都集中在最后的 10 毫秒内，在第二个 1 秒时间窗口内，100 次接口请求都集中在最开始的 10 毫秒内。虽然两个时间窗口内流量都符合限流要求（小于等于 100 个接口请求），但在两个时间窗口临界的 20 毫秒内集中有 200 次接口请求，固定时间窗口限流算法没法对这种情况进行限流，集中在这 20 毫秒内的 200 次请求有可能会压垮系统。

为了让流量更加平滑，于是就有了更加高级的滑动时间窗口限流算法、令牌桶限流算法和漏桶限流算法。因为我们主要讲设计而非技术，所以其他几种限流算法，留给你自己去研究，你也可以参看我之前写的关于限流框架的技术文档。

尽管固定时间窗口限流算法没法做到让流量很平滑，但大部分情况下，它已经够用了。默认情况下，框架使用固定时间窗口限流算法做限流。不过，考虑到框架的扩展性，我们需要预先做好设计，预留好扩展点，方便今后扩展其他限流算法。除此之外，为了提高框架的易用性、灵活性，我们最好将其他几种常用的限流算法，也在框架中实现出来，供框架用户根据自己业务场景自由选择。

## 限流模式

刚刚讲的是限流算法，我们再讲讲限流模式。我们把限流模式分为两种：单机限流和分布式限流。所谓单机限流，就是针对单个实例的访问频率进行限制。注意这里的单机并不是真的一台物理机器，而是一个服务实例，因为有可能一台物理机器部署多个实例。所谓的分布式限流，就是针对某个服务的多个实例的总的访问频率进行限制。我举个例子来解释一下。

假设我们开发了一个用户相关的微服务，为了提高服务能力，我们部署了 5 个实例。我们限制某个调用方，对单个实例的某个接口的访问频率，不能超过 100 次 / 秒。这就是单机限流。我们限制某个调用方，对 5 个实例的某个接口的总访问频率，不能超过 500 次 / 秒。这就是所谓的分布式限流。

从实现的角度来分析，单机限流和分布式限流的主要区别在接口访问计数器的实现。单机限流只需要在单个实例中维护自己的接口请求计数器。而分布式限流需要集中管理计数器（比如使用 Redis 存储接口访问计数），这样才能做到多个实例对同一个计数器累加计数，以便实现对多个实例总访问频率的限制。

前面我们讲到框架要高容错，不能因为框架的异常，影响到集成框架的应用的可用性和稳定性。除此之外，我们还讲到框架要低延迟。限流逻辑的执行不能占用太长时间，不能或者很少影响接口请求本身的响应时间。因为分布式限流基于外部存储 Redis，网络通信成本较高，实际上，高容错、低延迟设计的主要场景就是基于 Redis 实现的分布式限流。

对于 Redis 的各种异常情况，我们处理起来并不难，捕获并封装为统一的异常，向上抛出或者吞掉就可以了。比较难处理的是 Redis 访问超时。Redis 访问超时会严重影响接口的响应时间，甚至导致接口请求超时。所以，在访问 Redis 时，我们需要设置合理的超时时间。一旦超时，我们就判定为限流失效，继续执行接口请求。Redis 访问超时时间的设置既不能太大也不能太小，太大可能会影响到接口的响应时间，太小可能会导致太多的限流失效。我们可以通过压测或者线上监控，获取到 Redis 访问时间分布情况，再结合接口可以容忍的限流延迟时间，权衡设置一个较合理的 Redis 超时时间。

## 集成使用

前面剖析 Spring 框架的时候，我们讲到低侵入松耦合设计思想。限流框架也应该满足这个设计思想。因为框架是需要集成到应用中使用的，我们希望框架尽可能低侵入，与业务代码松耦合，替换、删除起来也更容易些。

除此之外，在剖析 MyBatis 框架的时候，我们讲到 MyBatis 框架是为了简化数据库编程。实际上，为了进一步简化开发，MyBatis 还提供了 MyBatis-Spring 类库，方便在使用了 Spring 框架的项目中集成 MyBatis 框架。我们也可以借鉴 MyBatis-Spring，开发一个 Ratelimiter-Spring 类库，能够方便使用了 Spring 的项目集成限流框架，将易用性做到极致。

## 重点回顾

好了，今天的内容到此就讲完了。我们一块来总结回顾一下，你需要重点掌握的内容。

我们将这个限流框架划分为限流规则、限流算法、限流模式、集成使用者这四个模块来分析讲解。除了功能方面的设计之外，我们重点讲了如何满足易用、灵活、易扩展、低延迟、高容错这些非功能性需求。

针对限流规则，大部分 Java 程序员已经习惯了 Spring 的配置方式。基于最小惊奇原则，在限流框架中，我们也延续 Spring 的配置方式，支持 XML、YAML、Properties 等几种配置文件格式。同时，借鉴 Spring 的约定优于配置设计原则，限流框架用户只需要将配置文件按照约定来命名，并且放置到约定的路径下，框架就能按照约定自动查找和加载配置文件。除此之外，为了提高框架的兼容性、易用性，除了本地文件的配置方式之外，我们还希望兼容从其他数据源获取配置的方式，比如 Zookeeper 或者自研的配置中心。

针对限流算法，尽管固定时间窗口限流算法没法做到让流量很平滑，但大部分情况下，它已经够用了。默认情况下，框架使用固定时间窗口限流算法做限流。不过，考虑到框架的扩展性，我们需要预先做好设计，预留好扩展点，方便今后扩展其他限流算法。除此之外，为了提高框架的易用性、灵活性，我们将其他几种常用的限流算法也在框架中实现出来，供框架用户根据自己的业务场景自由选择。

针对限流模式，因为分布式限流基于外部存储 Redis，网络通信成本较高，框架的高容错和低延迟的设计，主要是针对基于 Redis 的分布式限流模式。不能因为 Redis 的异常，影响到集成框架的应用的可用性和稳定性。不能因为 Redis 访问超时，导致接口访问超时。

针对集成使用，我们希望框架低侵入，跟业务代码松耦合。应用集成框架的代码，尽可能集中、不分散，这样删除、替换起来就容易很多。除此之外，为了将框架的易用性做到极致，我们借鉴 MyBatis-Spring 类库，设计实现一个 RateLimiter-Spring 类库，方便集成了 Spring 框架的应用集成限流框架。

## 课堂讨论

今天，我们提到配置限流规则的时候，时间粒度不能太大，也不能太小，限流值也要设置得合理，太大起不到限流的作用，太小容易误杀。那请你思考一下，如何选择合理的时间粒度和限流值？如何验证设置的合理性？

欢迎留言和我分享你的想法。如果有收获，也欢迎你把这篇文章分享给你的朋友。

更多福利推荐

充值限时膨胀

充 ¥500 得 ¥580

下单即赠精选爆款商品

戳此充值

充 ¥500 得 ¥580

充 ¥1000 得 ¥1200

充 ¥2000 得 ¥2500

充电宝

蓝牙耳机

智能手表

运动相机

无线耳机

智能音箱

无线充电器

无线鼠标

无线键盘

无线耳机

无线充电器

无线鼠标

无线键盘

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

志恒Z

1

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100