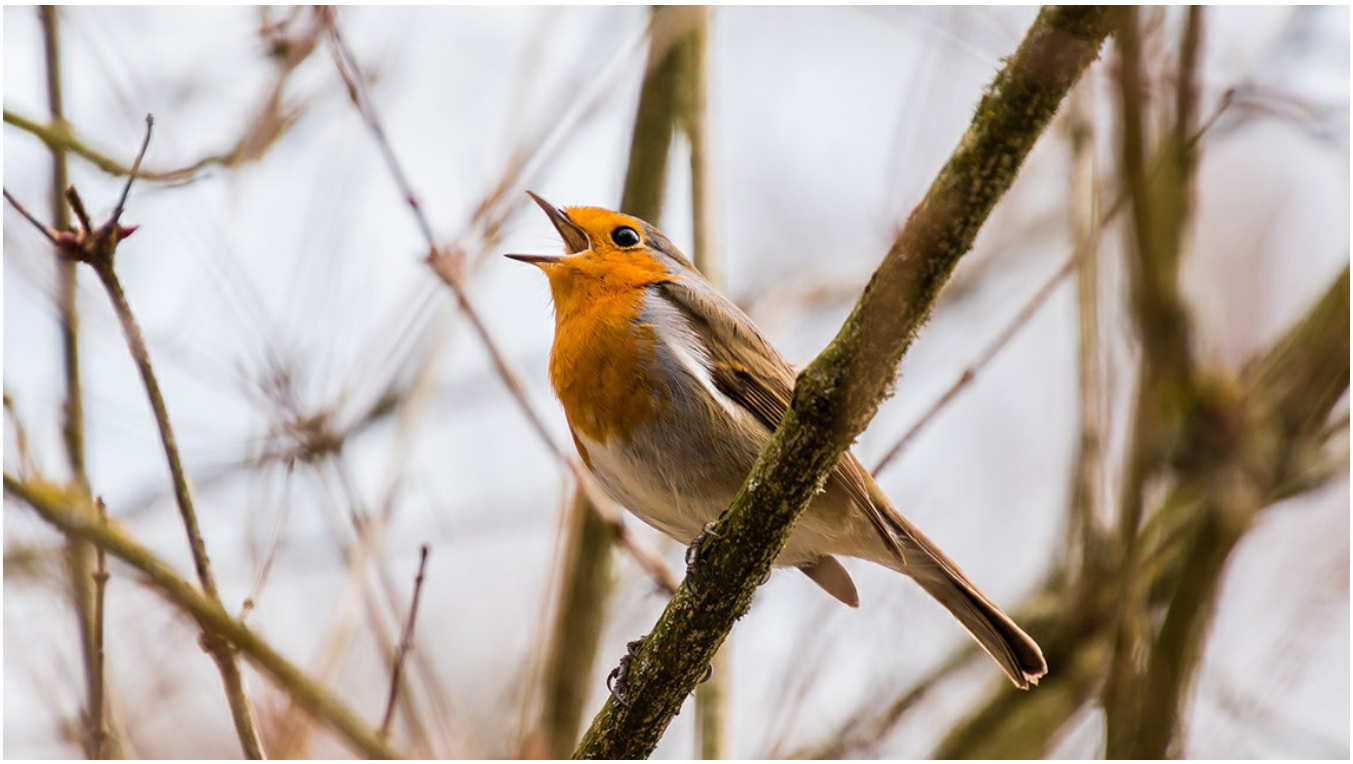
17 | 为什么需要经济的代码？

2019-02-11 范学雷



如果你在线购买过春运的火车票，经历过购票网站的瘫痪，你应该深有体会，网站瘫痪是一件让人多么绝望的事情。

根据有关报道，2014年1月9日，火车票售票网站点击量高达144亿次，相当于每个中国人点击了10次，平均每秒点击了16,000次，峰值的点击量可能远远超出16,000次。这么强悍的访问量，导致了火车售票网站多次瘫痪。这是一个典型的性能错配导致的重大网络事故，处理这么大的点击量需要特殊的程序设计和架构安排。

有句俗话说：“又要马儿跑，又要马儿不吃草。”马该怎么活呀？活不了呀！要想让马跑得快，既要有好马，也要有好料。

如果可以把软件比作一匹马的话，让这匹马出生时有一个优良的基因，平日里精心地伺候，是让它跑得快的先决条件。

前一段时间，我们讨论了如何让代码“写得又快又好、读得又快又好”的话题。接下来的这段时间，我们来聊聊怎么让代码“跑得又快又好”。跑得又快又好，一般也意味着更少的运营费用。该怎么让我们写的代码有一个跑得好的基因呢？

需不需要“跑得快”的代码？

很多项目是面向订单的，代码的功能是需要优先考虑的任务。这并没有错误。如果不能兼顾性能，这个债将来还起来会很痛苦，成本很高。而且，很多情况下，它是躲不开、赖不掉的。

怎么理解代码的性能？

为了理解这个问题，我们需要对代码的性能有一个共同的认识。代码的性能并不是可以多块地进行加减乘除，而是如何管理内存、磁盘、网络、内核等计算机资源。代码的性能与编码语言关系不大，就算是JavaScript编写的应用程序，也可以很快，C语言编写的程序也可能很慢。

事实上，代码的性能和算法密切相关，但是更重要的是，我们必须从架构层面来考虑性能，选择适当的技术架构和合适的算法。很多纸面上看起来优美的算法，实际上很糟糕。也有很多算法看起来不咋样，但实际上很高效。为了管理代码的性能，在一定程度上，我们需要很好地了解计算机的硬件、操作系统以及依赖库的基本运行原理和工作方式。一个好的架构师，一定会认真考虑、反复权衡性能要求。

需不需要学习性能？

一个程序员，可以从多个方面做出贡献。有人熟悉业务逻辑，有人熟悉类库接口，有人能够设计出色的用户界面。这都非常好，但是如果考察编程能力，有两件事情我们需要特别关注。

第一件事情是，我们的代码是不是正确？事实上，代码正确这个门槛特别低。如果代码出现了大范围的错误，说明编程还没有入门。

第二件事情是，我们的代码运行起来有没有效率，运营成本低不低？这也是我们判断代码是否经济的一个标准。编写经济的代码的门槛稍微高一些，它需要更多的知识和经验，但它也是能让我们脱颖而出的一个基本功。门槛越高，跨越门槛的价值就越大。我们要是一直不愿意跨越这个高门槛，面临的竞争压力就会越来越大。

这个价值到底有多大呢？就我熟悉的领域来说，如果你可以把Java垃圾管理器的效率提高50%，或者把列表的查询速度提高50%，更或者，你能用三五台服务器解决掉春运火车票售票网站崩溃的问题，那么找到一份年薪百万的工作是不难的。

当然上面的一些问题实现起来非常困难，比如提高Java垃圾管理器的效率。但是，需要我们解决的性能问题，很多时候，都不是技术问题，而是意识和见识的问题。成熟的解决方案就在那儿，容易理解，也容易操作。只是，我们没有想到，没有看到，也没有用到这些解决方案。我们越不重视性能，这些知识离我们就越远。

一个好的程序员，他编写的代码一定兼顾正确和效率的。事实上，只有兼顾正确和效率，编程才有挑战性，实现起来才有成就感。如果丢弃其中一个指标，那么大多数任务都是小菜一碟。

有过面试经验的小伙伴，你们有没有注意到，正确和有效地编码是面试官考察的两个重点？招聘广告可不会提到，程序员要能够编写正确的代码和有效的代码。但是一些大的企业，会考察算法，其中一条重要的评判标准就是算法够不够快。他们可能声称算法考察的是一个人的基本功，是他的聪明程度。但是如果算法设计不够快，主考官就会认为我们基本功不够扎实、不够聪明。

你看，算法快慢大多只是见识问题，但很多时候，会被迫和智商联系起来。这样做既无理，也无聊，但是我们也没有办法逃避开来，主考官可能也没有更好的办法筛选出更好的人才。

需不需要考虑代码性能？

具体到开发任务，对于软件的性能，有很多误解。这些误解，一部分来自我们每个人都难以避免的认知的局限性，一部分来自不合理的假设。

比如说，有一种常见的观点是，我们只有一万个用户，不要去操百万用户的心。这种简单粗暴的思考方式很麻烦！你要是相信这样的简单论断，肯定会懵懂得一塌糊涂。百万用户的心是什么心？你根本没有进一步思考的余地。你唯一能够理解的，大概就是性能这东西，一边儿玩去吧。

一开始，我们就希望大家能从经济的角度、从投入产出的角度、从软件的整个生命周期的角度来考虑代码。我们要尽量避免这种不分青红皂白，一刀切下去的简单方式。这种简单粗暴的方式可能会帮我们节省几秒钟的时间，我们思考的快系统喜欢这样，这是本性。但是，我们真的没必要在乎这几秒钟、几分钟，甚至是几小时，特别是在关乎软件架构和软件质量的问题上。该调用我们思考的慢系统的时候，就拿出来用一用。

我们可以问自己一些简单的问题。比如说，一万个用户会同时访问吗？如果一秒钟你需要处理一万个用户的请求，这就需要有百万用户、千万用户，甚至亿万用户的架构设计。

再比如说，会有一万个用户同时访问吗？也许系统没有一万个真实用户，但是可能会有一万个请求同时发起，这就是网络安全需要防范的网络攻击。系统保护的东西越重要，提供的服务越重要，就越要防范网络攻击。而防范网络攻击，只靠防火墙等边界防卫措施，是远远不够的，**代码的质量才是网络安全防护的根本**。

你看，哪怕我们没有一万个用户，我们都要操一万个用户的心；当我们操一万个用户的心的时候，我们可能还要操百万用户的心。

你有没有意识到，你操心的程度和用户量的关系不是那么紧密？你真正需要关心的，是你的代码有多重要？ 代码带来的绝对价值越大，消耗的绝对成本越高，它的性能就越重要。

当然，也不是所有的软件都值得我们去思考性能问题。有人统计过，大概90%以上的软件，都没有什么实际用处，也就是说，运营价值非常小。比如我们的毕业论文代码，比如入门教科书的示例代码，比如我们为公司晚会写的、用完就扔的抽奖程序。这是对的，大多数代码的性能优化是无用的，因为它们并没有多大的实际运营价值。

但是，如果我们要去开发具有商业价值的软件，就要认真思考代码的性能能够给公司带来的价值，以及需要支付的成本。

经验告诉我们，**越早考虑性能问题，我们需要支付的成本就越小，带来的价值就越大**。甚至是，和不考虑性能的方案相比，考虑性能的成本可能还要更小。

你可能会吃惊，难道优化代码性能是没有成本的吗？ 当然有。这个成本通常就是我们拓展视野和经验积累所需要支付的学费。这些学费，当然也变成了我们自身市场价值的一部分。

有时候，有人说：“我们只有一万个用户，不要去操百万用户的心。”其实，潜台词是说，我们还没有技术能力去操一百万个用户的心，也没有时间或者预算去支付这样的学费。这其实对我们是有利的。一旦我们有了这样的见识和能力，我们就可以发挥市场的价值。这是一个可以赚回学费的机会，也会让我们变得越来越有价值。

什么时候开始考虑性能问题？

为了进度，很多人的选择是不考虑什么性能问题，能跑就行，先跑起来再说；先把代码摞起来，再考虑性能优化；先把业务推出去，再考虑跑得快不快的问题。可是，如果真的不考虑性能，一旦出了问题，系统崩溃，你的老板不会只骂他自己，除非他是一个优秀的领导。

硬件扩展能解决性能问题吗？

有一个想法很值得讨论。很多人认为，如果碰到性能问题，我们就增加更多的机器去解决，使用更多的内存，更多的内核，更快的CPU。网站频繁崩溃，为什么就不能多买点机器呢？！

但遗憾的是，扩展硬件并不是总能够线性地提高系统的性能。出现性能问题，投入更多的设备，只是提高软件性能的一个特殊方法。而且，这不是一个廉价的方法。过去的经验告诉我们，提高一倍的性能，硬件投入成本高达四五倍；如果需要提高四五倍的性能，可能投入二三十倍的硬件也达不到预期的效果。硬件和性能的非线性关系，反而让代码的性能优化更有价值。

性能问题能滞后处理吗？

越来越多的团队开始使用敏捷开发模式，要求拥抱变化，快速迭代。很多人把这个作为一个借口：我们下一次迭代的时候，再讨论性能问题。他们忘了敏捷开发最重要的一个原则，就是高质量地工作。没有高质量的工作作为基础，敏捷开发模式就会越走越艰难，越走越不敏捷，越走成本越高。而性能问题，是最重要的质量指标之一。

性能问题，有很多是架构性问题。一旦架构性问题出现，往往意味着代码要推倒重来，这可不是我们可以接受的快速迭代。当然，也有很多性能问题，是技术性细节，是变化性的问题。对于这些问题，使用快速迭代就是一个经济的方式。

很多年以来，我们有一个坏的研发习惯，就是性能问题滞后处理，通过质量保证(QA)环节来检测性能问题，然后返回来优化性能。这是一个效率低、耗费大的流程。

当应用程序进入质量保证环节的时候，为时已晚。在前面的设计和开发阶段中，我们投入了大量时间和精力。业务也要求我们尽快把应用程序推向市场。如果等到最后一分钟，才能找到一个严重的性能问题，推迟产品的上市时间，错失市场良机，那么这个性能问题解决的成本是数量级的。没有一个企业喜欢事情需要做两遍才能做到正确的团队，所以我们需要在第一时间做到正

确。

要有性能工程的思维

采用性能工程思维，才能确保快速交付应用程序，而不用担心因为性能耽误进度。性能工程思维通过流程“左移”，把性能问题从一个一次性的测试行为，变成一个贯穿软件开发周期的持续性行为；从被动地接受问题审查，变成主动地管理质量。也就是说，在软件研发的每一步，每一个参与人员，都要考虑性能问题。整个过程要有计划，有组织，能测量，可控制。

采用性能工程思维，架构师知道他们设计的架构支持哪些性能的要求；开发工程师清楚应该使用的基本技术，而不是选择性地忽略掉性能问题；项目管理人员能够在开发软件过程中跟踪性能状态；性能测试专家有时间进行负载和压力测试，而不会遇到重大意外。实现性能要求的风险在流程早期得到确认和解决，这样就能节省时间和金钱，减轻在预算范围内按时交付的压力。

现在很多公司的研发，完美地匹配了敏捷开发和性能工程这两种模式。降低研发成本的同时，也促进了员工的成长，减轻了程序员的压力。

小结

最后，我们总结一下。编写有效率的代码是我们的一项基本技能。我们千万不要忽视代码的性能要求。越早考虑性能问题，需要支付的成本就越小，带来的价值就越大，不要等到出现性能问题时，才去临时抱佛脚。另外，性能问题，大部分都是意识问题和见识问题。想得多了，见得多了，用得多了，技术就只是个选择的问题，不一定会增加我们的编码难度和成本。

接下来的这一模块，我们会聚焦在解决性能问题的一些基本思路和最佳实践上，比如架构设计问题、内存管理问题、接口设计问题和开发效率问题等等。

最后问你个问题吧，你有因为性能问题承担过巨大的压力吗？这个性能问题是怎么来的？最后怎么解决的？欢迎你在留言区分享你的想法。

一起来动手

下面的这段代码，我们前面使用了很多次，主要是为了学习编码规范。其实，它也有性能问题。这一次，我们来试着优化它的性能。

我先要说明的是，如果你之前没有接触过类似的问题，那么它是有点难度的。如果你已经接触过类似的问题，这个问题就是小菜一碟。这就是一个见了多少、经验也就有多少的问题。

欢迎你把优化的代码公布在讨论区，我们一起来看看性能优化后的代码可以是什么样的？

```
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

王智 3

自我表示还没成长到这一步,但是感觉性能问题是很重要,平常看微信公众号,掘金等知识文章会有很多关于性能出现问题造成的很严重的后果,还有很多如何提高性能的文章,从这点来看,性能确实是软件开发中很重要的点.
就我现在的理解,只能感觉算法很重要,现在写代码都是只要能实现就行,能跑就行,没有考虑到其它,还得修改进步.
加油,新的一年!!!

2019-02-11

作者回复

后面我们会聊一些性能的问题，不同于算法的。

2019-02-12

pyhhou 1

像是结尾不应该抛出异常，还有 map 需要设置大小，不然会有频繁扩容的问题，另外就是一开始判断数组 nums 为 null 或者空的话可以直接返回空数组，这些我看前面的同学都说了。但是回看这个问题，往深了想，其实很多情况没有交代明白，其中一个我想到的就是内存的问题，这个示例算法的确很快，O(n)时间复杂度，但是也会有额外的 O(n) 的空间消耗，如果说数组 nums 非常非常的大，假如说是 100G，而此时的剩余内存不足 1G，这样子的话这种方法明显行不通；在这种极端情况下，仅仅从算法角度考虑的话，退而求其次的方法是将整个数组用快排排一下序，然后用双指针分别从数组两头相向走一遍，时间复杂度O(nlgn)，空间复杂度O(1)。这也是验证了老师文章当中讲到的，考虑一个算法好坏与否，仅仅考虑算法本身是完全不够的，再精美的算法也需要结合计算机硬件，操作系统还有整体的一个架构进行考虑筛选～

2019-03-03

陈杰 0

用set应该就可以了，不需要用map。如果用加法可以不用保存结果，但时间复杂度相对弱一些

2019-03-09

悲劇の輪廻 0

好的算法确实很重要，难怪说国外程序员应聘之前必先刷LC.....

2019-03-05

秦凯 0

对性能和资源消耗有一定的意识，但是具体的开发过程中或者应用运行过程中对性能进行监控、评测、分析就束手无策了。[捂脸]

2019-03-01

阳阳阳 0

nums.length 可以申明一个变量，不必每次循环都计算

2019-02-26

作者回复

nuns.length是一个变量，不是方法。

2019-02-27

小狼 0

示例代码是leetcode上两数之和的算法题，官方题解给出的一种答案。性能问题在于算法的时间复杂度和空间复杂度都是o(n)。该示例代码影响性能的最大因素就在于算法的时间辅助和空间复杂度可以再降一降，另外就是如果没有找到符合要求的两数，可以返回null，不必抛异常

2019-02-18

作者回复

能分享下复杂度怎么再降一降吗？

2019-02-18

鲸息 0

再往底层虚拟机考虑的话，问题可能是 Integer 的缓存问题，导致大量重复对象创建，因为默认范围是 -128 到 +127

2019-02-15

作者回复

没看明白-128到+127的说法。

2019-02-15

鲸息 0

还有一个难找的性能问题的话，我认为可能就是算法层面了。没有对数组进行任何处理就进行计算了，当数组很大时，可能会有很多不需要计算或者重复的。

2019-02-15

作者回复

你也许已经找到了这个难找的问题了。重复的会带来什么问题？

2019-02-15

aguan(^□□^) 0

1.判断入参num数组为空时，直接返回空，这样就不用创建map

2.当for循环结束后为找到满足条件的数据是返回空，而不是直接抛出异常

2019-02-14

作者回复

找的好，这两点都影响性能。还有一些问题，留言区里有找到，后面的章节我们也会讨论到。

2019-02-14

鲸息 0

性能问题有两个：

1. HashMap 没有指定大小，如果求组很大会导致频繁扩容
2. 抛异常不太好，异常堆栈会有性能开销

P.S.

拆箱装箱在编译时就已经做了，不会有太大性能问题

2019-02-14

作者回复

找的都对，这些都影响性能。还有一些问题，留言区里有讨论，我们后面的章节还会再讨论。

2019-02-14

Void_seT

0

性能问题是出现在“入参数组过大，导致map频繁扩容”么？

2019-02-13

作者回复

这是其中一个问题。这个练手题有多个性能问题，其中比较难找的一个，我们后面还会专门讨论。

2019-02-14

hua168

0

老师我有2个疑问：

1.是不是编程学到一定程度下都要求学一下算法？一些常用用算法，我看极客专栏也有算法的文章

2.我看很多小公司都没有用到什么多少算法之类，如果自己的目标是中小公司是不是不用学算法了？

2019-02-12

作者回复

算法还是要有个概念的，大家应该都学过数据结构与算法吧？

不过，算法太庞大驳杂了，我也不建议大家去钻研不常用的算法，比如加密算法的优化这种专业的东西。

编程学到一定程度，我觉得是根据要解决的问题去找适合的技术。要是需要实现算法，那就一定要钻研算法；要是不涉及算法，也许你很快就会忘记学过的算法。这个和公司大小没关系，和你要做的事情有关系。

2019-02-13

DemonLee

0

最后给出的代码性能问题是指 自动拆箱和装箱吗？老师可以在下一期里面给出解题思路么，谢谢

2019-02-11

作者回复

拆箱/装箱有点影响，但是不是主要的。练手题我们最后都会解释的。

2019-02-12