优秀的程序员都有哪些习惯?

本文由 伯乐在线 - 阿慧 翻译，Lada 校稿。未经许可，禁止转载！
英文出处：nostrademons。欢迎加入翻译组。

【伯乐在线导读】：「我不是卓越的程序员，我只不过是一个有着卓越习惯的程序员。」技术大牛 Kent Beck 曾这样说过自己。

7月初，nostrademons 在 Hacker News 发起一个讨论，是哪些习惯成就了优秀/卓越的程序员?

可变的习惯：学习着在不同的情况中采用不同的习惯。考虑到这一点，我总结了一些适用于不同情况的通用技巧：

为了数据科学类问题研究新领域的发展：

1.尽量亲自动手去完成事情。你将会有一种直觉，知道如何去处理该事物。

2.积累案例，从数据表中标注着你已获得的数据开始。

（关于这一点：rluhar 评论补充说，这不仅适用于数据科学，也适用于解决任何数值问题。用一个电子表格（或一个R / Python Notebook）来实现算法并获得一些结果，在过去帮助我真正理解了问题，避免走死胡同。例如，在建立一个外汇定价系统时，我能够使用电子表格来描述定价算法是如何工作的，并向交易者（最终用户）解释它。在执行和部署算法之前，我们可以调整计算并确保一切都清楚。很好的建议！）

3.在找到通用办法之前，先找到一种能解决当前问题的办法。

4.让算法本身输出调试信息。你应该能够转储每一步的中间结果，并用文本编辑器或是 Web 浏览器手动检查它们。

5.不要为单元测试操心，定义出正确行为才是首要的。

维护大的、不熟悉的代码库程序：

1.检查文件的大小。最大的文件总是包含了程序的实体部分，至少是指向程序实体内容的分派程序。 main.cc 通常很小，并且对寻找代码结构毫无用处。

2.从主循环调度开始单步调试程序。可以学到很多关于控制流的东西。

3.寻找数据结构，特别是做为参数传递到许多函数中的那些。大多数程序具有一个小的关键数据结构集合，找到它们，理解代码的其余部分会变得容易的多。

4.写单元测试。这是确认你所理解的代码与真实代码工作方式无误的最好方法。

5.移除代码，看看什么出问题了。（但不要 check in！）

性能工作：

0.一般不需要做，除非你所构建的东西对用户来说太慢了。制定出需要提高的性能目标，达成这个目标即可。

1.在开始所有工作之前，甚至是在剖析（profile）之前，建立一套代表典型现实世界的使用基准。别让性能倒退，除非你确信已经登峰造极，高处不胜寒，并且更好的解决方法还藏在世界的某个角落里无人发现（如果出现了那种情况，在版本控制系统（VCS）中标记你的分支，以便在发现错误之后回来更改。）。

2.许多性能瓶颈都出现在系统的交叉处。在所有 RPC 框架中搜集时间统计数据，并且有一些方式来传播和可视化每个服务器的请求时间，以及哪些部分的请求是并行的，哪里是关键路径。

3.剖析（Profile）。

4.通常，避免不必要的工作可使你赢在起跑线上。缓存最大的计算结果，粗略评估不常用的东西。

5.不要忽视常量因素。有时候一个渐进性更差的算法在实践中执行的很好，原因是其具有更好的缓存局部性。为此，你可以在多次调用的函数中识别出威胁。

6.当你获得了程序大致剖析之后，更改数据结构往往会获得更多收益。注意内存的使用，时常缩小内存需求来减小缓存压力，可显著地提高系统的速度。注意局部性，将常用数据放到一起。如果你的编程语言允许（为 Java 感到遗憾），可以消除指针雕镂（pointer-chasing）来支持值控制。

> 译注：指针雕镂（Pointer-chasing）程序：该程序中会遍历一个由指针链在一起的数据结构，即一个链表。但是在遍历的过程中会不断的引起内存操作。因为下一个元素总不在 Cache 中。

编码常规

1.不要想当然地去构建，确保你所加入的每个特性都有客户在用。

2.谨慎地控制依赖。为了某个效果而引入的库，可能会帮你节省一个小时，但也会导致更多地方被破坏——部署、版本控制、安全性、日志记录、意外的进程死亡。

3.当为个人或小团体开发的时候，把出现的问题累积起来，然后一次性全部解决（或者扔掉代码库，然后重新启动）。当为大型团队开发时，永远都不要让问题堆积，代码库应该始终处于新的开发人员可以看懂的状态，他们会说：“我知道这是做什么的，也知道如何更改它”（代码的）阅读者/编写者的比例结果是这样的：初始代码的编写多过阅读，因此可读性不那么重要，但成熟代码的阅读多过编写。（当你要求开发前一种初始代码去获得用户、资金和存活了，转换到后一种成熟代码中去就是给阅读者留下的操练了。）

欢迎大家补充。

> **打赏支持我翻译更多好文章，谢谢！**
>
> 打赏译者