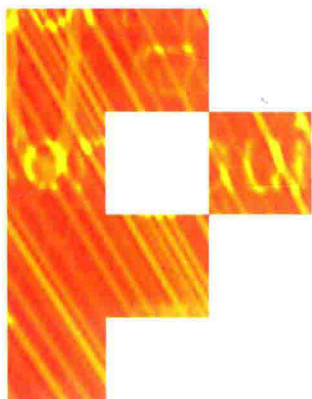
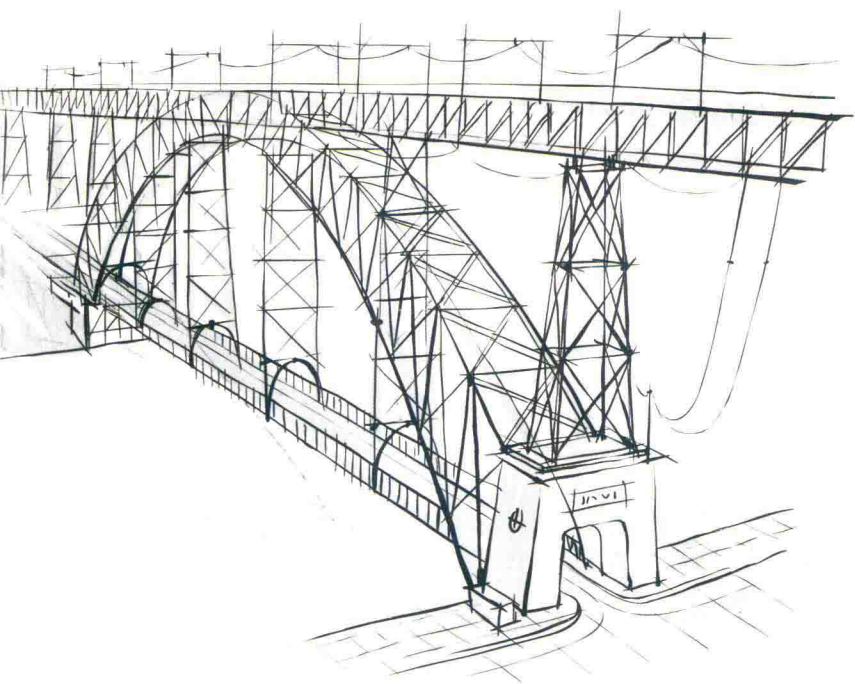




Python 设计模式

[美] 韦塞尔·巴登霍斯特(Wessel Badenhorst) 著

蒲成 译



清华大学出版社

Python 设计模式

[美] 韦塞尔·巴登霍斯特(Wessel Badenhorst) 著
蒲 成 译

清华大学出版社

北 京

Wessel Badenhorst

Practical Python Design Patterns: Pythonic Solutions to Common Problems

EISBN: 978-1-4842-2679-7

Original English language edition published by Apress Media. Copyright © 2017 by Wessel Badenhorst. Simplified Chinese-Language edition copyright © 2019 by Tsinghua University Press. All rights reserved.

本书中文简体字版由 Apress 出版公司授权清华大学出版社出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

北京市版权局著作权合同登记号 图字：01-2018-4393

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目(CIP)数据

Python 设计模式 / (美) 韦塞尔·巴登霍斯特(Wessel Badenhorst) 著；蒲成译. —北京：清华大学出版社，2019

书名原文：Practical Python Design Patterns: Pythonic Solutions to Common Problems

ISBN 978-7-302-51645-3

I. ①P… II. ①韦… ②蒲… III. ①软件工具—程序设计 IV. ①TP311.561

中国版本图书馆 CIP 数据核字(2018)第 257364 号

责任编辑：王 军 于 平

封面设计：周晓亮

版式设计：思创景点

责任校对：成凤进

责任印制：刘海龙

出版发行：清华大学出版社

网 址：<http://www.tup.com.cn>, <http://www.wqbook.com>

地 址：北京清华大学学研大厦 A 座 邮 编：100084

社 总 机：010-62770175 邮 购：010-62786544

投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈：010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者：三河市吉祥印务有限公司

经 销：全国新华书店

开 本：170mm×240mm 印 张：17.25 字 数：348 千字

版 次：2019 年 1 月第 1 版 印 次：2019 年 1 月第 1 次印刷

定 价：98.00 元

产品编号：080061-01

译者序

近年来，由于人工智能的兴起，Python 逐渐从科学计算领域进入普通大众的视野，其使用率一直呈线性增长。Python 在设计上坚持了清晰划一的风格，这使得它成为一门易读、易维护，并受到大量用户欢迎、用途广泛的语言。Python 的设计哲学是优雅、明确、简单，追求“用一种方法，最好是只有一种方法来做一件事”。有鉴于此，设计模式与 Python 的结合就变得顺理成章了。

设计模式是经过总结、优化的，可用于解决我们经常会碰到的一些编程问题的可重用解决方案。一个设计模式并不像一个类或一个库那样能够直接作用于我们的代码。设计模式更高级，它是一种必须在特定情形下实现的方法模板。设计模式不会绑定具体的编程语言。一个好的设计模式应该能够用大部分编程语言实现。最重要的是，设计模式也是一把双刃剑，如果设计模式被用在不恰当的情形下将造成灾难，进而带来无穷的麻烦。然而，如果设计模式在正确的时间被用在正确的地方，那么它将是我们的救星。

本书中的每一章都是以一个引用句子作为开头的，如果读者能明白其背景，那么一定会对本书作者的这一小情趣报以会心一笑，谁说钻研技术的人就毫无情趣了？译者对于这些短句子也是详细了解了其来源和背景，并且对作者在相应章节中放入对应句子的意义深以为然。为了让读者享受到这一乐趣，在翻译本书的过程中，译者并没有给出这些句子的详细出处和历史背景，这些内容就留待读者自行探究吧。毕竟，探索的过程，其本身的意义要远远大于平铺直叙的结果所能带来的影响。

本书从设计模式的角度，结合各个示例深入浅出地讲解了各种设计模式在 Python 中的实现，为读者描述了 Python 式设计模式的应有结构以及应该避免的误区。所以本书的高度较一般的 Python 开发类书籍要高，同时也要求读者具备一定的软件开发和架构经验，否则就可能无法很好地理解本书中讲解的内容。相信在阅读完本书之后，读者对于在 Python 中实现各种设计模式的处理一定会深有感悟。正如作者在书中所言，作为程序员，我们应该追求持续不断的进步。学无止境，大家共勉！

在此要特别感谢清华大学出版社的编辑，在本书翻译过程中他们提供了颇有助益的帮助，没有他们的热情付出，本书将难以顺利付梓。

本书主要章节由蒲成翻译，参与翻译的还有何东武、李鹏、李文强、林超、刘洋洋、茆永锋、潘丽臣和王滨。由于译者水平有限，难免会出现一些错误或翻译不准确的地方，如果有读者能够指出并勘正，译者将不胜感激。

译者

作者简介



Wessel Badenhorst 非常热衷于研究获取专家级能力的过程，尤其是编程领域的专家级能力。他获得了计算机科学的学士学位，并且在真实的应用场景中积累了大量经验。

技术编辑简介

作为个人独立贡献者、团队领导、项目经理以及工程副总裁，Michael Thomas 在软件开发领域已有超过 20 年的从业经验，他在移动设备方面的开发经验超过了 10 年。他目前专注于医疗领域，使用移动设备加速患者与健康医疗服务提供商之间的信息传递。

致 谢

非常感谢 Apress 的 Mark Powers 及其团队，没有他们，本书将无法面世。谢谢 Tanya、Lente 和 Philip，是你们让我在稍有闲暇时编著了本书。感谢读者积极追求成为更优秀的程序员，从而让这一职业变得更美好。

目 录

第 1 章 前述	1
1.1 大师	1
1.2 成为更优秀的程序员	3
1.2.1 刻意训练	4
1.2.2 单一关注点	4
1.2.3 快速反馈	5
1.2.4 自我拓展	5
1.2.5 站在巨人的肩膀上	6
1.2.6 如何入手	6
1.2.7 保持路线正确的能力	7
1.3 系统化思考	8
1.4 思维模型	8
1.5 这项任务的适用工具	9
1.6 设计模式的概念	9
1.6.1 是什么造就了设计模式	10
1.6.2 分类	10
1.7 将用到的工具	11
1.8 本书的阅读方式	12
1.9 配置 Python 环境	12
1.9.1 Linux 环境	12
1.9.2 Mac 环境	13
1.9.3 Windows 环境	14
1.9.4 VirtualEnv	15
1.10 编辑器	16
1.10.1 Atom	16
1.10.2 LightTable	16
1.10.3 PyCharm	16
1.10.4 Vim	17
1.10.5 Emacs	17

1.10.6 Sublime Text	17
1.11 本章小结	18
第 2 章 单例模式	19
2.1 问题	19
2.2 进入对象	23
2.3 整理	26
2.4 练习	29
第 3 章 原型模式	31
3.1 问题	31
3.2 迈出第一步之后	31
3.3 一个真实游戏的基础	32
3.4 实现原型模式	41
3.5 浅拷贝与深拷贝的对比	42
3.6 浅拷贝	43
3.7 处理嵌套结构	43
3.8 深拷贝	44
3.9 将所学知识用在项目中	45
3.10 练习	48
第 4 章 工厂模式	49
4.1 准备开始	49
4.2 游戏循环	50
4.3 工厂方法	54
4.4 抽象工厂	56
4.5 本章小结	57
4.6 练习	58
第 5 章 建造者模式	59
5.1 反模式	65
5.2 练习	70

第 6 章 适配器模式	71	第 10 章 责任链模式	113
6.1 不要重复自己(DRY)	72	10.1 安装一台 WSGI 服务器	114
6.2 关注点分离	73	10.2 身份验证头信息	115
6.3 样本问题	75	10.3 责任链模式	119
6.3.1 类适配器	76	10.4 在项目中实现责任链	121
6.3.2 对象适配器模式	77	10.5 一种更趋 Python 化的 实现	124
6.3.3 鸭子类型	77	10.6 结束语	128
6.4 在现实环境中实现适配器 模式	78	10.7 练习	129
6.5 结束语	80	第 11 章 命令模式	131
6.6 练习	81	11.1 控制 turtle	131
第 7 章 装饰器模式	83	11.2 命令模式	133
7.1 装饰器模式	85	11.3 结束语	138
7.1.1 闭包	89	11.4 练习	139
7.1.2 保留函数的 <code>__name__</code> 和 <code>__doc__</code> 属性	89	第 12 章 解释器模式	141
7.2 装饰类	92	12.1 特定领域语言	141
7.3 结束语	94	12.1.1 DSL 的优势	144
7.4 练习	95	12.1.2 DSL 的劣势	144
第 8 章 外观模式	97	12.1.3 组合模式	148
8.1 POS 示例	97	12.1.4 使用组合模式的内部 DSL 实现	149
8.2 系统演进	100	12.2 实现解释器模式	152
8.3 使外观模式凸显出来的 原因	101	12.3 结束语	157
8.4 结束语	103	12.4 练习	158
8.5 练习	104	第 13 章 迭代器模式	159
第 9 章 代理模式	105	13.1 迭代器模式的 Python 内部 实现	161
9.1 记忆法	105	13.1.1 Itertools	166
9.2 代理模式	107	13.1.2 生成器函数	167
9.2.1 远程代理	109	13.1.3 生成器表达式	169
9.2.2 虚拟代理	110	13.2 结束语	169
9.2.3 保护代理	110	13.3 练习	170
9.3 结束语	110		
9.4 练习	110		

第 14 章 观察者模式	171	18.2 结束语	226
14.1 结束语	183	18.3 练习	227
14.2 练习	183	第 19 章 模型-视图-控制器模式	229
第 15 章 状态模式	185	19.1 模型-视图-控制器概述	232
15.1 状态模式	187	19.1.1 控制器	234
15.2 结束语	191	19.1.2 模型	234
15.3 练习	191	19.1.3 视图	235
第 16 章 策略模式	193	19.1.4 总述	235
16.1 结束语	197	19.2 结束语	239
16.2 练习	197	19.3 练习	240
第 17 章 模板方法模式	199	第 20 章 发布-订阅模式	241
17.1 结束语	207	20.1 分布式消息发送器	246
17.2 练习	208	20.2 结束语	248
第 18 章 访问者模式	209	20.3 练习	249
18.1 访问者模式	218	附录 设计模式快速参考	251

前 述

设计模式有助于我们借鉴其他人成功的经验，而不是靠自己摸爬滚打。

——Mark Johnson

这个世界正在快速变化。

读者在阅读这段文字时，世界各地的人们正在努力学习编程，但大多数人方法笨拙。市场正面临着如潮水般的“程序员”的涌入，似乎没有什么可以阻止这种情况的发生。和其他事物的发展一样，不久大家就会看到，随着程序员的人数逐渐增多，雇用程序员的成本将逐渐减少。能够简单地编写几行代码或者修改一个软件中的某些东西将变成一种基本技能。

如果读者计划以程序员这一职业作为谋生手段的话，互联网会让这个问题变得更加糟糕。我们不仅要担心来自自身所处领域的程序员的竞争，还要面临来自世界各地的程序员的竞争。

想不想知道谁不用担心由数百个编程新兵训练营所培养的数以百万计的程序员？

1.1 大师

尽管每个人都会写字，但世上仍旧会出现海明威似的人物。尽管每个人都能使用 Excel，但仍旧会有财务建模师。同理，当每个人都会写代码时，世上也仍旧会有 Peter Norvigs^①这样的专家。

程序是一款工具。曾经有一段时间，只要知道如何使用这款工具就能让我们变得有价值，不过那段时期已结束。现在我们面临着新的现实情况，也就是许多

^① Peter Norvigs 是人工智能专家，是 *Paradigms of AI Programming* 和 *Artificial Intelligence: A Modern Approach* 的作者/合著者。在本书写作时任 Google 研发部总监。

行业在过去数年中不得不面对的问题。当一款新工具出现时，没人想要使用它。然后，某些人看到其价值，因而也就使得他们要比那些不会使用该工具的人强。之后，这款工具就会变得流行起来。互联网时代已经悄无声息地来临，每个人都能使用它。突然之间，能够创建一个网站变得不那么有价值了，所有从事咨询行业，依靠咨询服务赚取大笔利润的企业都被边缘化了。不过，无论市场如何变化，花时间掌握工具的人都能够抓住发展的机遇。之所以如此，是因为他们能够在每个层面都表现出众——工作质量、开发速度，以及最终成果的完美程度。

目前，我们可以看到对易于使用工具的普遍利用，接下来就会对容易的事情进行自动化处理。在生活的方方面面，简单的任务之所以简单，是因为它们不需要创造力和深刻的理解，而这些特性所带来的结果就是，这些任务正好就是最早转交给计算机处理的任務。

读者捧起这本书的原因就是希望变成一名更优秀的程序员。大家都不希望止步于数以千计这样那样的说明介绍。我们已准备好迈出下一步，帮助读者掌握其精髓。

在编程领域，我们希望能够解决重大且复杂的问题。为此，需要能够在较高的层次上进行思考。就像国际象棋的国际棋联大师们一样，相较于普通的象棋大师而言，他们能够在更大型的组块联合体(chunk)中领悟，我们也需要开发自己在更大组块联合体中解决问题的能力。当能够查看一个全新的问题并且快速将其解构成高层次的组成部分，并且这些组成部分是我们之前就已经解决过了时，我们就成为经验丰富可以举一反三的程序员了。

在我开始编程时，我还读不懂说明手册。那本手册封面上印有《太空侵略者》的图片，并且该手册许诺教会我如何自行开发一个游戏。它有点忽悠人，因为最后编出来的游戏是一个 for 循环和一个 if 语句，而非《太空侵略者》。不过我已经沉迷其中了。学习热情驱使我学习我能接触到的关于编程的一切知识，不过当时并没有很多内容可供我学习。在掌握其基础之后，我有些停滞不前了，即便在以优异成绩获得计算机科学学士学位期间也是如此。我觉得我所学的不过是新瓶装旧酒的基础知识而已。我切实获悉的一点是，似乎每个人都在等待着什么，虽然这一过程让人感觉缓慢和沮丧。

在现实世界中，情形似乎没太多变化。我意识到，如果我希望成为经验丰富可以举一反三的程序员，就必须做一些不同的事情。一开始，我想过获得一个硕士学位或者博士学位，不过最后我决定靠自己的力量进行更深入的研究。

我重新阅读了计算机技术类书籍的旧理论，它们被赋予了新的含义和新的生命。我开始定期参与编程的工作挑战并且学习用更地道的方式编写代码。然后，我开始研究对于我本人而言完全陌生的编程语言和范式。在我有所意识之前，我

已经将自己对于编程的思维模式完全转变了。曾经很棘手的问题变得微不足道，而一些看似不可能的任务变得仅仅只是有些挑战性而已。

我仍旧在学习，仍旧在深入研究。

读者们也可以这样做。可以从目前所阅读的这本书开始。在本书中，大家会找到若干工具和思维模式，它们有助于大家更好地思考和编码。掌握这一组模式之后，就要积极主动地寻找和掌握不同的工具与技术。研究不同的框架和语言，并且弄明白其伟大的原因。一旦能够理解某些人为何会钟爱一门不同于我们自己经常所用的语言时，可能就会在其中发现几个可以整合到我们思考方式中的理念。无论何时发现诀窍和模式，都要加以学习，并且将其转化成可以反复使用的抽象方案。有时候，我们可以直接分析已经在用的解决方案：是否可以找到更优雅的方式来实现其概念，或者在某些方面是否缺乏该理念？要持续不断地询问自己，如何才能改进我们的工具、提升我们的技能。

我们将要探究一个真实场景上下文中的一组基础设计模式。与此同时，大家将开始意识到，在未来遇到特定类型问题时，这些设计模式可以如何被视作可供使用的构造块。

我希望大家将本书中的设计模式用作方案蓝图，它将有助于大家尽快开始收集自己的问题解决方案组块。在阅读完本书之后，大家应该可以顺利地继续迈向作为程序员的下一个阶段了。大家还应该看到，这组抽象工具如何被转换成我们可能遇到的任何编程语言，以及如何才能使用来自其他语言的理念和解决方案，并且将之纳入到我们自己的思维工具箱中。

1.2 成为更优秀的程序员

要成为更优秀的程序员，我们就需要下决心去提高自身在编程方面的精通程度。每一天都是让自己变得比前一天更优秀的新机会。每一行代码都是提升改进的机会。那么该如何做呢？

- 当必须处理一段已有代码时，在处理完成之后应该让它比之前变得更好。
- 尝试每天完成一个快速解决问题的挑战。
- 寻求与编程技术比自己好的程序员共事的机会(开源项目是最好的机会)。
- 专注于刻意训练。
- 无论何时可以找到理由，都要练习系统化思考。
- 收集和分析思维模型。
- 熟练掌握工具并且理解它们适用于何处以及不适用于何处。

1.2.1 刻意训练

Anders Erickson 博士研究过那些达到一定能力水平的我们称之为大师的人。他发现，这些人有一些共同点。首先，正如 Malcolm Gladwell 的著作 *Outliers* 中所普及的概念一样，这些人似乎都在其相关领域的技能上花费了大量的时间。实际的数字可能不太相同，但经过调研，这一时长大致接近 10 000 小时。这是很长的一段时间，但是仅花费 10 000 小时，或者说十年时间进行实践还是不够的。经证明，要达到大师级别的能力水平，需要一种非常特定类型的实践。获得能主导相关领域的技能的关键就是刻意训练。

我们马上就要介绍刻意训练与编程的关系，不过首先来看看刻意训练是什么。

刻意训练是缓慢的、时常停顿的、反流程的。如果读者正在练习小提琴，那么刻意训练就势必涉及非常缓慢地演奏曲子，以确保能够完美地演奏每一个音符。如果读者正在刻意地练习网球，那么这可能意味着与教练一起反复地练习击球，以便进行细微的调整，直到能够在该位置重复完美地击球为止。

知识型工作的问题在于，标准的刻意训练方案看起来都让人觉得很陌生。相对于编程所涉及的过程而言，体育项目或者演奏乐器都很简单。要解构设计和开发软件解决方案的过程是相当困难的。弄明白如何训练一种思维方式是很难的。之所以如此，其中一个原因在于，过去的已经过去了。这句话的意思是，作为开发人员，我们几乎很少会被要求编写一个已经写好的软件。找出可以反复训练的“击球机会”是很难的。

就编程而言，我们希望学习解决问题的不同方式。我们希望找到迫使我们应对具有不同约束条件的同类问题的各种挑战。在彻底理解问题出处以及一组可能的解决方案之前，要持续致力于此。

实质上，刻意训练具有以下这些组成部分：

- 每一次训练环节都具有单一关注点。
- 尝试和反馈之间的间距(时长)要尽可能短。
- 要着手处理自身还不能应对的事情。
- 遵循前人的脚步。

1.2.2 单一关注点

在一个具体的训练环节期间，我们仅仅希望专注于一件事情的原因在于，我们想要将所有的注意力都放在期望提升的元素上。我们希望竭尽所能来完美地处理该事情，即便它是一次性事务。然后，我们希望重复这一过程以便得到另一个完美的回应，然后继续重复进行。每一次训练执行都是对技能的单次强化。我们希望增强那些能得出完美结果的模式，而非那些产生不合意输出的模式。

在本书的内容背景中，我希望读者一次仅针对单个设计模式进行学习。我们不仅要切实地寻求理解某个模式是如何发挥作用的，还要理解一开始为何要使用它以及它是如何适用于我们手头的问题的。另外，要思考能够使用这一设计模式所解决的其他问题。接着，我们要尝试使用正在研究的模式来解决其中一些问题。

我们希望为问题以及解决该问题的模式创建一个思维工具箱。理想情况下，我们要培养出一种意识，也就是，这些问题何时适合放入我们已经熟练掌握的工具箱之一来解决，然后就是能够快速且容易地解决该问题。

1.2.3 快速反馈

刻意训练经常被忽略的一个方面就是快速反馈的周期。反馈越快，连接性就越强，我们也就会更容易地从中学习到经验。这也就是为何像市场营销和写作这样的事情如此难以被人熟练掌握的原因。将字母码在一页纸上和从市场上得到反馈之间的时间的确太长了，以致我们完全不能看到练习尝试的效果。对于编程而言，情况就不同了，我们可以编写一段代码，然后运行它以便立即得到反馈。这就使得我们可以正确前行并且最终得到一个可以接受的解决方案。如果更进一步并且为我们的代码编写完善的测试，那么我们甚至会得到更多的反馈，并且相对于必须在每次做出变更时手动测试该过程而言，这一方式能够更加快速地得出解决方案。

帮助我们更加快速地从过程中学习经验的另一个技巧是，预测希望编写的代码块的输出结果。记录下与我们为何期望这一特定输出有关的内容。现在编写这段代码并且针对期望结果进行结果检查。如果结果不匹配，则要尝试解释为何是这样的情况，以及如何用另一段代码来验证我们的解释。然后，对这另一段代码进行测试。我们要持续这一过程，直到达到熟练掌握的水平。

我们将发现自己得到了不同层次的反馈，并且每一个反馈都有其价值。

第一个层次会直接表明解决方案是否可行。接着，我们可能会开始思考像“该解决方案的实现会有多难？”“这个解决方案对于该问题而言是否真的合适？”这样的问题。之后，我们可能会寻求额外的反馈，反馈的形式包括：简单的代码复审，着手处理项目，与思维模式相同的人探讨解决方案。

1.2.4 自我拓展

我们回避的事情是什么？这些事情是就编程而言引发我们轻微不适的一些方面。它可能是从磁盘上读取一个文件，或者连接到一个远程 API。无论所回避的事情是一些图形库还是一种机器学习框架的安装配置，这都没什么两样，我们全都面临着会让我们感觉不舒服的编程领域的某些内容。这些事情通常就是我们最

需要着手应对的。这里有一些让我们进行拓展并且迫使我们面对自身缺陷和盲点的领域。克服这些不适感的唯一方式就是深入探究，以各种方式反复多次使用它，以便能开始获得一些经验体会。我们必须能够非常顺手地使用它，以便不再需要翻阅关于堆栈溢出的文件打开协议。实际上，我们已经编写了一个更好的协议。我们可以信手拈来般轻松使用 GUI 并且从数据库中提取出数据。

对于这一层次的精通程度而言，是没有捷径可走的。唯一的方式就是头悬梁锥刺股般地刻苦练习。而这也是为何成为真正大师的人如此之少的原因所在。到达这一层次，意味着要在并不容易处理、让我们深感挫败的事情上花费大量的时间。正是由于在这些自我否定的事情上花费了如此多的时间，因此就任何领域而言，几乎没有大师还会骄傲自大。

从头至尾地练习本书中的设计模式是找出潜在可发展领域的一种好的方式。只要从单例模式开始学习并且坚持下去即可。

1.2.5 站在巨人的肩膀上

有人在编程领域做出了非常惊人的成就。这些人通常会在开发者大会上演讲，有时候也会在网上露面。建议看看这些人都在谈论些什么。要尝试理解他们处理一个新颖问题的方式。在他们展示解决方案时，要一边听一边记。要跟上他们的思路并且弄明白他们这样做的动机。尝试以我们所认为的这些人解决问题的方式来解决一个问题，用这种方式得出的解决方案与用我们自己的方式得出的解决方案有何不同？

真正伟大的开发人员对于编程都充满激情，并且应该不用过多的推动力就能让他们讲解其技能的更多细节。寻找这类开发人员常常出现的用户群，并且要借机与他们进行大量探讨，要持开放态度并且进行持续学习。

选取那些迫使我们使用还未精通的设计模式的私人项目，享受其处理过程。最重要的是，要学会对这一过程充满激情，并且不要沉迷于一些认知上的结果，而是要花时间成为一名更优秀的程序员。

1.2.6 如何入手

以达·芬奇决定以绘画作为其职业时开始练习的相同方式入手。

也就是模仿！

这是正确的。首先要识别出一些有意义的问题，一个已经被解决的问题，然后坦然地效仿该解决方案。但千万不要复制/粘贴。要通过亲自动手输入的方式来复制该解决方案。让我们自己的复制版本能够生效。在此之后，则要完全删除它。接下来，尝试一边回想一边解决该问题，仅在记不清楚时参考一下原始的解决方