链滴

The Key To Accelerating Your Coding Skills（中文翻译）

作者：tangseng233

原文链接：https://ld246.com/article/1608114183822

来源网站：链滴

<h2 id="-The-Key-To-Accelerating-Your-Coding-Skills"> The Key To Accelerating Your Codin Skills</h2>
<blockquote>
原作者：KEN MAZAIKA
原文地址：The Key To Accelerating Your Coding Skills
</blockquote>
学习编程的时候，有那么一个瞬间会感觉一切都发生了变化。在 Firehose 里，我们称这个瞬间做学习编程的拐点。越过了这个拐点以后，作为一个开发者，你的开发水平和方式会发生显著的变化从零开始到拐点的过程，是一个逐步在编程方面成为自力更生的开发者的过程，并且最终达到不需要何手把手指导的程度。在拐点之前的学习过程可能令人沮丧，然而一旦你越过拐点，会发现难以置信变化。

蓝色线条——学习具体知识
红色线条——学习独立解决问题
两个线条相交的地方就是拐点。
在 Firehose，虽然我们会教你使用 Ruby 创建 Web 应用，编写测试等等，但我们的目的决不仅仅如。我们最重要的目标是让我们的学生越过学习编程的拐点，以使他们具有独立解决自己遇到的问题的力。我们的教学方法重在培养独立解决问题的能力，相比创建一系列应用，这种能力才是无价之宝。
<h2 id="接受指导阶段-3-8个星期正式学习编程-">接受指导阶段（3-8 个星期正式学习编程）</h2>
刚开始学习编程的时候，有很多知识需要了解。这些知识是特定领域的知识，例如如何用 Ruby 一个循环或者使用 Ruby on Rails 从数据库中导出一些数据。特定领域相关的知识就是一个特定的编环境下所需要了解的内容。
成为自力更生的开发者的第一步，是学会如何完成具体的某一项任务。一旦你能够编写代码完成些具体任务，这些代码片段如何进一步在更大的范畴内组合，对你来说就会变得更加明晰。随着时间移，你会认识到设计模式的重要性，那些一开始陌生和令人困惑的东西，最终也变得易懂易用。
对于刚开始的学生，最重要的技能是注重细节
在阅读文档和教程的时候，注重细节是非常重要的。即使是微不足道的输入和拼写错误都会引发序出错。一开始，查看错误信息是令人沮丧的，但却是学习编程必不可少的步骤。在初学阶段，对于误信息和问题的处理会教给你在稳定环境下编程最重要的技能：注重细节。
根据错误信息进行 Debug 非常重要，因为处理错误信息就是编程的一部，编程新手和老手同样都要面对错误信息。唯一的区别是，处理错误信息的经验越丰富，花在修复问上的时间就越少，这是因为：
随着经验不断丰富，你会学习到如何通过阅读错误信息快速定位到问题的细节。第一次你看到某错误信息，可能要花好久才能发现错误所在。但当你看过成百上千条错误信息之后（你必定会看到成上千条），你能够立刻定位问题并了解修复错误的细节。
你应该从你解决的每条错误信息中汲取经验。可别仅仅只是修复错误，更要的是理解为什么会出错。通过从每个错误中汲取经验，下一次面对类似的问题时，你可以更加高效处理问题。
一开始遇到错误信息，你可能需要求助于外界。随着时间推移，你会先再次检查你的代码，或者效的使用 Google 搜索引擎来解决问题，这样你搬救兵的频率会大大下降。
在接受指导阶段，你会接受一系列的具体教学指引。一开始，你可能会发现跟上教学的进度并不么容易，错误信息此起彼伏。随着时间推移，你锻炼出了 debug 的能力，并且逐渐注重细节，这个候你跟上进度就变得更加容易。当你完成了整个接受指导的阶段，编写代码的速度会大幅上升。
在这个时候，一些人觉得非常自信——好像他们已经不再需要接受系统性的指导并试着造一些东——最后开心的掉入深渊而不自知。另外一些人会读更多的教程，试图获取更多的特定领域知识，以到“完全理解”的程度。不幸的是，教学指导只能带你走到这里，仅靠教程和指导无法获得真正的自。真正的自信来自于面对一个你无法解决的问题，经过一番挣扎之后最终独立解决问题的过程。
关于编程一个不可告人的小秘密是....
你永远不会知道解决问题所需要知道的一切。你也许认为继续编程之旅，总有一天你会学到一切需要习的东西，然后觉得编程索然无味。这一天永远不会到来。
编程是一种终身学习体验。经验丰富的软件工程师对于他们尚未解决的问题孜孜不倦地寻求解决

案，是因为这能让他们有机会学到更多的东西。如果你期望某一天你会了解编程的一切，记住，那一永远不会到来。这不是一件坏事。
大师失败的次数比新手尝试过的次数都多。
当你做好如下的准备时，就可以开始下一阶段了：你对错误信息已经习以为常，因为你看过太多错误信息了，你可以迅速解码出错误信息的意义，并且找到代码中出问题的地方
你已经是通过 Google 搜索解决问题的高手。当你要增加一个新功能或者遇到了一个令人困扰的误信息，你知道如何去搜索所需信息。
你能够参考你在应用的其他部分编写的代码并遵循设计模式去解决问题，而不总是寻求一步一步说明。
<h2 id="拐点阶段-良好心态下所需大概2-4个星期-">拐点阶段（良好心态下所需大概 2-4 个星期）<h2>
拐点阶段是学习编程中最消磨人意志的阶段，从各种角度来说，也是唯一重要的阶段。当你不再靠教程，开始解决没人提供解决方案的问题时，这就是拐点阶段。
有时你会觉得你还没有准备好面对这个阶段，可能转而回去根据详尽的说明去编写程序。不要成这种心态的牺牲品。你觉得最受打击的原因是：
在拐点阶段，你编程的速度可能只有接受指导阶段的十分之一到二十分之一这时你会产生疑问：是否能成为一个程序员。
在拐点阶段产生不安和疑惑是正常的。尽管你发现学习新东西和敲代码的速度大幅下降，但实际，你正在完成最重要的事情。当你在特定领域的知识足够丰富的时候，你所学的东西实际上是过程性识（https://en.wikipedia.org/wiki/Prcedural_knowledge）。
过程性知识指的是教会自己原本不知道的东西的能力。当你需要实现一个新功能的时候，你要到oogle 上搜索什么关键字？当你想要做很多事情的时候，你会感觉自己身处一片漆黑之中。学会依靠己在黑暗中寻找光明是非常关键的，因为你永远不可能了解所有东西，所以你必须教会自己如何解决头的问题。
大部分人没有意识到，学习编程需要同时学习特定领域知识与过程性知识。
<h2 id="在人生的每一天里探索自我边界以外的东西">在人生的每一天里探索自我边界以外的东西</2>
很多软件工程师一但找到自己的立足点，就停留在自己的舒适区内。这种程序员被叫做维护程序——显然不是你应该成为的类型。相反，每一天你都应该试着做超越自我的事情。大部分程序员辞职原因是：我已经解决了所有有趣的问题，再也没什么挑战了。
相比将代码和项目拉入到你的舒适区，更应该去解决那些超出你的技能范围之外的问题，这也是得和拓展新技能的唯一方法。
下边是一个越过了拐点的学员的感想：
我依然感觉到身处深渊，但我感觉棒极了，因为这就是我要呆的地方。
<h2 id="在Web开发中-实际上有两个拐点一同出现">在 Web 开发中，实际上有两个拐点一同出现<h2>
第一个拐点：Web 开发拐点，出现在你能够创建任何基于数据库驱动的应用时。这意味着你有力创建一个拥有许多页面，从一个简单的数据库进行存取数据的 Web 应用。Web 开发者管这个叫做“精通增删改查”。在这个阶段，你还可以通过阅读文档，Github 或者技术博客，将一个第三方库比如 Ruby Gem）集成到自己的 Web 应用中。
第二个拐点：算法与数据结构拐点，是一个不那么明显的拐点，但实际上更加重要。已经征服了个拐点的人，会精通他们所用的编程语言，了解编程的基础体系，拥有面对复杂问题的深厚知识储备越过算法与数据结构拐点的人通常可以：

写排序算法
实现并反转链表
理解和编写使用栈、队列、树的程序
使用递归和迭代思想编写程序

简要的说，一旦你越过了这个拐点，你就掌握了如何操作数据，并且了解代码的性能影响。大学传统的计算机科学教育特别注重让学生越过算法与数据结构拐点。但很多大学都使用在业界已经不流的语言来教授这一点，比如 Scheme，Racket 和 LISP。

在大部分技术面试中，面试官都假设面试者已经越过了 Web 开发拐点，因为这个拐点相对容易所以会将问题集中在评估面试者的算法和数据结构能力上。通常面试问题会集中在上边提到的几个方：排序算法，反转链表，使用栈、队列和树。
一旦开发者同时越过了 Web 开发拐点和算法与数据结构拐点，他就掌握了整个世界。
这些开发者能够解决两者兼备的挑战：在复杂的高级 Web 应用内构建复杂的算法。这是专业的eb 开发者每天都要做的事情。
<h2 id="征服拐点之后">征服拐点之后</h2>
征服拐点之后，你将明白的东西听上去有点违反直觉。深呼吸，然后：
对于学习编程，特定领域的知识在宏观角度并不重要。
我并没有开玩笑，特定领域的知识确实没有那么重要。
一旦你通过拐点，你会在一到两个星期，甚至几天内理解这句话。
真正重要的东西是：
对于一个 Web 框架牢固的掌握在任何编程语言中编写复杂算法的代码
招聘经理希望开发者同时具备 Web 开发技能和算法技能我在 Paypal 工作时，我的团队招进来个高级 Ruby Rails 开发工程师，但是之前他没有任何 Rails 经验，只有 Python，LISP 和 Perl 经验仅仅在几天之内，他已经给我们带来很大震撼。几星期之后是巨大的震撼。他迅速被提升为技术团队领导，也是我当时最成功的招聘决定之一。
不要盲目追求热点，许多人说：“最近 Angular JS 好热门”，“JavaScript 上升势头猛烈”，最近流行用...”。我对此的回复是：“那么又如何呢？”。当你开始学习编程时，你唯一的目标是找拐点然后越过拐点，当你成功之后，学习那些新的，好玩的东西毫不困难。
自力更生。拥有无需结构化的指导，就可以学习新编程技能的能力，意味着你无需等待别人帮助。对于你所需要学习的大部分知识，你可以搜索互联网或者阅读各种材料。
这并不意味着你可以迅速“了解”所有东西，而是意味着现在所有的东西都是“可以去了解”的实际上，你将所向披靡。
<h2 id="在拐点时你将练就的技能">在拐点时你将练就的技能</h2>
作为一个软件开发者，最好的参考资料，就是你编写过的类似代码。你完全理解了你所编写的代之后，不必将所有的细节都记住。当你开发一个新功能的时候，首先问自己：“我之前做过类似的东吗？”如果答案是 Yes，看一下以前写过的类似代码，在脑子里一行行的重放代码，重新解释给自己，然后问自己：“我现在可以使用相同的方式解决问题吗？”
视频并不是学习特定领域知识的好材料，因为视频通常耗费大量观看时间。比如你想集成 Google Maps API，你亲自动手，在 Github 上找到相关代码，复制到一个新建的项目，可能花费不到一分。而通过视频指导，看一遍可能需要 10-30 分钟。
<h2 id="高效率征服拐点的一些策略">高效率征服拐点的一些策略</h2>
由于征服拐点是学习编程中最重要的事情，所以你必须让自己尽可能顺利的越过这个阶段。这意着你必须在接受指导阶段就开始进行准备，并在整个过程中保持正确的心态。
在接受指导阶段，除了学习结构性的指导材料，还要全程给自己一些挑战。
对于每一课，尝试做超过教学范围的事情。如果书上有“挑战”或者“自己努力完成”的问题，它们全部做掉。解决没有指导的问题会带来无需结构化指导便能解决问题的重要经验。
尝试尽可能少的依赖教学材料。在 Firehose，我们通常让学生通过文档了解如何集成一些 gems 或者实现一些代码。对学生来说，他们主要根据程序文档操作，教学材料仅作为备查，而不是单纯的赖教学材料按部就班的学习。注意，文档会将阅读者视为已经跨越了拐点的开发者。习惯了在 Github 上阅读文档，对你自力更生是莫大的帮助。
关注要点与可复用性。从头开始编写应用程序，提交一个新应用到 Github 或者 Heroku，尽早建数据库迁移。
越过拐点可能具有挑战性，这里有一些建议值得尝试：
了解越过拐点是一个困难的过程，别给自己太大压力，并且设定现实的目标。你不能将你在接受导阶段的“超人”一般敲代码速度和这个阶段的蜗牛速度进行比较。要记得你一直在学习很多东西，这个阶段，你正在学习的是可以自己搞定全新事物的能力。
如果你自信心略有不足，要知道你的感受是完全正常的。继续努力，如果你依然很挣扎，试着和经越过拐点的人交流一下，也许他们会了解你现在所处的位置，并且告诉你这种感受只是暂时的。持努力，但不要过度劳累。在学习编程的这个阶段，你每天最多只能有效率的工作 6 个小时。在疲劳的态下学习只会延长你跨越拐点的时间。
这个阶段获取信心的最佳方式是解决自己遇到的疑难问题。你的情绪可能会像过山车一样。有时你觉得你自己好像被放在火上烤，但经过在一个问题上 15 小时的努力之后，你可能有完全相反的感

。
如果你在某件事情上花费了 5 分钟到 5 个小时，依然没有头绪，确实很令人沮丧，但每次你成搞定一个问题，实现一个功能，那种自信源源不断涌现的感觉是你最需要的。在不依靠帮助解决了很困难的问题之后，你会沉醉于在你的舒适区之外不断解决问题的感觉。
<h2 id="如何知道你已经跨越了拐点">如何知道你已经跨越了拐点</h2>
拐点过程的最后阶段是接受。接受软件开发是一个持续学习的过程。如果你感觉已经学习了所有西，只意味着你应该想一想如何解决更复杂的问题。（完）