Ruby 再入门

Delton Ding

Version 0.1.0, 2019-02-08

Table of Contents

简介

一本关于重新思考编程入门的教程。

如何编译这本书?

这本书使用 Asciidoc 标准编写。编译这本书需要有 Ruby 的环境。

使用 `bundle install` 安装工具，并使用 `bundle exec rake build` 进行编译。

编译完成的电子书在 `./build` 目录下。

如何给这本书提供意见/修改?

本书使用 Git 进行维护，并使用 GitHub 对 Git 仓库进行管理。请使用 GitHub 的 Issue 和 Pull Request 系统为本书提供意见和修改。

在线阅读

restartruby.com

下载

- PDF 版本
- Epub 版本

鸣谢

这本书的写作过程和发行过程离不开下面这些人的帮助，谢谢他们。

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。

本书按知识点的逻辑关系进行排序，因此难度上可能存在跳跃。在一些章节，本书会使用如下的提示来提醒初学者。

> 本章节涉及较为进阶内容，建议第一次接触编程或缺乏编程经验的开发者暂时跳过这一章节。

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「○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「○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"
```

第一周：Hello World 再入门

简介

Ruby 的 Hello World 怎么写?

```
puts 'Hello World'
```

学会了，今天也是努力的一天，睡觉吧。

等一等！睡太早啦！

什么是 Hello World?

Hello World 就是在屏幕上显示一行 Hello World 的最简单的程序。这通常是程序员接触一门新语言时所需要写的第一个程序。

为什么要学习 Hello World?

Hello World 程序的意义绝对不单单是打印一行「Hello World」这么简单。其核心是用来确认程序开发环境和运行环境是不是安装妥当，同时也能对语言的开发进行一个初步的认识。砍柴不误磨刀工，那我们花一些时间，仔细探讨 Ruby 的开发、运行环境配置。

让我们一起开心地开始吧。

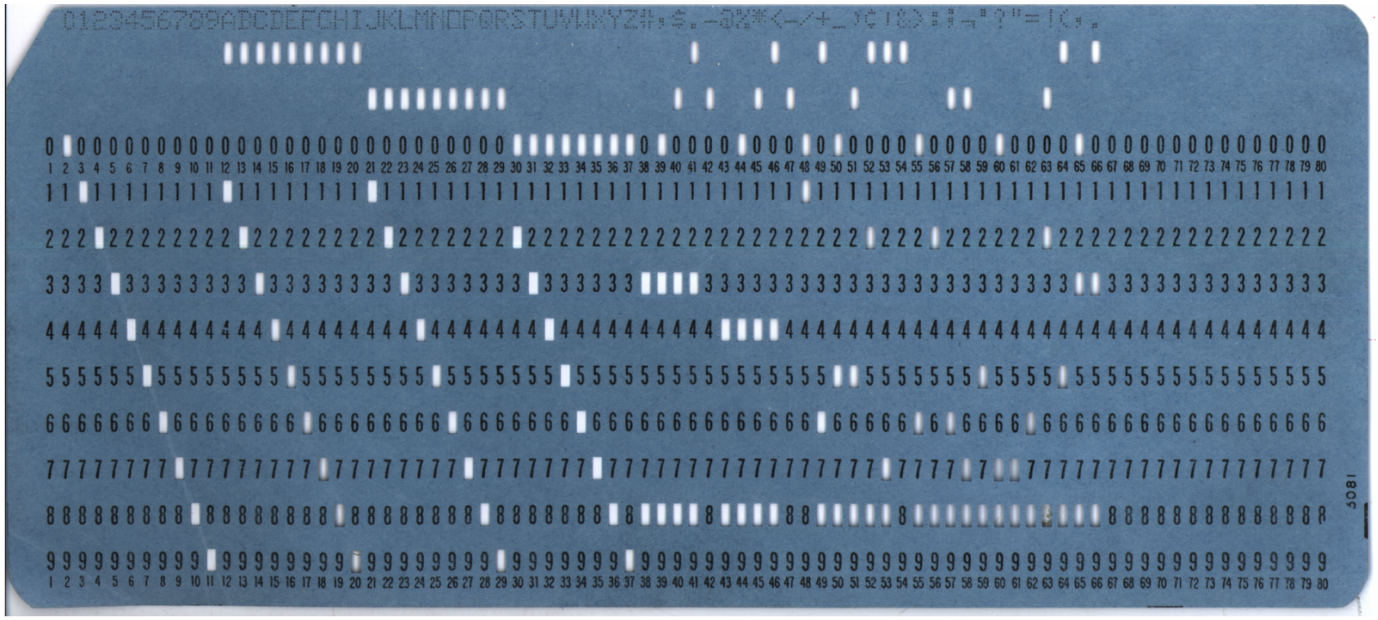
环境搭建

了解 Ruby 解释器

什么是解释器?

开发 Ruby 程序最重要环境的是 Ruby 解释器。「解释器」是在解释什么？解释前和解释后的东西是什么?

了解解释器，我们需要先了解另一个相近的概念「编译器」。在上世纪计算机刚刚出现的时候，程序员编程需要依赖纸带打孔卡或类似的穿孔纸带。



这里的打孔与不打孔就被作为一系列「开」和「关」信号传给电脑，形成程序交由计算机执行。今天的计算机已经淘汰了打孔纸带，计算机的启动和运行流程也变得越来越复杂，但程序依然是一系列「开」和「关」（1 或 0）的二进制信号。直接编写这些二进制，对于今天越来越复杂的程序开发要求来说，已经变得非常困难，于是我们发明了更接近于自然语言的高级语言。Ruby 就是这样一门高级语言。

高级语言不能直接被计算机执行，但是我们可以先通过一个「编译器」程序，读入这些高级语言，然后翻译成机器可以执行的二进制。「编译器」通常会把整个程序在运行前就完全编译到机器可以执行的二进制。「解释器」会在运行过程中，把程序一行一行直接翻译执行。解释器通常会比编译器执行得更慢，但同样也带来了更大的动态特性，允许更自由地开发方式。而 Ruby 的执行环境，通常就是这样一个「解释器」。

Ruby 解释器有什么?

Ruby 解释器是 Ruby 语言开发的核心。Ruby 是一个开放的语言，任何人都可以为 Ruby 实现自己的解释器。Ruby 的解释器多种多样，常见的仍在维护的解释器有：

- Ruby
- Ruby MRI (CRuby)
- JRuby
- TruffleRuby
- Rubinius
- RubyMotion
- Opal

- mruby
- mruby
- mruby/c

这些 Ruby 解释器各有一些差异，支持的语法和执行的性能也并不完全相同。本书所涉及的全部 Ruby 都指官方实现的 Ruby MRI。

MRI 代指 Matz’s Ruby Interpreter，即 Ruby 创始人松本行弘最早实现的 Ruby 解释器，是 Ruby 的官方解释器。虽然在 Ruby 1.9 之后的版本中，官方已经把虚拟机换成了由 Kiochi Sasada 主导的 YARV (Yet another Ruby VM) 解释器。但在 1.9 版本后，YARV 已经被合并到 MRI，此后我们已不特别区分 MRI 和 YARV 了，现在仍称呼这一解释器实现为 Ruby MRI。下文所有的 Ruby 解释器，如无特殊标注，都是 Ruby MRI 解释器的缩写。

在撰写本章节的时候，Ruby 的最新版本是 2.7.0，本书全部的代码都在 2.7.0 中进行过测试。我们会自动化测试本书代码在不同环境下的兼容性，以确保正确性。

安装方法

Windows 用户

在 Windows 上安装 Ruby 最简单的方式是 RubyInstaller。在本书的编写过程中，我们会测试书中所有涉及到的程序在 Windows 上的兼容性。但由于 Ruby 的第三方依赖，特别是一些设计给 *NIX 服务器的依赖程序，可能没有测试在 Windows 上的兼容性，从而可能在使用上会遇到一定的困难。

一些教程不推荐新手在 Windows 上开发 Ruby。但事实上，Ruby 的标准库对于 Windows 的兼容性还是相当良好的。如果没有人在 Windows 上使用 Ruby，那么 Ruby 运行在 Windows 上的问题会变得更多，这是一个恶性循环。但是对于初学者，使用 Linux/macOS 进行开发依然是我个人推荐的。主要问题是，初学者缺乏对环境问题处理的经验，遇到问题往往会不知所措。大多数服务器软件的生产环境更愿意使用自由的 Linux 操作系统，而使用 Ruby 开发服务器应用是最常见的用途，使用和生产环境一致的环境，至少是 *NIX 环境能有效避免问题发生的概率。

关于 PC 用户如何选择和安装 Linux 发行版，本书单独开设附录章节来描述，请参阅《附录一：新手如何安装 Linux 开发版？》。另外，在 Windows 10 中可以使用 Windows Subsystem for Linux (WSL) 来产生一个无缝的 Linux 环境。详情请参阅微软的 WSL 官方文档。

*NIX 用户（Linux、macOS、BSD 用户）

RVM

RVM 是最推荐新手安装 Ruby 的方法。笼统来说，使用 RVM 安装 Ruby 需要三步。如果你是 macOS 用户，你可能需要先安装 XCode 和 XCode Command Line Tools 才能安装 RVM。

XCode 可以从 App Store 直接下载到，安装完成后打开「终端（Terminal）」应用，使用 `xcode-select --install` 安装 XCode Command Line Tools。

对于 Linux 用户，请确认自己的 Terminal 是 login shell 的模式。一些 Linux 发行版自带的 Terminal 应用没有 login，可能没有加载用户的环境变量。

第一步是获取 GPG 公钥，RVM 使用 GPG 密钥系统来确保程序在传输过程中不被篡改。打开终端应用，输入下面的命令即可获取 GPG 密钥。

```
gpg2 --keyserver hkp://keys.gnupg.net --recv-keys
409B6B1796C275462A1703113804BB82D39DC0E3 7D2BAF1CF37B13E2069D6956105BD0E739499BDB
```

第二步则是一键安装脚本，下面的命令会下载 RVM。

```
\curl -sSL https://get.rvm.io | bash -s stable
source ~/.rvm/scripts/rvm # 载入 RVM 环境（新开 Terminal
就不用这么做了，默认自动重新载入的）
```

对于中国大陆地区用户，RVM 的自带镜像源可能下载速度太慢。为此 Ruby China 提供了镜像源，在执行第三步前可以使用

```
echo "ruby_url=https://cache.ruby-china.com/pub/ruby" > ~/.rvm/user/db
```

来切换镜像源。

最后一步就是执行下面命令来安装特定版本的 Ruby。

```
rvm install 2.7.0 # 这里的 2.7.0 可以切换成阅读本文时最新的 Ruby 版本。Ruby
最新版本可以在 https://www.ruby-lang.org/ 确认。
```

安装成功后可以使用

```
ruby -v
```

来检查所安装的 Ruby 版本有没有正确安装成功，如果返回了版本号，那么就是安装成功了。

详细的安装文档可以在 rvm.io 来查询。

snap

Ruby 在 2018 年 11 月 8 日加入了官方的 snap 套件支持。如果你使用 Ubuntu 16.04 的后续版本，你可以一键安装 Ruby。

```
sudo snap install ruby --classic
```

使用 Snap 安装的 Ruby 可能会在环境变量上需要一些额外的配置。建议检查官方的 新闻 来确认细节。

编译自源代码

编译自源代码是较为困难的安装方法，不建议新手使用这一方式。

在电子游戏《尼尔：机械纪元》的 用户协议 中，我们会发现出现了 Ruby License。可见在这款游戏中使用了 Ruby 语言实现了一定的功能。这款游戏首发在 PS4 平台上，而 PS4 的操作系统是一个修改自 FreeBSD 操作系统。所以 Ruby 语言对于 BSD 系的操作系统同样是非常友好的。

但如果你想在一些嵌入式设备上运行 Ruby 或者需要运行在 PS4 上，使用包管理器可能不是一个好主意，因为你不一定具有全局安装的权限或者不想引入额外的复杂度。这时候直接从源代码编译可能就变成了必须。

生日快乐！欢迎自己编译你的蛋糕。 (Photo by Monika Grabkowska on Unsplash)

从源代码编译安装很简单，你可以先从 Ruby 官方网站下载 最新的源代码，在解压后执行：

```
./configure
make
```

进行编译。Ruby 的编译过程中，一些组件是可选的，你需要自己确认这些可选的依赖是否准备妥当。如果编译后需要安装，你可以执行：

```
sudo make install
```

进行安装。

内置包管理器

使用例如 Ubuntu、Debian 内置的 `apt` 或者 CentOS、Fedora 内置的 `dnf` 或类似方法也可以很方便安装 Ruby。但是，大多数操作系统内建的软件源常有版本滞后、缺少组件的问题。如无必要，不推荐新手使用这样的安装方法。但如果你有一个可控、可信、维护良好的软件源，这也是一个不增加复杂度安装 Ruby 的好方法。

如何管理 Ruby 版本

使用 RVM 的一个好处是可以很好管理不同的 Ruby 版本。刚开始写 Ruby 的时候可能不太会意识到这件事情的重要性。但是下面这几种需求在实际开发中可能会发生：

- Ruby 版本更新了，怎么升级？
- 想临时测试某一特定版本 Ruby 的特性，怎么临时切换？
- 如何指定项目的 Ruby 版本，来确保服务器运行环境和开发环境一致？

如何安装新的 Ruby 版本？

在安装之前，你可以先用下面命令检查可以安装的 Ruby 版本：

```
rvm list known
```

如果你想要安装的版本没有出现在这个列表中，你也许需要更新 RVM。更新 RVM 的方法非常简单：

```
rvm get head
```

和你安装第一个 Ruby 版本一样，安装新的 Ruby 版本的方法完全一样，比如我想安装 2.7.0 版本，就可以使用下面命令。

```
rvm install 2.7.0
```

如何切换 Ruby 版本？

如果一个常见的需求是安装新 Ruby 版本后希望把默认 Ruby 版本切换到新安装的版本上，命令则是：

```
rvm --default use 2.7.0
```

如果你只是需要在当前 Shell 环境下临时切换，不需要设置成默认，只要把 `--default` 参数拿掉即可：

```
rvm use 2.7.0
```

切换后，可以用

```
ruby -v
```

确认切换后的 Ruby 版本。

如何设置项目的特定 Ruby 版本?

在项目根目录下放置一个名称为 `Gemfile` 的文件，并在里面写入如下的内容：

```
ruby '2.7.0'
```

RVM 就会在 shell 切换到这个目录下之后自动切换当前的 Ruby 版本。这样的设置还有一个好处，就是著名的 Ruby 托管平台 Heroku 也是使用这一方法来切换 Ruby 版本的。如果之后你需要将自己制作的网站托管到 Heroku 上的话，可以利用这一特性自动设置 Heroku 上的 Ruby 版本。

Gemfile 是 Bundler 提供依赖管理的重要文件，有关于这方面的功能，我们会在「第六周：Ruby 工程化入门」中重点介绍。

由于 Ruby 在设置版本上并没有官方制定的标准。管理项目版本依赖于不同版本管理工具的具体实现。在 RVM 中除了上面的方法，还有 4 种方法可以设置项目的 Ruby 版本。具体可以参考 RVM 的 Typical RVM Project Workflow。其中使用 `.ruby-version` 设置 Ruby 版本的方法可以同时在 RVM 和另一个在 Ruby 上常用的版本管理工具 rbenv 通用。