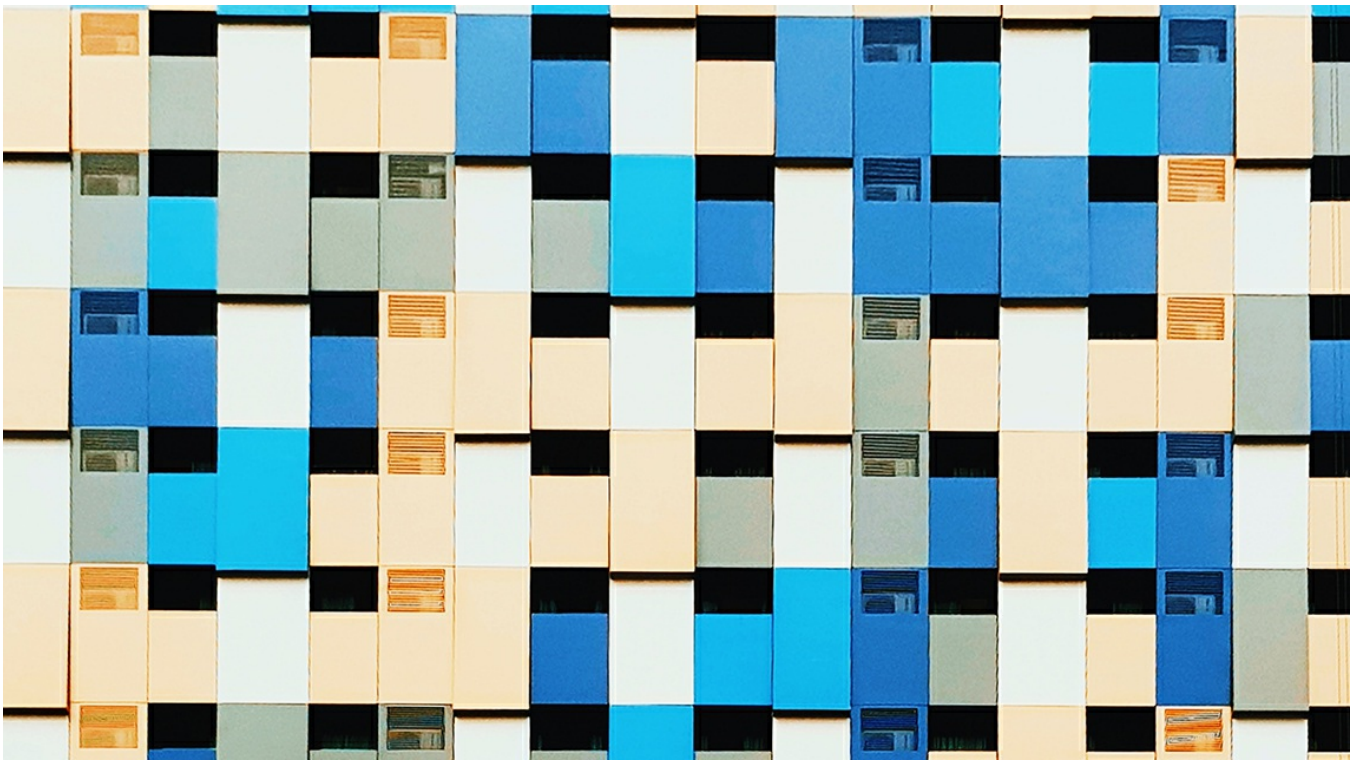
18 | 思考框架：什么样的代码才是高效的代码？

2019-02-13 范学雷



如果让你设计一个有十亿用户使用的售票网站，你会考虑哪些问题？如果让你设计一个有一万亿用户使用的服务，你又会考虑哪些问题？不要以为有一万亿个用户的服务离我们很远，它正在快速地逼近我们。

我们前面讨论了，代码的性能是关于如何管理内存、磁盘、网络和内核等计算机资源的。该怎么衡量这些资源管理的好坏呢？这就需要一些评价指标。

这些指标不仅指导着代码的交付标准，也指导着我们编码时的技术选择。

用户的真实感受

最直接的指标就是用户的真实感受。用户的感受是我们软件开发最基本的风向标，当然也是代码性能追求的终极目标。

如果去超市买东西，我们享受的是购物的过程，讨厌结账。结账之所以令人讨厌，一小部分原因在于这时我们要付钱，更大的原因在于这个过程排队时间可能会很长。如果再算错了帐，就更让人不爽了。

用户对于软件性能的要求，和我们超市结账时的要求差不多：等待时间要短，出错的概率要小。

等待时间要短

这个概念很好理解。等待时间越短，我们越喜欢。最好是一点儿都感觉不到等待时间。使用“感

觉”、“快”、“慢”这种词汇，有点主观了。有一种统计方法，被广泛地用来评价应用程序性能的满意度，它就是应用程序性能指数（Apdex）。

根据任务的响应时间，应用程序性能指数定义了三个用户满意度的区间：

- **满意**：如果任务的响应时间小于T，用户感觉不到明显的阻碍，就会比较满意；
- **容忍**：如果任务的响应时间大于T，但是小于F，用户能感觉到性能障碍，但是能够忍受，愿意等待任务的完成；
- **挫败**：如果任务的响应时间大于F或者失败，用户就不会接受这样的等待。挫败感会导致用户放弃该任务。

在互联网领域，最佳等待时间（T）和最大可容忍等待时间（F）的选择有着非常经典的经验值，那就是最佳等待时间是2秒以内，最大可容忍等待时间是最佳等待时间的4倍，也就是8秒以内。

有了统计数据，应用程序性能指数可以按照下属的公式计算：

> Apdex = (1 × 满意样本数 + 0.5 × 容忍样本数 + 0 × 挫败样本数) / 样本总数

假如有一个应用，100个样本里，有70个任务的等待时间在2秒以内，20个任务的等待时间大于2秒小于8秒，10个任务的等待时间大于8秒。那么，这个指数的就是80%。

> Apdex = (1 × 70 + 0.5 × 20 + 0 × 10) / 100
> = 0.8

80分的成绩能不能让我们满意呢？ 通常来说，80分的成绩还算过得去，90分以上才能算是好成绩。

需要特别注意的是，这个等待时间是用户能够感受到的一个任务执行的时间，不是我们熟悉的代码片段执行的时间。比如说，打开一个网页，可能需要打开数十个连接，下载数十个文件。对于用户而言，打开一个网页就是一个完整的、不可分割的任务。它们并不需要去理解打开网页背后的技术细节。

有了这个指数，我们就知道快是指多块，慢是指多慢；什么是满意，什么是不满意。这样我们就可以量化软件性能这个指标了，可以给软件性能测试、评级了。

体验要一致

为什么90分以上才算是好成绩呢？ 这就牵涉到用户体验的一致性。一致性原则是一个非常基本的产品设计原则，它同样也适用于性能的设计和体验。

一个服务，如果10次访问有2次不满意，用户就很难对这个服务有一个很高的评价。10次访问有

2次不满意，是不是说明用户可以给这个服务打80分呢？显然不是的。他们的真实感受更可能是，这个服务不及格。特别是如果有对比的话，他们甚至会觉得这样的服务真是垃圾。

如果你们了解近年来浏览器的发展历史，就会看到一个巨大的市场份额变迁。微软的IE浏览器在不到十年的时间内，从无可动摇的市场霸主，被谷歌的Chrome浏览器超越，大幅度被甩在了身后，最后被深深地踩在脚下。其中一个非常重要的因素就是，Chrome浏览器的响应速度更快，用户体验更好。就连Windows的用户，都抛弃了IE，转而使用Chrome。不是说IE浏览器不好，而是相比之下，Chrome更好。

一个服务能够提供一致的性能体验，拿到90分甚至95分以上的好成绩，其实有很多挑战。但正是这些挑战，让优秀的程序员和优秀的产品脱颖而出。

比如说，为了性能和安全，谷歌的浏览器和谷歌提供的很多服务之间，甚至抛弃了成熟通用的TCP协议，转向使用性能和安全性更好的QUIC协议。

难道财大气粗、脑力激荡的微软没有反击吗？ 反击当然有，Windows 10启用了全新浏览器Edge，但是没有掀起半点波澜。 2018年10月，微软宣布重构Edge浏览器，使用谷歌的Chrome引擎技术。

这就是一个利用性能优势和用户体验赢得市场地位，成为后起之秀的经典案例。它告诉我们，仅仅做到好，还不能生存，要做到最好。

浏览器是客户端，服务端也需要提供一致的体验吗？

比如说，有一个服务在一年12个月的时间里，有11个月的服务都特别流畅，人人都很满意。但是有半个月，网站经常崩溃或者处于崩溃的边缘，平常需要2秒就搞定的服务，此时需要不停地刷屏排队，甚至30分钟都完成不了。但这项服务特别重要，没有可替代的，不能转身走开，只好隔几秒就刷一次屏。

手动刷屏太累呀，谁也不愿意过5秒点一下刷新。为了解放大家的双手、眼睛还有绝望的心，自动刷屏软件出现了，每隔几秒可以自动模拟刷屏，给大家带来了一线的生机。大家都很欢喜，纷纷安装，用过的奔走相告。久而久之使用刷屏软件的人多了，人们就更加访问不到服务了，等待时间会变得更长，于是又有更多的人使用刷屏软件，更频繁地刷屏，形成了一个恶性循环。

就这样，1千万个人的活动，制造出了100亿个人的效果。我相信，只要你经历过这种让人崩溃的场景，就不会因为它有11个月的优良服务记录为它点赞。如果有客户评价系统的话，你大概率会给个零分，然后丢下一堆鼓励的话。如果这个服务出现了竞争者，你可能会立即走开投向新服务的怀抱。

代码的资源消耗

如何让用户对服务感到满意呢？这就需要我们通过代码管理好内存、磁盘、网络以及内核等计算

机资源。

管理好计算机资源主要包括两个方面，一个方面是把有限的资源使用得更有效率，另一个方面是能够使用好更多的资源。

把资源使用得更有效率

这个概念很好理解，指的就是完成同一件事情，尽量使用最少的计算机资源，特别是使用最少的内存、最少的CPU以及最少的网络带宽。

愿景很美好，但是我们的确又做不到，怎么可能“又要马儿跑，又要马儿不吃草”呢？这个时候，就需要我们在这些计算机资源的使用上做出合理的选择和分配。比如通过使用更多的内存，来提高CPU的使用效率；或者通过使用更多的CPU，来减少网络带宽的使用；再或者，通过使用客户端的计算能力，来减轻服务端的计算压力。

所以，有时候我们说效率的时候，其实我们说的是分配。计算机资源的使用，也是一个策略。不同的计算场景，需要匹配不同的策略。只有这样，才能最大限度地发挥计算机的整体的计算能力，甚至整个互联网的计算能力。

能够使用好更多的资源

这个概念也很好理解，就是当我们面对更多计算机资源的时候，能够用上它们、用好它们。遗憾的是，很多代码是做不到这一点的。

比如说，有一个非常成功的应用程序，受欢迎程度远远超过预期，用户量急剧攀升，系统的响应时间急剧下降，服务器面临崩溃的危险。这是值得庆贺的时刻，是不是？也是可以大胆增加投入的时机，对不对？

这时候，如果换一个128个内核、64TB内存的计算机，把服务器搬到网络骨干机房，取消带宽流量限制，我们能保证这个应用程序用得上这些资源吗？能够解决眼前的危机吗？如果一台机器不够用，这个应用程序可以使用好4台或者16台计算机吗？这个，真的不一定。即便有充足的资源，应用程序的瓶颈可能也不是充沛的资源可以解决的。

不是所有的应用程序设计都能够用好更多的资源。这是我们在架构设计时，就需要认真考量的问题。

算法的复杂程度

如果给定了计算机资源，比如给定了内存，给定了CPU，我们该怎么去衡量这些资源的使用效率？

一个最重要、最常用、最直观的指标就是算法复杂度。对于计算机运算，算法复杂度又分为时间复杂度和空间复杂度。我们可以使用两个复杂度，来衡量CPU和内存的使用效率。

算法复杂度的计算，我相信是大家耳熟能详的内容，我们就不在这里讨论它们的细节问题了。

小结

编写有效率的代码是我们的一项基本技能。要学会这项技能，我们就要了解该怎么去设计、分析、验证代码的效率。从小的代码层面看，我们要有意识、要能够给理解并计算算法的复杂度，来尽量提高每一段代码的效率。从大的架构层面看，我们要采用合适的技术，指导实现的代码能够把有限资源使用的更有效率，也能够在必要时使用更多的资源。从更大的产品层面看，我们一定要关切用户的使用体验和真实感受，特别是糟糕状况下的感受，及时地做出调整。

衡量代码性能的体系和指标很多，你还知道哪些方法？欢迎你分享在留言区，我们一起来学习。

一起来动手

下面的这段Java代码，你能够计算出它的时间复杂度和空间复杂度吗？你知道有什么工具可以分析出这段代码里，哪些地方最耗费时间吗？如果你找到了性能的瓶颈，你有优化的办法吗？

欢迎你在留言区讨论上面的问题，我们一起来看看这一小段代码，是不是可以做的更好？

```
import java.util.HashMap;
import java.util.Map;

class Solution {
    /**
     * Given an array of integers, return indices of the two numbers
     * such that they add up to a specific target.
     */
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (map.containsKey(complement)) {
                return new int[] { map.get(complement), i };
            }
            map.put(nums[i], i);
        }
        throw new IllegalArgumentException("No two sum solution");
    }
}
```

精选留言

轻歌赋

4

hashmap的默认大小的问题吗？
个人感觉是这个原因，解决办法就是创建map时指定大小。
另外hash的计算都是o1的时间复杂度，但是put这种写操作要比读操作慢。这个暂时没有想到替代方案。
每次都要创建差值感觉比较慢，可以反过来查询当前值，存放差值。
传入的数组不知道有没有序，但是可以默认有序，然后前后交换着访问。即先访问第一个，再访问最后一个，再访问第二个，反复进行
两数之和不知道有没有负数参与的情况。
个人感觉这个算法的时间复杂度已经是o1了，只能从其他角度考虑更快，空间复杂度偏大，考虑有序情况可以前后交换查找。但是无序情况没有帮助。
当数据很大的情况下，可以考虑并行算法。
这个问题的解法感觉最终的结果不应该是单个，可能有多组两数之和都满足的情况，程序设计个人感觉有点问题，实在是想不到更多了。欢迎大家和老师评论，目前还是实习阶段，经验难免不足，希望各位能够指出不足，共勉

2019-02-13

作者回复

很厉害！最大的性能问题虽然还没有找到，但哪真的是时间积累的问题。你找的一些问题已经很有见地了，比如负数、有序、并行。比如按照我们的思考习惯，很难想到负数的问题，这里面有一个安全漏洞，我们第三篇接着聊。

2019-02-14

苏志辉

0

感觉剖异常比较耗性能，需要生成堆栈，可以返回空值

2019-03-15

作者回复

是的，没有匹配的可以看作一个正常状况。返回空值（空数组），更好些。

2019-03-15

王子瑞Aliloke有事电联

0

很想有时间的时候自己试着开发一个浏览器；|

但这个目标的是我考上研究生之后实施。

2019-03-05

作者回复

嗯，期待！

2019-03-05

李星

0

for循环时间复杂度可能是O(n),hashmap的put和containskey都是O（1）吧

2019-02-17

作者回复

是的

2019-02-17

aguan(^Ⅲ^)

0

时间复杂度O(n)，n表示数组的长度

空间复杂度O(1)

2019-02-14

作者回复

时间复杂度有三个，一个是for语句，一个是hashMap查询，一个是hashMap的put()。空间复杂度是hashMap占用的空间。你再分开来想想？

2019-02-14