90 | 项目实战一：设计实现一个支持各种算法的限流框架（分析）

王争 2020-05-29

结束了开源实战，从今天开始我们正式进入项目实战模块。在开源实战中，我带你一块剖析了几个著名的开源项目，比如 Spring、MyBatis、Google Guava 等，剖析了它们背后蕴含的设计思想、原则和模式。

如果说前面讲开源实战是学习别人怎么做，那现在我们讲项目实战就是带你一块做。在这个过程中，我会带你实践之前学过的设计思想、原则和模式，给你展示怎么应用这些理论知识，让你开发出跟前面那些著名开源项目一样优秀的软件。

在项目实战中，我找了三个稍微有点难度的项目：限流框架、幂等框架、灰度发布组件，带你一起来实现。针对每一个项目，我都会从分析、设计、实现这三个部分来讲解。当然，还是那句老话，项目本身的讲解不是重点，重点还是学习它们背后的开发套路。这才是最有价值的部分。

接下来的三节课，我们讲第一个实战项目，限流框架。今天，我们先讲其中的分析环节，介绍项目背景，分析项目需求。

话不多说，让我们正式开始今天的学习吧！

项目背景

我们先来讲下需求诞生的背景。这个背景跟我们下一个实战项目幂等框架也有关系，所以要从很久很久讲起，希望你能耐心看完，不然后面可能会看不懂。

公司成立初期，团队人少。公司集中精力开发一个金融理财产品（我们把这个项目叫做 X 项目）。整个项目只做了简单的前后端分离，后端的所有代码都在一个 GitHub 仓库中，整个后端作为一个应用来部署，没有划分微服务。

遇到了行业风口，公司发展得不错，公司开始招更多人，开发更多的金融产品，比如专注房贷的理财产品、专注供应链的产品、专注消费贷的借款端产品等等。在产品形态上，每个金融产品都做成了独立的 App。

对于不同的金融产品，尽管移动端长得不一样，但是后端的很多功能、代码都是可以复用的。为了快速上线，针对每个应用，公司都成立一个新的团队，然后拷贝 X 项目的代码，在此基础之上修改、添加新的功能。

这样成立新团队，拷贝老代码，改改就能上线一个新产品的开发模式，在一开始很受欢迎。产品上线快，也给公司赢得了竞争上的优势。但时间一长，这样的开发模式暴露出来的问题就越来越多了。而且随着公司的发展，公司也过了急速扩张期，人招得太多，公司开始考虑研发效率问题了。

因为所有的项目的代码都是从 X 项目拷贝来的，多个团队同时维护相似的代码，显然是重复劳动，协作起来也非常麻烦。任何团队发现代码的 bug，都要同步到其他团队做相同的修改。而且，各个团队对代码独立迭代，改得面目全非，即便要添加一个通用的功能，每个团队也都要基于自己的代码再重复开发。

除此之外，公司成立初期，各个方面条件有限，只能招到开发水平一般的员工，而且追求快速上线，所以，X 项目的代码质量很差，结构混乱、命名不规范、到处是临时解决方案、埋了很多坑，在烂代码之上不停地堆砌烂代码，时间长了，代码的可读性越来越差、维护成本越来越高，甚至高过了重新开发的成本。

这个时候该怎么办呢？如果让你出出主意，你有什么好的建议吗？

我们可以把公共的功能、代码抽离出来，形成一个独立的项目，部署成一个公共服务平台。所有金融产品的后端还是参照 MVC 三层架构独立开发，不过，它们只实现自己特有的功能，对于一些公共的功能，通过远程调用公共服务平台提供的接口来实现。

这里提到的公共服务平台，有点类似现在比较火的“中台”或“微服务”。不过，为了减少部署、维护多个微服务的成本，我们把所有公共的功能，放到一个项目中开发，放到一个应用中部署。只不过，我们要未雨绸缪，事先按照领域模型，将代码的模块化做好，等到真的有哪个模块的接口调用过于集中，性能出现瓶颈的时候，我们再把它拆分出来，设计成独立的微服务来开发和部署。

经过这样的拆分之后，我们可以指派一个团队，集中维护公共服务平台的代码。开发一个新的金融产品，也只需要更少的人员来参与，因为他们只需要开发、维护产品特有的功能和代码就可以了。整体上，维护成本降低了。除此之外，公共服务平台的代码集中到了一个团队手里，重构起来不需要协调其他团队和项目，也便于我们重构、改善代码质量。

需求背景

对于公共服务平台来说，接口请求来自很多不同的系统（后面统称为调用方），比如各种金融产品的后端系统。在系统上线一段时间里，我们遇到了很多问题。比如，因为调用方代码 bug 、不正确地使用服务（比如启动 Job 来调用接口获取数据）、业务上面的突发流量（比如促销活动），导致来自某个调用方的接口请求数突增，过度争用服务的线程资源，而来自其他调用方的接口请求，因此来不及响应而排队等待，导致接口请求的响应时间大幅增加，甚至出现超时。

为了解决这个问题，你有什么好的建议呢？我先来说说我的。

我们可以开发接口限流功能，限制每个调用方对接口请求的频率。当超过预先设定的访问频率后，我们就触发限流熔断，比如，限制调用方 app-1 对公共服务平台总的接口请求频率不超过 1000 次 / 秒，超过之后的接口请求都会被决绝。除此之外，为了更加精细化地限流，除了限制每个调用方对公共服务平台总的接口请求频率之外，我们还希望能对单独某个接口的访问频率进行限制，比如，限制 app-1 对接口 /user/query 的访问频率为每秒钟不超过 100 次。

我们希望开发出来的东西有一定的影响力，即便做不到在行业内有影响力，起码也要做到在公司范围内有影响力。所以，从一开始，我们就不想把这个限流功能，做成只有我们项目可用。我们希望把它开发成一个通用的框架，能够应用到各个业务系统中，甚至可以集成到微服务治理平台中。实际上，这也体现了业务开发中要具备的抽象意识、框架意识。我们要善于识别出通用的功能模块，将它抽象成通用的框架、组件、类库等。

需求分析

刚刚我们花了很大篇幅来介绍项目背景和需求背景，接下来，我们再对需求进行更加详细的分析和整理。

前面我们已经讲过一些需求分析的方法，比如画线框图、写用户用例、测试驱动开发等等。这里，我们借助用户用例和测试驱动开发的思想，先去思考，如果框架最终被开发出来之后，它会如何被使用。我一般会找一个框架的应用场景，针对这个场景写一个框架使用的 Demo 程序，这样能够很直观地看到框架长什么样子。知道了框架应该长什么样，就相当于应试教育中确定了考试题目。针对明确的考题去想解决方案，这是我们多年应试教育锻炼之后最擅长做的。

对于限流框架来说，我们来看下它的应用场景。

首先我们需要设置限流规则。为了做到在不修改代码的前提下修改规则，我们一般会把规则放到配置文件中（比如 XML、YAML 配置文件）。在集成了限流框架的应用启动的时候，限流框架会将限流规则，按照事先定义的语法，解析并加载到内存中。我写了一个限流规则的 Demo 配置，如下所示：

复制代码

```
configs:
- appId: app-1
  limits:
  - api: /v1/user
    limit: 100
  - api: /v1/order
    limit: 50
- appId: app-2
  limits:
  - api: /v1/user
    limit: 50
  - api: /v1/order
    limit: 50
```

在接收到接口请求之后，应用会将请求发送给限流框架，限流框架会告诉应用，这个接口请求是允许继续处理，还是触发限流熔断。如果我们用代码来将这个过程表示出来的话，就是下面这个 Demo 的样子。如果项目使用的是 Spring 框架，我们可以利用 Spring AOP，把这段限流代码放在统一的切面中，在切面中拦截接口请求，解析出请求对应的调用方 APP ID 和 URL，然后验证是否对此调用方的这个接口请求进行限流。

复制代码

```
String appId = "app-1"; // 调用方APP-ID
String url = "http://www.eudemon.com/v1/user/12345";// 请求url
RateLimiter ratelimiter = new RateLimiter();
boolean passed = ratelimiter.limit(appId, url);
if (passed) {
  // 放行接口请求，继续后续的处理。
} else {
  // 接口请求被限流。
}
```

结合刚刚的 Demo，从使用的角度来说，限流框架主要包含两部分功能：配置限流规则和提供编程接口（RateLimiter 类）验证请求是否被限流。不过，作为通用的框架，除了功能性需求之外，非功能性需求也非常重要，有时候会决定一个框架的成败，比如，框架的易用性、扩展性、灵活性、性能、容错性等。

对于限流框架，我们来看它都有哪些非功能性需求。

易用性方面，我们希望限流规则的配置、编程接口的使用都很简单。我们希望提供各种不同的限流算法，比如基于内存的单机限流算法、基于 Redis 的分布式限流算法，能够让使用者自由选择。除此之外，因为大部分项目都是基于 Spring 开发的，我们还希望限流框架能否非常方便地集成到使用 Spring 框架的项目中。

扩展性、灵活性方面，我们希望能够灵活地扩展各种限流算法。同时，我们还希望支持不同格式（JSON、YAML、XML 等格式）、不同数据源（本地文件配置或 Zookeeper 集中配置等）的限流规则的配置方式。

性能方面，因为每个接口请求都要被检查是否限流，这或多或少会增加接口请求的响应时间。而对于响应时间比较敏感的接口服务来说，我们要让限流框架尽可能低延迟，尽可能减少对接口请求本身响应时间的影响。

容错性方面，接入限流框架是为了提高系统的可用性、稳定性，不能因为限流框架的异常，反过来影响到服务本身的可用性。所以，限流框架要有高度的容错性。比如，分布式限流算法依赖集中存储器 Redis。如果 Redis 挂掉了，限流逻辑无法正常运行，这个时候业务接口也要能正常服务才行。

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。

今天，我们主要对限流框架做了大的项目背景、需求背景介绍，以及更加具体的需求分析，明确了要做什么，为下两节课的设计和实现做准备。

从今天的讲解中，不知道你有没有发现，基本的功能需求其实没有多少，但将非功能性需求考虑进去之后，明显就复杂了很多。还是那句老话，**写出能用的代码很简单，写出好用的代码很难。**对于限流框架来说，非功能性需求是设计与实现的难点。怎么做到易用、灵活、可扩展、低延迟、高容错，才是开发的重点，也是我们接下来两节课要讲解的重点。

除此之外，今天我们还实践了一些需求分析的方法，比如画线框图、写用户用例、测试驱动开发等等。针对限流框架，我们借助用户用例和测试驱动开发的思想，先去思考，如果框架最终被开发出来之后，它会如何被使用。针对具体的场景去做分析，更加清晰直观。

课堂讨论

在今天介绍项目背景的时候，我讲了公司遇到的一个开发问题，并提出了解决方案，你也可以留言分享一下，你所在公司或者项目中，遇到过哪些比较头疼的开发问题，又是如何解决的？

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。