Ruby 再入门

Delton Ding

Version 0.1.0, 2019-02-08

Table of Contents

写在一切之前

为什么要写这本书?

市面上关于编程入门的书多到无法想象，但是其中令人满意的却很少。事实上，写一本面向初学者的书比写一本面向专家的书更困难。面向专家的书籍，最重要的是专业性和正确性，而面向初学者的书籍在此基础上还要加上通俗性。市面上的一些书籍，在通俗性和正确性上往往二者不能兼顾。

对于初学者来说，给予错误资讯的危险性是极大的。如果你今天拿到一份乐谱准备练琴，练了一个月发现，调号就没看对。等你再去纠正，形成了肌肉记忆的手再想纠正回来就必须付出更多的代价。我虽然对于 Ruby 编程也算颇有经验，但也绝不能说内容 100% 的正确性。我也不敢像 Donald E. Knuth 老先生那样为第一个发现错误的准备 2.56 美金，此后每发现一个翻倍，这样我离破产基本上是不远的。维护这本书正确性的核心，一方面是反复校稿，其次是维护了一套 CI 系统来自动化测试书中代码行为是否和描述一致，但更重要的就是秉持开放的态度，基于 GitHub 进行协作。欢迎大家自由为书籍纠错、 提意见。

至于通俗性，我觉得大家要有对编程难度的一个正确认知。如果你觉得编程对你很难，这里的「难」指的是「问题困难」还是「问题复杂」。「问题困难」往往是一些计算机科学学科强相关的内容，比如算法、数据结构、形式语言；而「问题复杂」常常是工程问题，里面的内容通常是你所熟知的，唯一的问题是你没有办法很好解构这个问题，将复杂的实际问题转换成一系列你熟悉的小问题。

我非常喜欢陶哲轩先生所写的《实分析》，因为他把一个「困难」的数学分析问题讲得很简单易懂，和他比起来我还差太远。好在这本书所涉及的问题，几乎都是后一类问题，很少会涉及前一类问题。所以大可不必妄自菲薄，你一定具备理解这本书内容的前置知识。同时，这本书会在每个章节后通过一系列复杂性逐渐上升的综合练习，让你可以慢慢解构问题，最终能够独当一面完成一个复杂的项目。

什么人适合读这本书?

- 想要了解编程的人
- 想要将编程作为自己工作而入门的人
- 想要学习 Ruby 编程语言的人
- 有一定编程基础，想尝试 Ruby 语言的人
- 有一定 Ruby 基础，想进一步理解 Ruby 的人

什么是 Ruby?

Ruby 是最初由 Matz（Yukihiro Matsumoto）于 1993
年开发，现在作为开源软件开发的语言。它可以在多个平台上运行，并在世界各地使用。尤其适合于网站的开发
。

为什么选择 Ruby?

因为我喜欢 Ruby。

Ruby
是一门适合用来作为入门教学的语言。使用这门语言不需要对计算机组成原理有着比较深刻的理解。使用这门语言进行教学，我们不但可以学习形如面向对象等在工程中常用的范式，还可以掌握形如 Lambda
演算、元编程这样非常 Lisp 的特性。站在功利的角度来说，Ruby 不是一门非常大众的语言，虽然可能岗位相比 Java、PHP 少很多，但是相对的竞争的求职者也更少。但是，编程的工具绝不是一招鲜吃遍天的，10
年前所流行的框架放到今天可能多半已经被淘汰了。如果单靠一门技术就想在这个行业活到退休是非常困难的，但是这些框架背后的思想确是共通的。

Ruby 的创始人 Matz 曾经说过：「Ruby
的首要设计目标是让这个世界上每一位程序员都变得充满生产力，享受编程，以及变得开心。」无论你最后会不会选择 Ruby 作为工作，都不妨碍我们用 Ruby 作为教学语言。

怎么阅读这本书?

我把这个书的设计理念定义为「在两座山脊之间寻找山谷」。这两座山脊分别是站在语言学习者角度语法使用的山脊，以及站在语言设计和实现角度对语言特性思考的山脊。

所谓「学而不思则罔，思而不学则殆」。如果只学习基本的语法使用，没有认清很多设计面向的场景，没有理解语言的思考方式，不但写出来的代码非常脏乱差，而且实际使用的时候往往不知从何下手。

这本书每个章节的开头会有一个标记 ○ 或者 ◎。○ 表示这个章节是在自上而下讨论一个使用方法，而 ◎
则是在自下向上讨论具体的实现和思路。如果你是第一次接触编程，我推荐阅读顺序是先把所有 ○
章节读完，再来读 ◎ 章节。如果你是有一定编程经验，甚至是已经掌握一定 Ruby 编程能力的，可以同时阅读 ○ 和 ◎ 章节。

除了读之外，一定要
练习。软件工程是工程学，是由大量的细节组合起来的工程。如果看完书中的内容，不上机练习，不多加尝试，不勤加思考，那么收益必然是零。因为会有大量的问题在没有尝试之前不会发现，等把书看完了，除了掌握了基

本概念，剩下的必然都还在云里雾里。

遇到问题怎么办?

如果你从「〇家」买了一套家具回家，发现不会安装怎么办？你必然是先要仔仔细细检查安装说明书，如果实在不行，再打电话给「〇家」，写代码也是这样。大家的时间通常都是有限而宝贵的，如果遇到什么鸡毛蒜皮的事情，都问别人，很可能会得到一句愤怒的 RTFM 作为回答。这是 Reading the F**king Manual 的缩写，意思是「去读＊＊的手册」。而且这也会影响到自己独立解决程序问题能力的养成，绝对不是好习惯。对于遇到程序问题，一般的解决流程是：

1. 检查开发手册。对于 Ruby 这一手册通常是 Ruby 的 官方文档，对于 Linux/UNIX 系统调用，手册可以通过 `man 调用名称` 和 `info 调用名称` 的系统命令来查询。
2. 使用搜索引擎查找问题。除非你在使用非常先进的技术，通常你不会是第一个遇到这个问题的。许多网站，例如 stack overflow 就大量收录了程序相关的问题。上网找找，有极大的概率有人会遇到和你同样的问题，并且已经有解决方案了。
3. 问问小黄鸭。很多看起来很奇怪的问题往往来源于粗心。放一只橡皮小黄鸭在桌子上，给它一行一行耐心解释你所写的代码，你可能会突然发现问题所在。
4. 询问行业专家或开发者。如果真的不幸全部都没有解决问题，确实有必要找个人问问。在网上问问题是个很好的途径，一方面你可以问到平时不容易接触到的领域专家，同时也可以留下记录帮助更多人。但是注意，一定要把自己问题的触发条件、相关的环境配置、代码、遇到问题的详情好好描述出来。如果就说一句「我做了 xxx，它不工作」，别人就算看到也无从下手。具体如何提问可以参考 stack overflow 的《How do I ask a good question》。

豆知识：`man` 命令和 `info` 命令有什么区别?

`man` 是在相当早年的 UNIX 系统中就已经内建在系统中了。而 `info` 则是 GNU 项目的文档系统。`info` 使用 Texinfo 作为其源文件的形式，提供了更加丰富的文本格式，通常内容也会比 `man` 来得更完整。但在长长的 `info` 页面中通过快捷键导航快速找到自己想要的东西也是一件需要熟悉的事情。

闲聊：如果我觉得 Manual 的缩写 `man` 命令冒犯到了我的性别平等主义怎么办?

如果你是用 zsh 作为自己的 shell，zsh 有非常有用的 `alias`
功能可以来解决这一问题。把这些东西加入到你的 zsh 配置文件中就好：

```
alias man="info"
alias woman="info"
alias lgbt="info"
alias lgbtq="info"
alias lgbtqi="info"
alias lgbtqia="info"
alias lgbtqiap="info"
alias lgbtqiapk="info"
```