有哪些优秀程序员知道而新手不知道的秘诀？ - 文章 - 伯乐在线

【伯乐在线导读】：

> 有国外网友在 Quora 上发帖提问：优秀程序员有哪些秘诀？ 本文摘编了两个高分回复，一起来围观。欢迎大家在评论中补充你眼中优秀程序员的特点。

Jens Rantil 的观点：2.9k 顶

大多数情况下，使用继承从长远看来是一个很糟糕的面向对象设计。它影响了代码的重用性和可测试性。推荐使用组件和接口的方式替代它。

避免过早引入接口，除非你能够完全控制它。“过早接口”同样会导致设计问题。

深度代码嵌套（both intra-function and inter-function） 1 ）难于维护 2 ）更容易产生bug 3 ）难于复用。精炼的代码层次划分更利于以后的代码测试和代码复用。参照上一条关于继承的问题。

预估时间在开发中是一件很难的事情。这就是为什么在很多地方使用敏捷开发（Scrum）和 Sprints 。参见：《为什么程序员总是不能准确估测项目时间？》、《趣文：为什么软件开发周期通常是预期的两三倍？》

完美的加密是很难做到的。除非有一个很好的理由，否则不要尝试自己去发明它。

无副作用逻辑（Side-effect free logic）是非常棒的，这使了解程序状态更简单(参看下一条)，同时也简化了自动化测试。

学习去了解程序的状态切换和生命周期。参看 Jens Rantil 的博客。

如果选则合适的实现方式，并发并不难。线程池、队列、观察者模式、非可变性和Actor模型编程可以帮你很多。

过早优化是万恶之源。一个好的开发过程一般是：1）让它工作。 2）使代码很漂亮。 3）优化。

了解程序的基本数据结构和理解时间复杂度。这是一个有效使代码变得更快在的方法，且不会增加代码复杂度。

练习粗略计算。一段代码通常在内存中占用多少？相关；

应用程序早晚会出问题；不好的部署，意想不到的行为，意想不到的输入或意想不到的外部加载。为了应对这些问题。我们需要确保用日志记录下那些未捕获的异常，在部署工作完成后进行测试（并可能回滚），并不停地运行测试，也要确保为所有内存中的队列和线程池设置限制（理智的）。相关;

如果你监控队列大小的时候，发现它通常总是满的或空的。应对好这些。

网络服务和外部服务应该一直保持片状（易剥离的，即插即播）。为每一个套接字（Socket）设置超时，在每次HTTP读或者连接操作时都设置超时。建议可以将外部网络封装到一个重试/循环断路库中。(参考 Netflix/Hystrix 和 rholder/guava-retrying)

写出你自己都愿意读的代码。增加注释，除非你认为还能在一年后看懂它。在一个月后你会需要这些注释的。有些相关;

为项目设置构建工具，这样别人可以更快开始工作。在你的README文件中记录下要构建、运行、测试和打包的命令。

确保你的项目可以从命令行构建，会使事情变的容易得多。

在许多语言中，处理第三方依赖关系是一个真正混乱的事情（看看你的Java和Python工程）。特别是当两个不同的库依赖于不同版本。使你远离这些麻烦的好方法：1）不断审视你的依赖库。 2）自动化测试有助于解决此类问题。 3）总是关注你应该使用哪个版本的第三方依赖库。

阅读流行的开源项目，是学习编写可维护代码和软件开发流程的好方法。
为应用添加的每一行代码都会增加项目的复杂性，同时使它发生错误的可能性更高。删除代码是一个消除错误的好方式。相关;

应用中的每一个基础服务（数据库、缓存、消息队列等）都是一个bug的来源，需要对它们进行维护，同时也需要学习新知识。更不用说，这种依赖性可能会降低你的生产力。仔细衡量新基础设施的效率。你还需要用一个新的基础设施取代一个旧的吗？

代码路径导致的失败，很少在测试或执行中出现。但往往这就是bug的原因。

输入验证不仅仅是出于安全原因的，它能帮助你尽早发现bug。
和上一条有一些关系：有些相关状态验证和输出验证可以作为输入验证同样有用，无论是为了发现内在的bug，还是为了安全性。

代码评审（code review）是一个用来提高程序员的好方法。你会得到关于你代码的批评，同时你将学会描述为什么其他人的代码的好坏。它还引导你发现常见的错误。

学习一门新编程语言是一种很好的方式，以了解新的范式和旧的习惯问题。

总是指定编码转换文本与字节；当读/写网络、文件或加密时。如果你依赖于语言环境的字符集最终你一定会遇到数据损坏。如果你可以自己选择，请使用UTF字符。

了解你的工具，包括编辑器、终端、版本控制系统（如git）和编译工具。

学会在使用你的开发工具时不用鼠标。学习尽可能多的键盘快捷键。它会让你更高效而且这更符合人体工学。

重用代码不是一个最终的目标，也不会使你的代码更易于维护。重用复杂的代码，但请注意，两个域之间的代码重用可能使它们产生过度的依赖。

长时间坐在电脑会伤害你的身体。 1 ）了解你的身体状况。看看你的背、脖子和手腕。当你身体感到不适的时候休息一会。养成一个暂停的习惯（泡茶，拿咖啡）对你的身心会有意想不到的益处。 2 ）时不时的远离屏幕来休息一下你的眼睛。 3 ）选一个适合你手腕的好的键盘。

自动化测试，特别是单元测试，不仅仅是测试您的代码。它们还： 1 ）告诉大家代码的使用范围； 2 ）还能帮助你提前的适应生产环境。后者是为什么一些人声称测试优先的开发方法可以产生干净的API。

适度的测试。因为缺少测试会导致bug，而查找bug会降低你的开发速度。过度的测试也会降低你的速度，因为每个改变都需要更新太多的测试。

和编译语言相比，动态语言通常需要更多的测试工作来保证它的正常工作。(离线代码分析工具也很有用。)

竞态比普遍认为的更常见。这是因为计算机通常比我们习惯于TPS。

理解吞吐量和延迟之间的关系在优化您的系统时非常有用。相关;

很多时候高吞吐量可以通过引入智能匹配的方式来完成。

保持你提交的代码简短、可用，提交代码的时候写下你做了什么的日志，告诉大家你为什么做这些。有效的提交信息是解决bug的先决条件。

保持你的本地分支尽可能的新。我的经验是：失败发生的风险和你本地版本保持的时间成正比。不要在一个分支上工作超过两周时间。那些比较大的特性，我们可以将它分为多个小的特性并分多次提交它。

尽早的了解你的产品运行环境，并根据环境对部署策略进行调整。
令人惊讶的是，频繁的发布代码会降低风险，而不是增加。

学习面向对象的语言是很容易的。掌握良好的面向对象设计是困难的。了解面向对象设计和面向对象的设计模式将改善你的面相对象设计的理解。

好的架构师很可能会写出很糟糕的代码。然而，一个优秀的架构师知道如果保证系统是解耦的，可更替的。首先要保证这是一个理智的分离的架构。其余的问题可以在时间不紧的时候进行清理。

巴士系数也是你团队中的一个严重风险。（一个项目至少失去若干关键成员的参与（被巴士撞了，比喻）即导致项目陷入混乱、瘫痪而无法存续时，这些成员的数量即为巴士系数。）作为团队的一员，大多数你写的代码需要被别人读或者修改。这包括你早些时期在项目中写的代码。在一开始你就应该写好文档和提交信息。此外，代码评审和脚本也可以帮助你分享一些知识。最后，确定你不是唯一一个知道密码的人。等等

Jeff Darcy 的观点，4.6k 顶

了解概念

通过记忆和模式识别来解决问题要比单纯的寻找问题的原因快的多。如果你曾经解决了一个类似的问题，那么你很可能直观的想起解决方案。就算做不到这一点，至少过去保留的经历会让你更容易的找到解决问题的灵感。“自动化”的解决一个问题看起来很像魔术，但是这就是 Miguel Paraz 建议的“练习 练习 再练习”的真实效果。

了解你的工具

这是一个在编程时维持而不是终止连续性的方法。每当你不得不考虑如何去学习编辑器、版本控制系统、调试器，这将打断你的深度思考。这些“小插曲”虽小，但是它们加起来就不小了。比起什么也不做，那些愿意花时间学习工具、练习使用工具、用自动化工具的人的生产力会得到几倍的提高。

做好时间管理

使自己再次进入工作状态。你想写代码，写代码。你想回顾一些修改补丁，回顾一些修改补丁。如果想头脑风暴新的算法……你懂得，不要试图将这些事情一起做完，一定不要让自己被电子邮件、IRC、Twitter和Quora打断。让你的大脑专注于一件事，在做完这件事情之后再转向做其他的事情。

安排好处理事情的顺序

我发现这方面很多人做的很失败。每一个问题都可以从不同的方面来对待。通常，解决一个问题的一方面会比其他方面容易很多。因此，正确的顺序尤为重要。恐怕没人能说出如何排序的标准答案，但是当你在当前领域经验越来越丰富的时候，再去实践，你的经验会给你很多启发。

重用所有你能重用的

重用的想法。重用代码。每次你把一个新问题变成一个你已经知道如何解决问题，计算充满这样的机会——你可以节省时间。不要担心如果转换后的解决方案并不是绝对完美的当前问题。你可以完善之后，如果你真的需要，通常你会发现，你最好继续下一个问题。

这些事情真的会降低你的效率。随着你每天遇到更多的问题，你将获得更多的经验，这会让你能更快的

解决更多的问题，这样一直下去。这是一个循环，一旦你能够正确的对待这些事情时，你的效率和价值将会得到大幅度的提升。

欢迎大家在评论中补充你眼中优秀程序员的特点。

加入伯乐在线专栏作者。扩大知名度，还能得赞赏！详见《招募专栏作者》

> 打赏支持译者翻译更多好文章，谢谢！

5 赞 9 收藏 5 评论