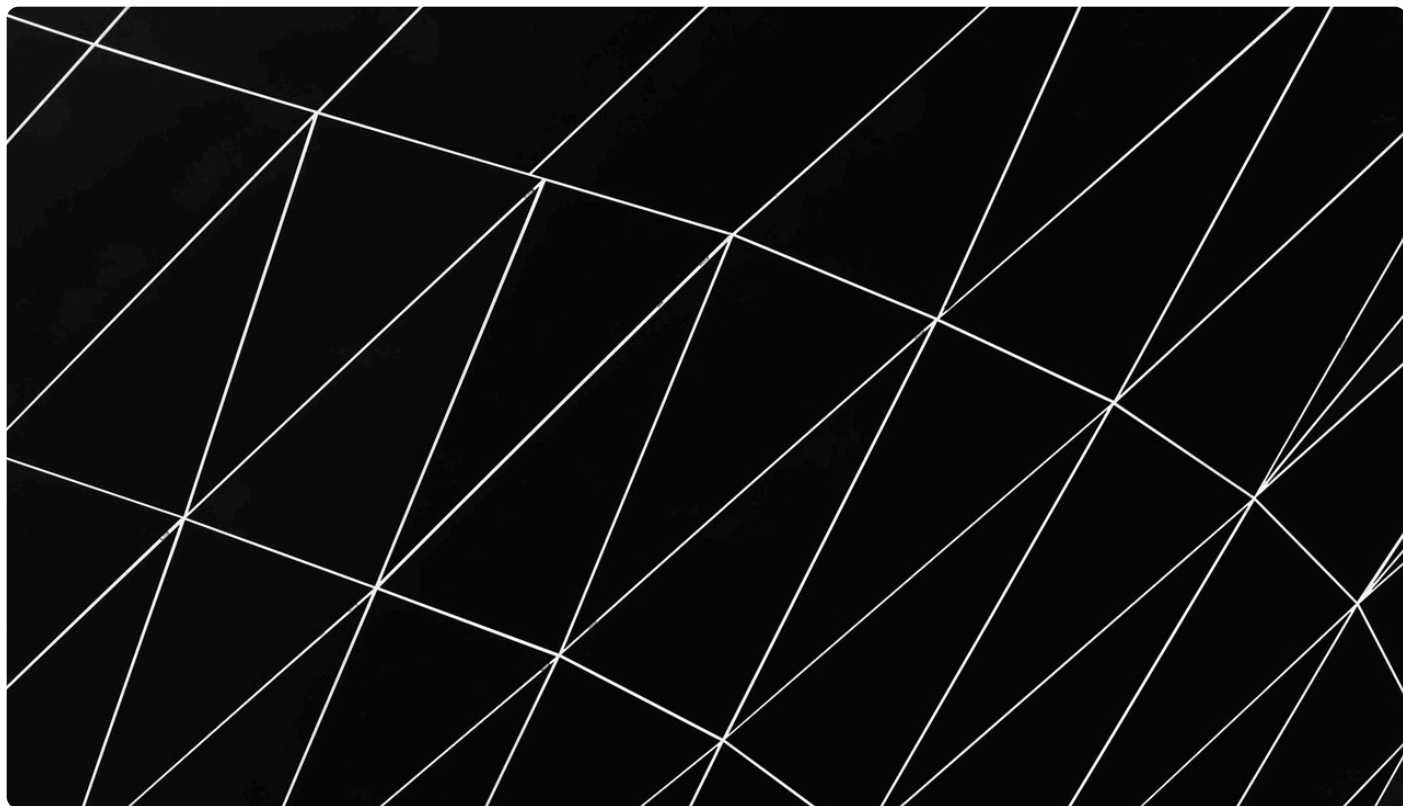


25 | 实战二（上）：针对非业务的通用框架开发，如何做需求分析和设计？

王争 · 设计模式之美



上两节课中，我们讲了如何针对一个业务系统做需求分析、设计和实现，并且通过一个积分兑换系统的开发，实践了之前学过的一些设计原则。接下来的两节课，我们再结合一个支持各种统计规则的性能计数器项目，学习针对一个非业务的通用框架开发，如何来做需求分析、设计和实现，同时学习如何灵活应用各种设计原则。

话不多说，让我们正式开始今天的内容吧！

项目背景

我们希望设计开发一个小的框架，能够获取接口调用的各种统计信息，比如，响应时间的最大值（max）、最小值（min）、平均值（avg）、百分位值（percentile）、接口调用次数（count）、频率（tps）等，并且支持将统计结果以各种显示格式（比如：JSON 格式、网页格式、自定义显示格式等）输出到各种终端（Console 命令行、HTTP 网页、Email、日志文件、自定义输出终端等），以方便查看。

我们假设这是真实项目中的一个开发需求，如果让你来负责开发这样一个通用的框架，应用到各种业务系统中，支持实时计算、查看数据的统计信息，你会如何设计和实现呢？你可以先自己主动思考一下，然后再来看我的分析思路。

需求分析

性能计数器作为一个跟业务无关的功能，我们完全可以把它开发成一个独立的框架或者类库，集成到很多业务系统中。而作为可被复用的框架，除了功能性需求之外，非功能性需求也非常重要。所以，接下来，我们从这两个方面来做需求分析。

1. 功能性需求分析

相对于一大长串的文字描述，人脑更容易理解短的、罗列的比较规整、分门别类的列表信息。显然，刚才那段需求描述不符合这个规律。我们需要把它拆解成一个一个的“干条条”。拆解之后我写在下面了，是不是看起来更加清晰、有条理？

接口统计信息：包括接口响应时间的统计信息，以及接口调用次数的统计信息等。

统计信息的类型：max、min、avg、percentile、count、tps 等。

统计信息显示格式：Json、Html、自定义显示格式。

统计信息显示终端：Console、Email、HTTP 网页、日志、自定义显示终端。

除此之外，我们还可以借助设计产品的时候，经常用到的线框图，把最终数据的显示样式画出来，会更加一目了然。具体的线框图如下所示：

邮件

2019.3.21 00:00:00—2019.3.22 00:00:00

接口	响应时间					请求次数	
	max	min	avg	p99	p999	count	tps

命令行

2019.3.8 12:23:22

```
{
  "api1": {
    "max": 345,
    "min": 23,
    "avg": 124,
    "p99": 265,
    "p999": 309,
    "count": 6789,
    "tps": 101
  },
  "api2": {
    //...
  }
}
```

网页

2019.3.21 19:07:53 ~ 2019.3.26 17:03:51 统计

接口	响应时间					请求次数	
	max	min	avg	p99	p999	count	tps

实际上，从线框图中，我们还能挖掘出了下面几个隐藏的需求。

统计触发方式：包括主动和被动两种。主动表示以一定的频率定时统计数据，并主动推送到显示终端，比如邮件推送。被动表示用户触发统计，比如用户在网页中选择要统计的时间区间，触发统计，并将结果显示给用户。

统计时间区间：框架需要支持自定义统计时间区间，比如统计最近 10 分钟的某接口的 tps、访问次数，或者统计 12 月 11 日 00 点到 12 月 12 日 00 点之间某接口响应时间的最大值、最小值、平均值等。

统计时间间隔：对于主动触发统计，我们还要支持指定统计时间间隔，也就是多久触发一次统计显示。比如，每间隔 10s 统计一次接口信息并显示到命令行中，每间隔 24 小时发送一封统计信息邮件。

2. 非功能性需求分析

对于这样一个通用的框架的开发，我们还需要考虑很多非功能性的需求。具体来讲，我总结了以下几个比较重要的方面。

易用性

易用性听起来更像是一个评判产品的标准。没错，我们在开发这样一个技术框架的时候，也要有产品意识。框架是否易集成、易插拔、跟业务代码是否松耦合、提供的接口是否够灵活等

等，都是我们应该花心思去思考和设计的。有的时候，文档写得好坏甚至都有可能决定一个框架是否受欢迎。

性能

对于需要集成到业务系统的框架来说，我们不希望框架本身的代码执行效率，对业务系统有太多性能上的影响。对于性能计数器这个框架来说，一方面，我们希望它是低延迟的，也就是说，统计代码不影响或很少影响接口本身的响应时间；另一方面，我们希望框架本身对内存的消耗不能太大。

扩展性

这里说的扩展性跟之前讲到的代码的扩展性有点类似，都是指在不修改或尽量少修改代码的情况下添加新的功能。但是这两者也有区别。之前讲到的扩展是从框架代码开发者的角度说的。这里所说的扩展是从框架使用者的角度说的，特指使用者可以在不修改框架源码，甚至不拿到框架源码的情况下，为框架扩展新的功能。这就有点类似给框架开发插件。关于这一点，我举一个例子来解释一下。

feign 是一个 HTTP 客户端框架，我们可以在不修改框架源码的情况下，用如下方式来扩展我们自己的编解码方式、日志、拦截器等。

 复制代码

```
1 Feign feign = Feign.builder()
2     .logger(new CustomizedLogger())
3     .encoder(new FormEncoder(new JacksonEncoder()))
4     .decoder(new JacksonDecoder())
5     .errorDecoder(new ResponseErrorDecoder())
6     .requestInterceptor(new RequestHeadersInterceptor()).build();
7
8 public class RequestHeadersInterceptor implements RequestInterceptor {
9     @Override
10    public void apply(RequestTemplate template) {
11        template.header("appId", "...");
12        template.header("version", "...");
13        template.header("timestamp", "...");
14        template.header("token", "...");
15        template.header("idempotent-token", "...");
16        template.header("sequence-id", "...");
```

```
17 }
18
19 public class CustomizedLogger extends feign.Logger {
20     //...
21 }
22
23 public class ResponseErrorDecoder implements ErrorDecoder {
24     @Override
25     public Exception decode(String methodKey, Response response) {
26         //...
27     }
28 }
```

容错性

容错性这一点也非常重要。对于性能计数器框架来说，不能因为框架本身的异常导致接口请求出错。所以，我们要对框架可能存在的各种异常情况都考虑全面，对外暴露的接口抛出的所有运行时、非运行时异常都进行捕获处理。

通用性

为了提高框架的复用性，能够灵活应用到各种场景中。框架在设计的时候，要尽可能通用。我们要多去思考一下，除了接口统计这样一个需求，还可以适用到其他哪些场景中，比如是否还可以处理其他事件的统计信息，比如 SQL 请求时间的统计信息、业务统计信息（比如支付成功率）等。

框架设计

前面讲了需求分析，现在我们来看如何针对需求做框架设计。

对于稍微复杂系统的开发，很多人觉得不知从何开始。我个人喜欢借鉴 TDD（测试驱动开发）和 Prototype（最小原型）的思想，先聚焦于一个简单的应用场景，基于此设计实现一个简单的原型。尽管这个最小原型系统在功能和非功能特性上都不完善，但它能够看得见、摸得着，比较具体、不抽象，能够很有效地帮助我理清更复杂的设计思路，是迭代设计的基础。

这就好比做算法题目。当我们想要一下子就想出一个最优解法时，可以先写几组测试数据，找找规律，再先想一个最简单的算法去解决它。虽然这个最简单的算法在时间、空间复杂度上可能都不令人满意，但是我们可以基于此来做优化，这样思路就会更加顺畅。

对于性能计数器这个框架的开发来说，我们可以先聚焦于一个非常具体、简单的应用场景，比如统计用户注册、登录这两个接口的响应时间的最大值和平均值、接口调用次数，并且将统计结果以 JSON 的格式输出到命令行中。现在这个需求简单、具体、明确，设计实现起来难度降低了很多。


我们先给出应用场景的代码。具体如下所示：

 复制代码

```
1 //应用场景：统计下面两个接口(注册和登录)的响应时间和访问次数
2 public class UserController {
3     public void register(UserVo user) {
4         //...
5     }
6
7     public UserVo login(String telephone, String password) {
8         //...
9     }
10 }
```

要输出接口的响应时间的最大值、平均值和接口调用次数，我们首先要采集每次接口请求的响应时间，并且存储起来，然后按照某个时间间隔做聚合统计，最后才是将结果输出。在原型系统的代码实现中，我们可以把所有代码都塞到一个类中，暂时不用考虑任何代码质量、线程安全、性能、扩展性等等问题，怎么简单怎么来就行。

最小原型的代码实现如下所示。其中，recordResponseTime() 和 recordTimestamp() 两个函数分别用来记录接口请求的响应时间和访问时间。startRepeatedReport() 函数以指定的频率统计数据并输出结果。

 复制代码

```
1 public class Metrics {
2     // Map的key是接口名称，value对应接口请求的响应时间或时间戳；
```

```

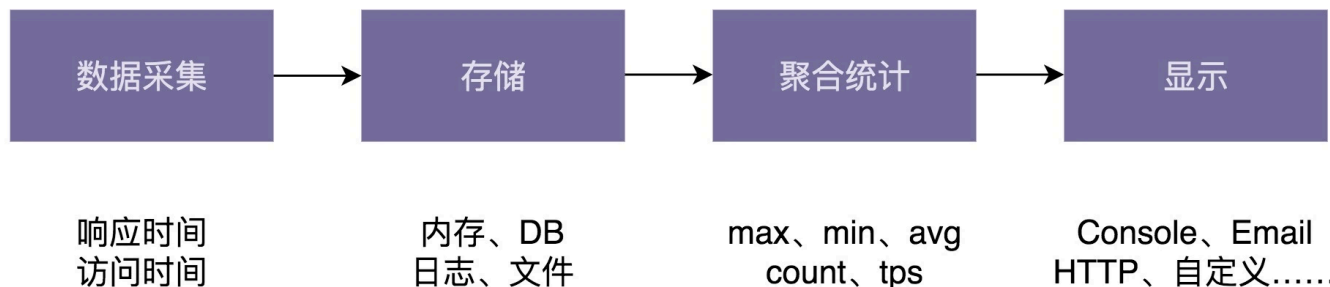
3     private Map<String, List<Double>> responseTimes = new HashMap<>();
4     private Map<String, List<Double>> timestamps = new HashMap<>();
5     private ScheduledExecutorService executor = Executors.newSingleThreadScheduledE
6
7     public void recordResponseTime(String apiName, double responseTime) {
8         responseTimes.putIfAbsent(apiName, new ArrayList<>());
9         responseTimes.get(apiName).add(responseTime);
10    }
11
12    public void recordTimestamp(String apiName, double timestamp) {
13        timestamps.putIfAbsent(apiName, new ArrayList<>());
14        timestamps.get(apiName).add(timestamp);
15    }
16
17    public void startRepeatedReport(long period, TimeUnit unit){
18        executor.scheduleAtFixedRate(new Runnable() {
19            @Override
20            public void run() {
21                Gson gson = new Gson();
22                Map<String, Map<String, Double>> stats = new HashMap<>();
23                for (Map.Entry<String, List<Double>> entry : responseTimes.entrySet()) {
24                    String apiName = entry.getKey();
25                    List<Double> apiRespTimes = entry.getValue();
26                    stats.putIfAbsent(apiName, new HashMap<>());
27                    stats.get(apiName).put("max", max(apiRespTimes));
28                    stats.get(apiName).put("avg", avg(apiRespTimes));
29                }
30
31                for (Map.Entry<String, List<Double>> entry : timestamps.entrySet()) {
32                    String apiName = entry.getKey();
33                    List<Double> apiTimestamps = entry.getValue();
34                    stats.putIfAbsent(apiName, new HashMap<>());
35                    stats.get(apiName).put("count", (double)apiTimestamps.size());
36                }
37                System.out.println(gson.toJson(stats));
38            }
39        }, 0, period, unit);
40    }
41
42    private double max(List<Double> dataset) { //省略代码实现}
43    private double avg(List<Double> dataset) { //省略代码实现}
44 }

```

我们通过不到 50 行代码就实现了最小原型。接下来，我们再来看，如何用它来统计注册、登录接口的响应时间和访问次数。具体的代码如下所示：

```
1 //应用场景：统计下面两个接口(注册和登录) 的响应时间和访问次数
2 public class UserController {
3     private Metrics metrics = new Metrics();
4
5     public UserController() {
6         metrics.startRepeatedReport(60, TimeUnit.SECONDS);
7     }
8
9     public void register(UserVo user) {
10         long startTimestamp = System.currentTimeMillis();
11         metrics.recordTimestamp("register", startTimestamp);
12         //...
13         long respTime = System.currentTimeMillis() - startTimestamp;
14         metrics.recordResponseTime("register", respTime);
15     }
16
17     public UserVo login(String telephone, String password) {
18         long startTimestamp = System.currentTimeMillis();
19         metrics.recordTimestamp("login", startTimestamp);
20         //...
21         long respTime = System.currentTimeMillis() - startTimestamp;
22         metrics.recordResponseTime("login", respTime);
23     }
24 }
```

最小原型的代码实现虽然简陋，但它却帮我们将思路理顺了很多，我们现在就基于它做最终的框架设计。下面是我针对性能计数器框架画的一个粗略的系统设计图。图可以非常直观地体现设计思想，并且能有效地帮助我们释放更多的脑空间，来思考其他细节问题。



如图所示，我们把整个框架分为四个模块：数据采集、存储、聚合统计、显示。每个模块负责的工作简单罗列如下。

数据采集：负责打点采集原始数据，包括记录每次接口请求的响应时间和请求时间。数据采集过程要高度容错，不能影响到接口本身的可用性。除此之外，因为这部分功能是暴露给框架的使用者的，所以在设计数据采集 API 的时候，我们也要尽量考虑其易用性。

存储：负责将采集的原始数据保存下来，以便后面做聚合统计。数据的存储方式有多种，比如：Redis、MySQL、HBase、日志、文件、内存等。数据存储比较耗时，为了尽量地减少对接口性能（比如响应时间）的影响，采集和存储的过程异步完成。

聚合统计：负责将原始数据聚合为统计数据，比如：max、min、avg、percentile、count、tps 等。为了支持更多的聚合统计规则，代码希望尽可能灵活、可扩展。

显示：负责将统计数据以某种格式显示到终端，比如：输出到命令行、邮件、网页、自定义显示终端等。

前面讲到面向对象分析、设计和实现的时候，我们讲到设计阶段最终输出的是类的设计，同时也讲到，软件设计开发是一个迭代的过程，分析、设计和实现这三个阶段的界限划分并不明显。所以，今天我们只给出了比较粗略的模块划分，至于更加详细的设计，我们留在下一节课中跟实现一块来讲解。

重点回顾

今天的内容到此就讲完了。我们来一起总结回顾一下，你需要掌握的重点内容。

对于非业务通用框架的开发，我们在做需求分析的时候，除了功能性需求分析之外，还需要考虑框架的非功能性需求。比如，框架的易用性、性能、扩展性、容错性、通用性等。

对于复杂框架的设计，很多人往往觉得无从下手。今天我们分享了几个小技巧，其中包括：画产品线框图、聚焦简单应用场景、设计实现最小原型、画系统设计图等。这些方法的目的是为了问题简化、具体、明确，提供一个迭代设计开发的基础，逐步推进。

实际上，不仅仅是软件设计开发，不管做任何事情，如果我们总是等到所有的东西都想好了再开始，那这件事情可能永远都开始不了。有句老话讲：万事开头难，所以，先迈出第一步很重

要。

课堂讨论

今天的课堂讨论题有下面两道。

1. 应对复杂系统的设计实现，我今天讲到了聚焦简单场景、最小原型、画图等几个技巧，你还有什么经验可以分享给大家吗？
2. 今天提到的线框图、最小原型、易用性等，实际上都是产品设计方面的手段或者概念，应用到像框架这样的技术产品的设计上也非常有用。你觉得对于一个技术人来说，产品能力是否同样重要呢？技术人是否应该具备一些产品思维呢？

欢迎在留言区写下你的答案，和同学一起交流和分享。如果有收获，也欢迎你把这篇文章分享给你的朋友。

AI智能总结

本文介绍了如何进行非业务的通用框架开发的需求分析和设计。文章首先提到了项目背景，即开发一个能够获取接口调用统计信息并支持多种显示格式输出的框架。接着，对功能性需求和非功能性需求进行了详细分析。在功能性需求方面，文章列举了接口统计信息、统计信息类型、显示格式、显示终端等需求，并提出了隐藏的需求，如统计触发方式、统计时间区间和统计时间间隔。在非功能性需求方面，文章强调了易用性、性能、扩展性、容错性和通用性的重要性。通过详细的需求分析和设计思路，为读者提供了开发非业务通用框架的指导和思路。

在框架设计方面，文章提到了聚焦于一个简单的应用场景，基于此设计实现一个简单的原型的思想。通过实现一个最小原型系统，帮助理清更复杂的设计思路，为迭代设计提供基础。文章还分享了几个小技巧，包括画产品线框图、聚焦简单应用场景、设计实现最小原型、画系统设计图等，以简化、具体、明确问题，逐步推进设计开发。此外，文章还提到了产品设计方面的手段或概念在技术产品设计上的重要性，以及技术人员是否应具备一定的产品思维。

总的来说，本文通过需求分析和设计思路，为读者提供了开发非业务通用框架的指导和思路，并分享了简化设计思路的小技巧，以及产品设计概念在技术产品设计中的重要性。

© 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。



progyoung

2019-12-30

老师，本文中的案例统计时间时对业务代码是侵入式的，有没有非侵入式的案例呀？

作者回复: 可以使用类似spring aop 做到无侵入

共 9 条评论 >

👍 31



Lyre

2020-01-06

复杂的系统设计，首先应该梳理出功能点，整理架构设计，画出架构设计图，有了总体的规划，做下去才更顺畅。对吗老师

作者回复: 是的，先做设计，后写代码。先做顶层设计，再做细分设计。



👍 19



北天魔狼

2019-12-30

一直没有做过关于统计和监控的项目，希望老师可以出一个小的MVP🙏🙏🙏

作者回复: 39 40讲 会给出完善的代码

共 4 条评论 >

👍 9



Flynn

2019-12-30

老师，后面有TDD相关的内容讲解和练习么

作者回复: 有单元测试的讲解



👍 5



worthto

2020-04-27

Metrics 类的线程是一个单线程的模型，如果线程池使用多个线程，老师能否帮我们设计一款支持多线程并发的统计模型。

作者回复: 😊 这不属于专栏学习的重点



波波

2020-04-04

老师，计算最大响应时间max应该是 响应时间减去请求时间吧，为什么代码中只用到了响应时间呢？

作者回复: 好像这是两个概念啊，不用减啊



淤白

2020-11-23

打卡：用Java实现了文中案例。

作者回复: 加油



王先森

2020-05-22

案例统计时间时对业务代码是侵入式,针对php语言是可以有定时脚本去统计,大家还有什么其他的方式么？

作者回复: java spring可以放到切面中进行~



莫大-潇湘夜雨

2020-01-10

老师，案例选用hashmap来存储监控数据，因为涉及到并发访问，是不是换成并发包更合适一些呢？

作者回复: 是的, 不过这是最小原型, 我也讲到了可以暂时不用考虑并发问题

共 2 条评论 >



JM

2020-01-03

这个例子里面每个接口程序块的开头和结尾都要加上timestamp, 如果要统计成百上千个接口, 是不是有更高效的方式呢? 每个接口可能会隶属于不同的team, 那这样是不是需要和多个team沟通合作, 沟通协作成本很高啊。

作者回复: 1. 我们可以通过切面的方式, 在某一个地方统一加timestamp, 而不用每个接口都加timestamp

2. 每个接口隶属不同team是什么意思呢?

