拆解

在学习编程的过程中，你会不由自主地学会一个重要技能：

> 拆解

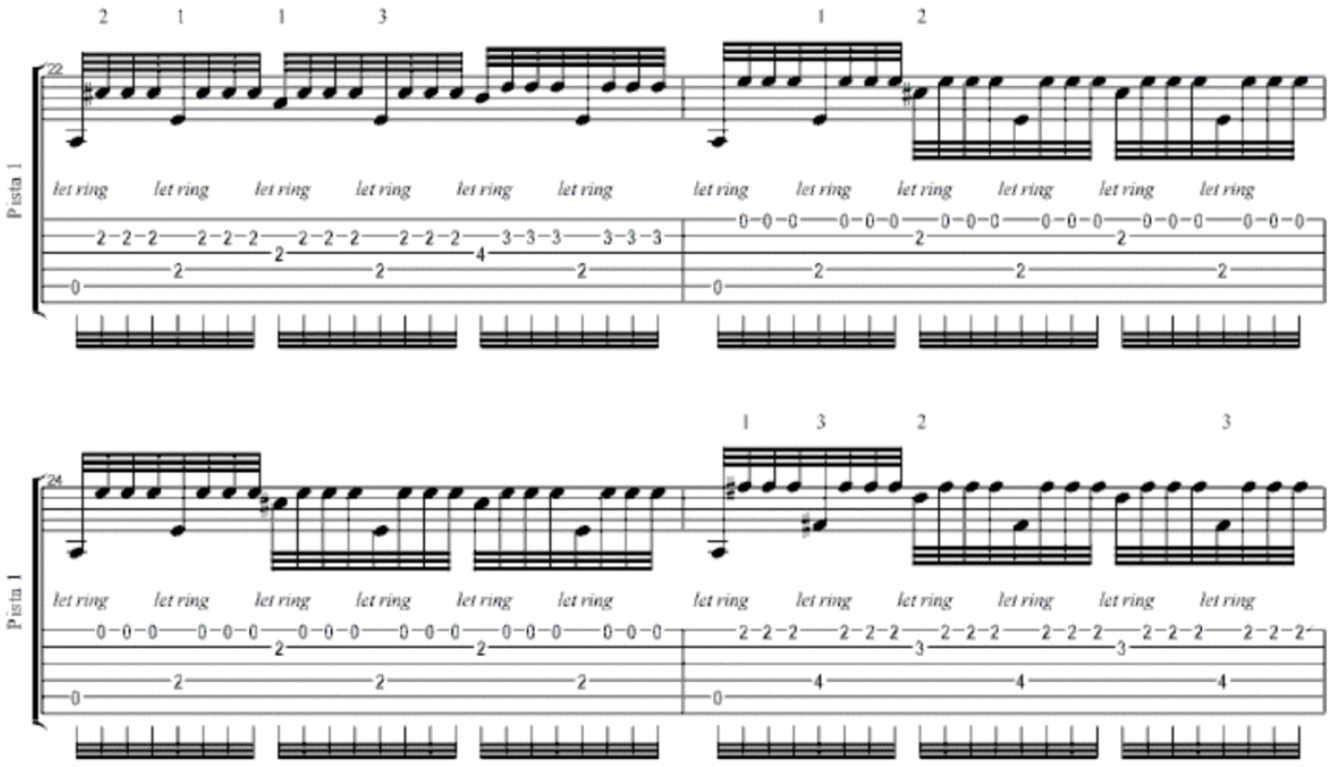
这么简单的两个字，在人生中的作用重大到不可想像..... 而且它也的确是自学能力中最重要的底层能力之一。

横向拆解

我很幸运，12 岁的时候有机会学习编程（习得了最基本的概念，那时候学的编程语言是 BASIC），所以，相对其他人在 “拆解任务” 方面有更强的初始意识。

后来，15 岁开始学着玩吉他的时候，发现道理其实是一样的。

有个曲子很难（当然也非常好听），曲名是 Recrerdes Da La Alhambra 阿罕布拉宫的回忆。你看看曲谱就知道它多难了：



那怎么办？怎么办？！—— 我的办法听起来看起来都很笨：

> - 每次只弹一个小节；
>   - 而且还是放慢速度弹，刚开始很慢很慢；
>   - 等熟悉了之后逐渐快起来，直到正常速度；
> - 再开始弹下一个小节；
>   - 同样是放慢速度弹，刚开始很慢很慢；
>   - 等熟悉了之后逐渐快起来，直到正常速度；
> - 再把两个小节拼起来；
>   - 有些小节拼起来相对容易，另外一些需要挣扎很久才顺畅；

如此这般，最终就把这个很难的曲子弹出来了 —— 其实所有的初学者都是这么干的。

可以听听这个曲子放松一下（当然肯定不是我弹的哈哈）：

```
from IPython.display import IFrame

IFrame('https://www.youtube.com/embed/OOsRMECWKAE?', width='800', height='450')
```

```
    <iframe
        width="800"
        height="450"
        src="https://www.youtube.com/embed/OOsRMECWKAE?"
        frameborder="0"
        allowfullscreen
    ></iframe>
```

提起这事，总是会不由自主地叹口气 —— 因为在这事上我运气太差，刚把这个曲子练完没多久，还没来得及找人显摆，就摔断了掌骨和指骨，给我的手指灵活性造成了不可修复的损伤，于是，后来只能用拨片玩玩吉他了……

话说回来，自学的一个重要技巧就是，

> 把那些很难的任务无限拆分 —— 直至每个子任务都很小，小到都**可操作**为止。

比如，正则表达式，这个你必须学会的东西，学会学好真的不那么容易。一切的技能都一样，没学会之前都是很难很难的，可学会之后用熟了，你就会“发现”那东西其实也没多难……

那刚开始的时候怎么办？你其实需要运用拆分的本领：

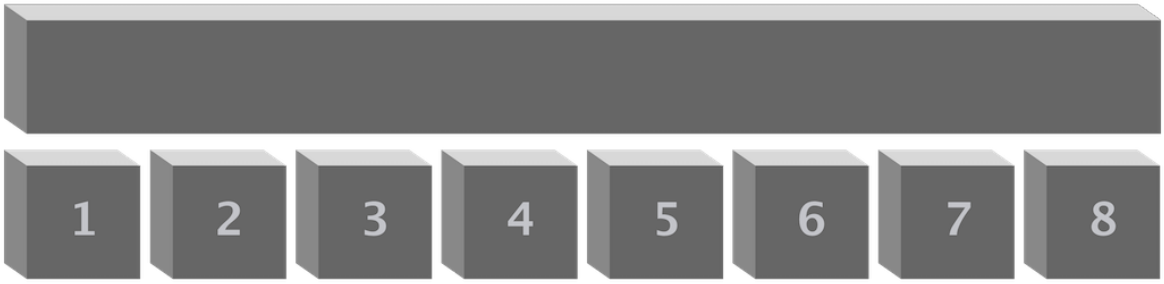
> - 先囫囵吞枣至少读一遍教程；
> - 给自己搭好测试的环境（或许在 Regex101.com 上，或许用一个编辑器，比如 VS Code）；
> - 先不管什么意思，找一些 Regex 自己试试；
> - 正式进入“精度”状态，每一小节每一小节地突破；
> - 搞定一小节之后，就把它与之前的小节再反复翻两三遍；
> - 把学习任务拆分成若干块，再重新逐个突破，比如，匹配，替换，在编辑器中使用，在 Python 代码中使用；
> - 把各种操作符与特殊字符拆分成若干个组，而后，熟悉到牢记（而不用将来反复回来查询）；
> - ……

事实上，当你习惯这么做了之后，就会“发现”一切的自学任务，其实都不是“难”，不过是“繁杂程度不一”而已。

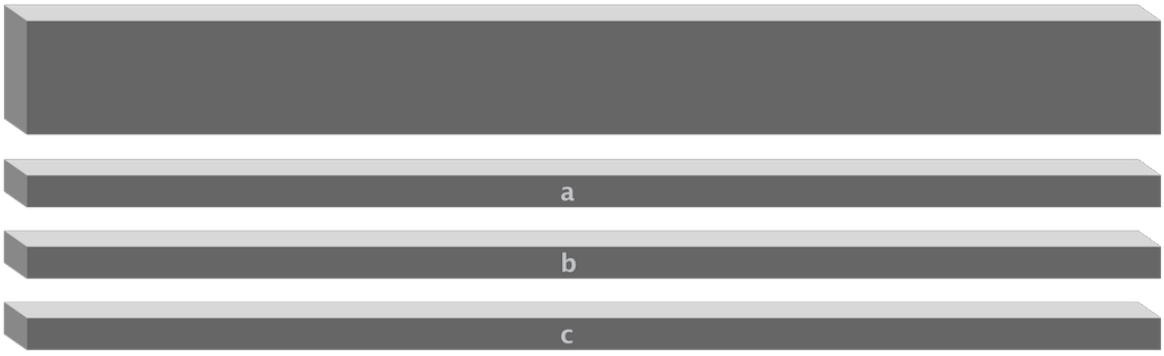
很多人最终自学失败，要么是不懂拆分任务，要么就仅仅是**怕麻烦**而已 —— 还是那句话，人活着就挺麻烦的……

纵向拆解

拆解的第一种方法是把某个任务拆分成若干个小任务，正如上面的讲解那样，我称它为“横向拆解”。



另外一种方法，我称它为“纵向拆解”（有时，我也会用“分层拆解”这个说法）。



这种方式在自学复杂的概念体系时特别管用。

编程这种东西，之所以刚开始觉得难学，是因为其中涉及的概念除了之前我们强调的“过早引用”之外，还有个特征：

> 有的概念虽然并不同属一个层面，却常常纠缠在一起，没有明确的提示.....

比如，常量、变量、字符串、函数、文件等等的概念，其实并不是某个特定的编程语言的专有概念，它们是所有编程语言都会涉及到的概念，因为计算机处理的就是这些东西，而无论哪个编程语言最终都要通过计算机去处理这些东西。

又比如说，分支与循环，每个编程语言都有对应的语句，所以，分支循环在逻辑判断、流程管理这个层面，而分支循环的“**实现**”应该划分到另外一个层面中去；而每个语言实现分支循环语句的语法多少有点差异 —— 这些细节属于那个编程语言本身：

```
# Python 这么写：

for i in range(n):
    ...
```

```
// JavaScript 这么写：

var i;
```

```
for (i = 0; i < n; i++) {
  ...;
}
```

自学正则表达式的时候也如此。最基本的规则是属于 Regex 自己的；而后各种语言的实现各不相同，那是各个编程语言层面的；在各种编辑器中，除了基础的规则之外，也有它们自己的定制…… 看起来细节很多，但分层分类之后，就会变得很容易理解、很容易记住。

遇到“面向对象编程”也是如此。类、实例、对象、继承、多态…… 这些其实并不属于某一个编程语言，但它们也确实在几乎所有编程语言中被实现、被应用 —— 所谓的难，无非是因为属于两个层面甚至多个层面的概念被拧在一起教、学、练……

再比如说，在我把这个用编程当作习得自学能力的第一个“实战项目”之时，甚至要把“读”和“写”分成两个层面，先照顾“读”，至于“写”，要等到有了基本的“读”的能力之后再说；即便是到了“写”，还要划分至少两个层面，首先是从“简单的函数”开始，而非上来就要写个“大程序”…… 这种拆分层面的技能好像可以用在方方面面呢！

所以，要在自学的过程中，不停地想办法把它们从层面上区分开来 —— 不能总是把它们混在一块儿“大锅烩”。

日常生活中，我们会遇到被评价为“理解能力强”的人，而另外那些不被如此评价的人就很不理解，很迷惑：

> 我到底差在哪儿了，你不说我理解能力强？难道我的理解能力很差吗？

当老师当久了，经常被这种现象震惊：

> 原来很简单的东西竟然可能成为很多人一生的障碍。

—— 并且，这话重复多少遍都不过分。

大多数人不太在意自己脑中的概念之间的关系，因为平日里这也不怎么耽误事。但一旦遇到复杂一点的知识体系，就完全搞不定了…… 而所谓知识体系的复杂，无非就是新的概念多一些，概念之间的关联更复杂一些…… 而概念之间的关联更复杂一些，无非是各个概念之间不仅只有一种联系，最后会形成网状连接……

—— 在《通往财富自由之路》那本书里，我几乎用了整本书的篇幅去讲解、厘清概念及其之间关系的重要性。

复杂吗？其实并不复杂 —— 在横向纵向分别逐步分清之后。

可问题在于，脑子里概念关联一团糟的人，自己并不觉得，甚至无法知道 —— 他们是那种跟你一块去看一场电影却能看到另外一部电影的人。说他们理解能力差过分吗？他们不能理解被评价为理解能力差，难道不是很自然吗？

分清概念的方法是什么？其实也不难，就是不断拆解，不断主动整理。每次用图表整理那些概念的时候，就会发现比原来更清晰一些，多次整理，最终就谙熟于心了。

想要再举更恰当更惊人的例子很难，勉为其难再举个例子。

当我在 2011 年遇到比特币的时候，现在回头看，在当时的情况下，我平日里习惯对概念及其关联进行各种纵向横向的拆解这件事给我“创造”了巨大的好运。后来我在《INBlockchain 的开源区块链投资原则》里提过这事：

> “比特币”这个概念，可以有多重的理解 —— 这也是为什么人们感到迷惑，或者相互之间很难达成一致理解的根本原因。
>
> 首先，比特币是世界上第一个，也是迄今为止最成功的区块链应用。
>
> 其次，比特币是一家世界银行，只不过它不属于任何权威管辖，它是由一个去中心化网络构成的。
>
> 另外，这家叫做比特币的，去中心化的世界银行，发行了一个货币，恰好也叫“比特币”。有些人更喜欢使用相对小心的说法，把这个货币指称为 BTC，而不是“比特币”（Bitcoin）。
>
> 最后，即便在比特币横空出世的七年后（2017），也很少有人意识到比特币（或者 BTC） 其实也可以被理解为这家叫做比特币的去中心化的世界银行的股票。

—— 这无非就是把一个概念拆分成若干个层面再对每个层面准确理解而已。

但毫无疑问，这点靠很简单很简单的方法练就的理解能力，帮了我大忙。

触类旁通

无论听起来多么简单的任务，落实成代码肯定没那么简单，没那么容易。

以后你会越来越清楚的：写程序的主要工作量，往往并非来自于在编辑器里敲代码那个阶段。

> 更多的工作量，其实在于如何才能在脑子里把整个流程拆解清楚，考虑到各个方面.....

所以，编程，更多是拿着纸笔梳理细节的工作。一旦所有的细节都想明白了，落实成代码其实是飞快的 —— 越是工程量大的项目越是如此。

这个道理在哪里都是相同、相通的。不说编程，说写书，也是一样的。

随着时间的推移，你花在“拆解”上的时间会越来越多，因为所有大的工程，都可以被拆解成小工程 —— 于是，也为了做出大工程，拆解的工作首先是必须，其次是最耗时费力但最值得的。

我身边很多人，包括出版社的专业编辑，都慨叹过我的“写书速度”。我猜，实际上把他们惊到甚至惊倒的，并不是他们以为的“李笑来写书的速度”，而是“李笑来打字的速度”而已。

当我告诉他们我要写一本什么书了的时候，实际上，有个工作早就完成了：“系统梳理要写的那本书的所有细节”，剩下的只是落笔把那些东西写出来而已 —— 当然，我是敲出来，用我那几乎无以伦比的输入速度敲出来 —— 那当然“显得”很快了！

创业也好，投资也罢，还是一样的。因为我这个人脸皮厚，不怕人们笑话，所以我可以平静地说这事：

> 我参与过（或投资）**很多**失败的创业项目.....

对所有复盘的结果，无一例外，根源都是当初立项的时候，很多重要细节还没搞清楚，甚至没想到要去搞清楚，就已经开始行动..... 于是，在成本不断积累的情况下，没完没了地处理各种“意外”，没完没了地重新制定目标，没完没了地拖延，没完没了地“重新启动”..... 直至开始苟延残喘，最后不了了之。

拆解得不够，就容易导致想不清楚，想错，想歪.....

也许，有人可能会理直气壮地反问，“怎么可能从一开始就把所有情况都想清楚么！”唉，是呀，以前我也是这么想的..... 直到吃了很多亏，很多很多亏，很多很多很大很大的亏，才“发现”且不得不痛下决心去接受：事先想不清楚的，就不要去做。

这是一种特殊、且重要、又极有价值的能力。现实生活中，后来我也见过若干有这种能力的高人，比如，你可以到网上搜一个人名，庄辰超，他就是我见过的能做到干什么事之前都能全都想清楚的真人活人之一。

自学的时候，拆解任务的重要性更是如此。

这本“书”的一个特点，就是把“自学”（或者平日里称为“学习”）这个流程，拆解为“学”、“练”、“用”、后面还会讲到“造”总计四个环节来处理 —— 从内容编排本身就这么干，甚至，在开头相当一部分，就明确说明，“根本不指望你读过一遍就会了”，还反复提醒，“要重复读很多遍，虽然第一遍必须囫囵吞枣”.....

对于初学者常面临的尴尬，我们也从一开始就提醒，编程语言，和你之前在学校里学的语文，本质上没什么区别，先学会读，而后在多读多读再多读的同时，开始练习写 —— 这才真的很自然。

即便是开始讲如何写，我们的做法也是“从写函数”开始，而不是“来，让我们写个程序.....” —— 这一点点看起来不起眼的差异，作用是很大的，因为从“小而完整”的东西开始做（任何事）非常重要。

“小”无所谓，“完整”才是关键。