无网环境下的 Python 开发指南

发布 *0.0.1*

yangquan

2020 年 01 月 05 日

Contents

说起有关 Python 的指南，便一定得提到 K 神创建的『Python 最佳实践指南』，英文名字是『The Hitchhiker’s Guide to Python!』。毫不夸张地说，K 神创建的指南真正算地上是一份完美的指南示例，优雅无比，涵盖了有关 Python 安装、配置和日常使用的全部内容，并带有个人主观意见地推荐不同领域的 Python 框架，还提供非常多的学习资料，强烈推荐阅读。

K 神全名 Kenneth Reitz，被许多人称为社区英雄。只要是英雄，便会有很多崇拜者和模仿者，我也是其中之一。因为 K 神提供的指南实在过于优秀，作为迷弟的我也便学着他创建了这份指南，重点面向无网的工作环境，希望能够为无网环境下的 Python 开发人员提供一些参考，让 Python 能够进入更多无法联网的领域和场景，创造更多的价值。

让我们开始指南吧！

『万事开头难』，但安装 Python 一点都不难。

CHAPTER 1

Python 安装

1.1 Anaconda 发行版本

Anaconda 是一个包含了大量内置数据科学包的 Python 发行版本，有开源的免费版本，也有收费的企业版本。因为 Anaconda 内置了大量的 Python 包，包括 Jupyter、Spyder、Numpy、Scipy、pandas 等常用的 Python 包，其安装包较大，大概有 500 多 M。**正因为内置了大量的 Python 包，在离线环境下选择使用 Anaconda 安装 Python 会是一个明智的选择**。Anaconda 不仅提供了能够在多个平台上（Windows、macOS、Linux）一键安装的可执行二进制文件，还能够避免无法联网安装常见 Python 包的问题，并提供了相应的环境变量配置，能够让系统默认的 python 解释器和 python 命令变成最新安装的 Python 版本。Anaconda 的下载包，可以去 Anaconda 的官网下载。

1.2 Python 官方支持

Python 的官网提供了不同 Python 版本的官方安装包，包括了 Python 源码、macOS 安装二进制文件、Windows 安装二进制文件等，也提供了一些不常用平台的安装方法。如果你想要获取最新版本的 Python，便可以去 Python 官网上下载相应的安装包进行安装。

针对 Linux，Python 官方提供的是源码安装方式，**但源码安装 Python 需要有一定的 Linux 操作系统基础，因为源码安装 Python 的时候，需要安装一些额外的系统库和头文件，这样才能完整安装 Python，否则安装的 Python 是不完整的，一些扩展库是无法使用的，比如 zlib、sqlite、readline 等**。此外，Linux 操作系统大多有自己的软件包管理器，比如 Centos 的 yum、Ubuntu 的 apt，因此我们也可以下载指定 Python 版本相应的 rpm 包及其依赖或者 deb 包及其依赖完成 Python 的安装。这里就不详细介绍源码安装 Python 和下载 rpm 包以及 deb 包的方法，具体的操作可以查看我在 Github 上开源的文档仓库：python-guide-offline。

『工欲善其事，必先利其器』，选择一个合适的工具再开始。

CHAPTER 2

代码编写工具

2.1 Pycharm

Pycharm 是 JetBrains 公司针对 Python 推出的专业开发工具。就像 IDEA 在 Java 工程师之间的流行，Pycharm 也无可争议的是目前最好用的 Python IDE。它提供了强大的智能提示功能，支持大量的 Web 框架和性能测试工具，**并具备远程运行、调试程序的功能**。远程调试运行 Python 程序的功能有时是非常实用的，因为经常我们编写的 Python 代码是需要运行在 Linux 服务器上的，而不是运行在本地的 Windows 服务器上。

2.2 VSCode

虽然 Pycharm 功能强大，但有时会觉得 Pycharm 过于庞大和笨重，启动速度感人。因此，有时你还会需要 VScode 的帮助。 VSCode 是微软开源的一款现代化编辑器，支持各种语言，并以扩展的模式提供了强大的功能支持，Nodepad ++ 肯定因为 VSCode 减少了不少用户。最近微软在极力推广 Python，还推出了远程模式的 VSCode，还官方支持了 NoteBook。不管怎样，当你需要打开单个 Python 文件，或者需要阅读某个项目的源码时，VsCode 都是一个不错的选择。

2.3 Jupyter

Jupyter 项目主要提供交互式计算，帮助用户在数据探索和算法设计阶段，获知代码的中间结果，从而迭代地去编写并改进代码。**事实上，我们无法一次性就设计好代码逻辑，而需要在获知代码中间结果的基础上，持续编程**。

Jupyter 的名称来自，Julia、Python 和 R 三种编程语言的组合，以表明它对这三种编程语言的支持。然而事实上，目前 Jupyter 已经支持 40 种编程语言的交互式计算了。Jupyter 项目包含了多个子项目，包括 JupyterNotebook、JupyterLab 和 JupyterHub。

JupyterLab 是下一代的 JupyterNotebook，提供了一个基于 Web 的交互式开发环境，整合了 Notebook 的功能，并同时提供了多种数据的预览，相比 Notebook 有更好的 UI 设计和体验。同时，JupyterLab 还提供了可扩展的模块化支持，阿里的 PI 平台中关于拖拽神经网络的模块便是基于 JupyterLab 开发。

JupyterHub 则主要面向多用户，通过 JupyterHub 你可以方便地为多个的用户提供交互式开发服务，而不相互影响。

试着在多个 Python 开发环境中无缝切换。

CHAPTER 3

开发环境管理

3.1 本地 Python 仓库

管理 Python 包最好的工具就是 pip，但因为无法联网，我们无法配置 pip 源，从而随心所欲地下载相应的 Python 包。还好 Python 社区提供了一些搭建本地 Python 仓库的工具，能够方便我们在联网环境中下载需要的 Python 包，拷贝到离线环境后，通过配置本地的 Python 仓库，使用 pip install 命令安装指定的 Python 包。Awesome Python 网站提供了 4 个用于搭建本地的 Python 仓库，包括 warehouse、bandersnatch、devpi、localshop。warehouse 是 Python 官方的 Python 仓库，bandersnatch 是对整个 Python 库的镜像，容量过大。devpi 和 localshop 使用起来都需要更多的精力。**如果只是小型的团队，更加推荐先是使用 Pypiserver，它更加小型，也更加实用**。

虽然说 Python 代码也是跨平台的，但各个 Python 包仍然有不同操作系统和芯片指令集的区别。去查看 PyPi 网站提供的有关 tensorflow 的 Python 包列表，便可以看到不同 Python 版本、不同操作系统以及不同芯片指令集的区别：

因此当我们使用 `pip download` 命令下载 Python 包的时候，默认只是下载当前命令运行平台相关的 Python 包，包括当前的 Python 版本、操作系统以及芯片指令集。在离线环境下，由于导入资料的成本的高昂，我们有时希望一次导入所有版本的 Python 包和依赖；另一方面，又由于我们经常会命令从 Windows 操作系统上下载 Linux 系统上的 Python 包和依赖，我们也希望能够直接下载不同操作系统、不同芯片指令集甚至 Python 版本的 Python 库。**虽然 pip 提供了以下跨平台的操作指令和参数，但有时仍然会因为依赖的问题而导致下载失败**。因此，如果你需要下载某个 Python 包，我这里会推荐我之前编写的一个 Python 库：pip-download，虽然目前还是我一个人使用，但还是很方便的可以试一下。

3.2 虚拟环境

开发的 Python 项目多了以后，便经常会遇到依赖冲突的问题。项目 1 用了 A 版本的 Python 库 package，项目 2 用了 B 版本的 Python 库 package，那么如果都用系统版本的 Python 环境，是无法同时安装同一 Python 库的两个版本。因而，virtualenv 和 Python 3 自带的 venv 模块便能发挥左右。你可以通过执行如下的命令建立虚拟环境，并进行激活，从而拥有一个干净的 Python 依赖：

```
$ python3 -m venv venv
$ source venv/bin/activate

$ pip install virtualenv
$ virtualenv
```

此外，Anaconda 的 Python 版本也自带了虚拟环境管理工具 conda，Pyenv 则是一个能够建立不同 Python 版本虚拟环境的一个工具，而在 Python 包依赖及其管理上，还有很多知名的工具值得了解：poetry。Python 在包及其依赖上的管理实在过于宽泛，导致了诸多不方便和不一致。但我们还有未来不是吗?

选择一个好用的框架，避免重复造轮子。

CHAPTER 4

Python 框架

4.1 web 框架

Python 已经发展了很多成熟的框架，使用起来非常的方便。这里主要推荐 web 开发的一些框架，而其它的 Python 框架则会持续更新。Flask 是 Python 中的一个微服务 Web 框架，对 Web 开发新手极其友好，比 Spring MVC 和 Spring Boot 好太多了。Django 是 Python 中的一个企业级 Web 框架，包含各种开箱即用的功能，Instagram 便是 Django 的忠实用户。而 Django Rest Framework 和 Flask-Rest 则是编写 restful 接口两个非常好用的工具。

随着异步范式的流行，Python 生态圈也出现许多的异步 Web 框架，以下项目也要开始了解：Starlette、Sanic、fastapi、uvicorn。

吃饭，睡觉，阅读文档。

CHAPTER 5

Python 文档资料

5.1 Python 官方文档

在知乎上有个一个好友，每次我提出有关 Python 语言的问题时，他都能引用 Python 的官方文档给我解答，足以可见 Python 文档的强大。目前，Python 的文档已经有了中文翻译版本，在 Python 官方文档的侧边栏，可以下载不同版本和不同形式的 Python 文档，导入到无法联网环境中，便能够随时浏览和查阅 Python 的官方文档。有时下载官方的文档无法提供搜索的功能，则可能是因为 *search.js* 这个文件权限不足造成的。

5.2 Python 书籍

目前，我只能推荐 3 本我阅读过的 Python 书籍，包括《Python Tricks》、《Python Cookbook》、《Fluent Python》。《Python Tricks 》非常简单易懂，适合入门 Python 后开始 Python 进阶的第一本读物。《Python Cookbook》写得非常好，是编写 Python 代码时可以随时查找样式代码的最好示例。《Fluent Python》也是我极其推荐的一本书，我甚至后悔没有早点读到这本书，读完整本书才真正体会到之前对 Python 的理解有多么肤浅，之前写的关于 Python 的文章实在是不堪入目。不过，我一点都不灰心，因为一件事不可能一开始就做的很好的，总是慢慢地做，慢慢地坚持，才会做得很好。

没有规矩不成方圆，来干了这份完美的 Python 编码规范。

CHAPTER 6

Python 代码编程规范

如果想要别人看你的代码时，面露微笑，而不是难色。那就看看这些 Python 编程规范吧。规范了自己，才能规范别人。

目前，有两份 Python 编程规范比较被人认可，一份是 Python 官方的 PEP 8 风格指南，一份是互联网谷歌公司的风格指南。

6.1 谷歌风格指南

谷歌推出了一份 Python 代码的风格指南，原文请参见 这里，中文翻译版请参见 这里。

6.2 PEP 8 风格指南

PEP 的全称是 Python Enhancement Proposals，PEP 8 这个提案是关于 Python 代码风格的官方指南，原文在 这里，下面是一些总结，具体的例子请参加原文。

1. 尽信书则不如无书，代码风格并不是一定要遵守，在保证代码可读性的基础上，去满足代码风格。
2. 用 4 个空格进行缩进，同时利用括号、方括号、大括号或者悬挂缩进来对齐包裹其中的元素。
3. 代码行最多包含 79 个字符，注释内容和文档字符串每行最多包含 72 个字符。有时可以使用反斜杠符 \ 来连接超长的 with 语句和 assert 语句。
4. 在二元操作符 (比如：+、-、*、/等) 的前面断行，而不是后面，使用 `a \n + b`，而不是 `a + \n b`。

5. 用 2 个空行包围定义在 Python 文件里的函数和类，用 1 个空行包围定义在类里的方法，空行的数量可以根据需要进行增加或者删减。
4. 每行只导入一个 Python 库，但可以在一行导入同一个 Python 库的多个模块；import 语句应该统一放置在 Python 文件的最开头，放在模块注释和文档后面，同时在全局变量和常量之前；import 语句应该按照标准库 -> 第三方库 -> 本地库的顺序排列，并用空行进行分割；尽量避免使用 `from <module> import *`。
5. Python 模块级别的双下划线变量（比如：`__all__`、`__author__`、`__version__`) 应该放在模块字符串和 `from __future__ import` 语句的后面，同时放在普通 `import` 语句的前面。
6. 单引号字符和双引号字符是一样的，可以替换着使用；使用三个双引号来引用文档字符串。
7. 合理使用空格，避免在圆括号、中括号、大括号、逗号、分号、冒号、等号的周围使用多余的空格；在适当第二元操作符周围加上一个合适的空格。
8. 一个元素的元组应该保留一个尾随逗号，列表或者参数以每一行的方式扩展时，需要保留一个尾随逗号。
9. 及时更新注释，注释的开头第一个字母大写。

用绝对防御保护好你的源代码。

CHAPTER 7

保护 Python 代码

7.1 现有解决方案

知乎网友 Prodesire 发表了两篇文章介绍了他的 Python 源代码加密解决方案，值得参考：

1. 如何保护你的 Python 代码（一）——现有加密方案
2. 如何保护你的 Python 代码（二）——定制 Python 解释器

7.2 全新解决方案

Python wiki 里面有一个关于 How do you protect Python source code? 的回答，里面提到唯一安全的代码是驻留在远程机器上的代码，同时也提到对于 Java 、C 来说，保护代码也会是徒劳的，而和你的用户建立一个更好的互助关系也是一个不错的选择。但无论如何，在法律还不成熟的中国，保护 Python 代码还是需要做的。针对当前的解决方案都不是特别优雅，我在 Github 上新开了一个项目叫做 goldenmask，希望可以提供一个方便易用的 Python 代码加密工具，让 Python 开发人员能够更加关注在开发这件事上。期待你的加入！

Python 太慢，也可以跑过兔子。

CHAPTER 8

加速 Python 代码

Python 有时是很慢，但试试下面的方法，或许你再也不会说出这样的观点。

1. 异步
2. 多线程
3. 多进程
4. C 语言扩展

高级编程，aka: High Level Programing。

CHAPTER 9

Python 高级编程

总是再使用 Python 库，你也想自己编写 Python 库了。下面这些高级特性你必须了解：

1. 装饰器
2. 元类
3. 描述符
4. 可迭代对象、迭代器、生成器
5. 上下文管理器
6. 协程
7. 多继承
8. 协议
9. 接口
10. 重载运算符

未完待续。

CHAPTER 10

待补充的内容

- Python 框架介绍，包括命令行、GUI 等
- Python 源码编译为 rpm、deb 包
- 优秀的 Python 开源项目
- 优秀的 Python 开发人员
- 个人 Python 风格指南
- 详细的 Python 加速方案
- 详细的 Python 代码加密方案
- 详细的 Python 高级编程概念解释说明