

软件架构

Python语言实现

[印度] 阿南德·巴拉钱德拉·皮莱 (Anand Balachandran Pillai) 著

李必信 廖力 王璐璐 周颖 等译

SOFTWARE ARCHITECTURE
WITH PYTHON

如何用Python语言设计和构建高度可扩展、健壮、干净和高性能的软件架构产品



机械工业出版社
China Machine Press

架构师书库

SOFTWARE ARCHITECTURE
WITH PYTHON

软件架构

Python语言实现

[印度] 阿南德·巴拉钱德拉·皮莱 (Anand Balachandran Pillai) 著

李必信 廖力 王璐璐 周颖 等译



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

软件架构: Python 语言实现 / (印) 阿南德·巴拉钱德拉·皮莱著; 李必信等译. —北京: 机械工业出版社, 2018.2

(架构师书库)

书名原文: Software Architecture with Python

ISBN 978-7-111-59094-1

I. 软… II. ①阿… ②李… III. 软件工具 - 程序设计 IV. TP311.561

中国版本图书馆 CIP 数据核字 (2018) 第 025273 号

本书版权登记号: 图字 01-2017-7518

Anand Balachandran Pillai: Software Architecture with Python (ISBN: 9781786468529).

Copyright © 2017 Packt Publishing. First published in the English language under the title “Software Architecture with Python”.

All rights reserved.

Chinese simplified language edition published by China Machine Press.

Copyright © 2018 by China Machine Press.

本书中文简体字版由 Packt Publishing 授权机械工业出版社独家出版。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。

软件架构: Python 语言实现

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 张梦玲

责任校对: 李秋荣

印 刷: 北京市荣盛彩色印刷有限公司

版 次: 2018 年 3 月第 1 版第 1 次印刷

开 本: 186mm×240mm 1/16

印 张: 25

书 号: ISBN 978-7-111-59094-1

定 价: 79.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88379426 88361066

投稿邮箱: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光/邹晓东

近年来，人们对软件架构的关注度和重视程度越来越高了，具体表现在：一方面，几乎所有高等院校的计算机学院、软件学院，甚至物联网学院、网络安全学院等都在开设软件架构相关课程，而且很多高校和研究院所投入大量的人力从事软件架构的研究和开发；另一方面，大型软件企业、互联网企业和电商企业也开始非常重视软件架构的设计、实现和演化，很多著名的企业（例如，华为、腾讯、阿里巴巴、微软、IBM、谷歌等）都成立了专门的软件架构部门，甚至成立了研究院。每年的世界架构师大会、各种类型的软件架构研讨会、工作会议等，把全世界从事软件架构相关工作的人聚集在一起，为软件架构的知识体系添砖加瓦，为软件架构的未来发展出谋划策。

遗憾的是，现有的很多软件架构书籍要么只注重理论阐述、内容太过抽象，缺少软件架构实践内容；要么融合了很多与软件架构核心内容不太相关的其他知识点，使得大家在学习时无法掌握重点，甚至是无法正确地理解软件架构。由此带来的后续问题是，学生虽然学习了软件架构课程，但对软件架构的认识仍然不够深入，仍然缺乏软件架构的实践经验，走入社会后也很难承担软件架构方面的工作。

作为一本软件架构方面的书，本书的定位准确，且全面、系统地介绍了软件架构的核心内容。本书共 10 章，在阐述软件架构各个层面的知识点的同时，又重点突出了软件架构的特点和重要性，如何设计好的软件架构，软件架构对系统性能、可靠性和安全性的影响，以及软件架构与其他各种软件质量属性（例如，可测试性、可维护性等）的关系等。另外，本书的一个主要特色是，结合 Python 语言阐述了设计和实现良好软件架构的过程，是一本难得的既有软件架构的理论阐述，又有软件架构的实践案例的好书。

本书的用途和读者对象如下：①作为高等院校的教科书，面向高年级本科生和研究生；②作为软件架构研究人员的参考书，本书打穿了软件架构从设计到实现再到演化的各个环节，为实验和仿真研究提供了很好的基础；③作为软件架构师及其他工程技术人员的工具书，本书中的软件架构设计和实现案例为解决实践中遇到的架构问题具有很好的借鉴作用。

参加本书翻译工作的人员主要是来自东南大学软件工程研究所、东南大学计算机科学与工程学院的教师和部分高年级研究生，包括廖力、王璐璐、周颖、宋启威、韩伟娜、

李慧丹、谢仁松、杨安奇、杜鹏程、尹强、宋震天、葛丹薇、孔祥龙、王桐、刘辉辉、熊壬浩、邱建鹏、汪小飞、苏晓威、段鹏飞、王家慧、汤立辉等。在翻译过程中，还得到了倪巍伟、周晓宇、戚晓芳、汪鹏、张祥等老师的帮助。在此，对他们的辛勤劳动表示衷心的感谢，也特别感谢机械工业出版社张梦玲编辑的无私帮助。

限于水平，对内容的理解和中文语言表达难免存在不当之处，在此敬请读者批评指正。但无论如何，本书是一本非常优秀的软件架构读物，本人也十分荣幸能够向读者推荐，认真地阅读本书一定使你受益匪浅。

李必信

2017 年于南京九龙湖

Anand Balachandran Pillai 是一名工程技术专家，在软件企业有 18 年以上的工作经历，在产品工程、软件设计、架构设计和相关研究方面具有非常丰富的经验。

他曾获得印度理工学院机械工程专业学士学位。曾在 Yahoo!、McAfee 和 Infosys 等公司任职，担任产品开发团队的首席工程师。

他的主要兴趣在于软件性能工程、高可扩展性架构、安全和开源社区等方面。他也经常在 Startups 工作，担任首席技术专家或顾问。

他还是班加罗尔 Python 用户联盟的奠基人和 Python 软件协会 (PSF) 的会士。Anand 现在是 Yegii 公司的首席架构师。

.. 关于评审人 ..

Mike Driscoll 从 2006 年开始使用 Python。他喜欢写一些关于 Python 的博客，见 <http://www.blog.pythonlibrary.org/>。他曾合著了《the Core Python refcard for DZone》一书，并参与了《Python 3 Object Oriented Programming》《Python 2.6 Graphics Cookbook》《Tkinter GUI Application Development Hotshot》的评审工作和其他几本书的撰写工作。他最近刚完成《Python 101》的编写，目前正在写作他的下一本书。

感谢我的妻子 Evangeline 一如既往的支持，感谢我的朋友和家人对我的无私帮助。

软件架构，可以说是为特定的应用软件创建一个蓝图设计。软件架构中存在两大挑战：首先，软件架构与需求必须保持一致；对尚未发现的需求或者发生演化的需求都是如此；其次，尽管常常发生架构实现的变更，但软件架构与其对应的架构实现必须保持一致。

本书包含很多示例和用例，通过这种直观的方法来帮助你获取成为一名成功的软件架构师所需的一切。本书将帮助你了解 Python 的来龙去脉，以便可以用 Python 来构建和设计高度可扩展的、健壮的、简洁的、性能强大的应用程序。

主要内容

第 1 章介绍了软件架构的核心思想，简要介绍了架构质量属性和一些隐含的原理。这将使你能够在软件架构原理和基本属性方面拥有良好的知识基础。

第 2 章包括开发中软件架构的可修改性和可读性。它将帮助你深入理解架构的可维护性等质量属性，并获得用 Python 编写代码来测试应用程序的各种技巧和策略。

第 3 章帮助你理解软件架构的可测试性，以及如何为 Python 应用程序构建架构以满足可测试性。你还将了解可测试性和软件测试的各个方面，以及 Python 中可用的各种库和模块，以便编写各种可测试的应用程序。

第 4 章讨论了在编写 Python 代码过程中关于性能的方方面面。你不仅可以学习架构性能的基本知识，还可以掌握在何时何地需要进行性能优化。例如，你会学习到何时进行 SDLC 的性能优化。

第 5 章不仅阐述了编写可扩展应用程序的重要性，还讨论了实现应用程序可扩展性的各种不同方法，并论述了如何利用 Python 来实现各种可扩展性技术。你不仅能学到可扩展性的理论方面的知识，还能学到业界的最佳实践。

第 6 章讨论了架构安全性的方方面面，并使你掌握一些最佳实践和技巧来编写安全性高的应用程序。你会了解在 Python 架构应用程序中可能出现的各种不同的安全问题，以及 Python 是如何从头开始保障安全性的。

第 7 章从程序员实用性的角度，简要论述了 Python 中出现的各种设计模式以及每个

模式的理论背景。这些设计模式对程序员来说是非常实用的。

第 8 章从较高抽象层次角度介绍 Python 中现有的架构模式，同时给出了几个示例，用来说明如何利用 Python 库和框架来实现基于这些模式的高层次架构问题的解决方法。

第 9 章讨论如何正确地在远程环境中或云上使用 Python 轻松部署代码的方方面面。

第 10 章讨论了一些 Python 代码调试技术，包括最简单实用的打印语句、日志记录和系统调用跟踪机制等，这些对程序员来说都是非常容易获得的，也有助于系统架构师指导他的团队。

阅读本书需要准备什么

为运行本书中展示的大部分代码示例，需要在系统中安装 Python 3。其他的预备知识会在相应的实例中提到。

本书的读者对象

本书适用于有经验的 Python 开发人员，他们渴望成为企业级应用程序的架构师；本书也适用于软件架构师，他们希望利用 Python 的特长来创建更有效的应用程序蓝图。

约定

书中的代码块设置如下：

```
class PrototypeFactory(Borg):
    """ A Prototype factory/registry class """

    def __init__(self):
        """ Initializer """

        self._registry = {}

    def register(self, instance):
        """ Register a given instance """

        self._registry[instance.__class__] = instance

    def clone(self, klass):
        """ Return clone given class """

        instance = self._registry.get(klass)
        if instance == None:
            print('Error:', klass, 'not registered')
        else:
            return instance.clone()
```

当希望重点关注代码块的某个特定部分时，会将相关的行 (line) 或项 (item) 设置成

粗体:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,VoiceMail(u100)
exten => s,102,VoiceMail(b100)
exten => i,1,VoiceMail(s0)
```

任何命令行输入或输出写成如下形式:

```
>>> import hash_stream
>>> hash_stream.hash_stream(open('hash_stream.py'))
'30fbc7890bc950a0be4eaa60e1fee9a1'
```

新术语和重要词汇以粗体形式表示。

示例代码下载

可以从 <http://www.packtpub.com> 下载本书的示例代码文件(需要 Packt 账户)。也可以访问 <http://www.packtpub.com/support> 并注册账户, Packt 将通过 email 把文件直接发送给你。

可以按照以下步骤下载代码文件:

- (1) 使用你的电子邮件地址和密码注册或登录到我们的网站。
- (2) 将鼠标指针悬停在顶部的 **SUPPORT** 选项卡上。
- (3) 单击 **Code Downloads & Errata** 选项。
- (4) 在 **Search** 框中输入图书的名称。
- (5) 选择要下载代码文件的书。
- (6) 从下拉菜单中选择本书的购买渠道。
- (7) 单击 **Code Download** 下载。

你还可以单击 Packt Publishing 网站中图书页面上的 **Code Files** 按钮下载代码文件,也可以通过在 **Search** 框中输入图书的名称来访问此页面。请注意,你需要首先登录 Packt 账户。

下载文件后,请确保使用这些最新版软件来解压缩文件或文件夹:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

该书的代码包也由 GitHub 托管在 <https://github.com/PacktPublishing/Software-Architecture-with-Python>。你还可从 Packt 提供的图书和视频目录中获取其他代码包,网址为 <https://github.com/PacktPublishing/>。

.. 目 录 ..

译者序

关于作者

关于评审人

前言

第 1 章 软件架构原理 1

1.1 软件架构定义 2

1.1.1 软件架构与设计 2

1.1.2 软件架构相关的几个方面 3

1.2 软件架构的特征 3

1.2.1 用架构来定义一种结构 3

1.2.2 由架构来挑选一组核心元素 4

1.2.3 由架构来捕获早期的设计决策 4

1.2.4 由架构来管理利益相关者的需求 5

1.2.5 架构影响着组织结构 5

1.2.6 架构受到环境的影响 6

1.2.7 架构是对系统的文档化 6

1.2.8 架构通常会遵循某个模式 7

1.3 软件架构的重要性 7

1.4 系统架构与企业架构 8

1.5 架构的质量属性 10

1.5.1 可修改性 11

1.5.2 可测试性 13

1.5.3 可扩展性 14

1.5.4 性能 15

1.5.5	可用性	16
1.5.6	安全性	17
1.5.7	可部署性	18
1.6	本章小结	19

第2章 编写可修改可读的代码 20

2.1	什么是可修改性	20
2.2	与可修改性相关的几个方面	20
2.3	理解可读性	21
2.3.1	Python 和可读性	21
2.3.2	可读性 - 反模式	22
2.4	增强可读性的各种技术	24
2.4.1	文档化代码	24
2.4.2	遵守编码和风格规范	30
2.4.3	审查和重构代码	31
2.4.4	注释代码	31
2.5	可修改性的基础——内聚和耦合	32
2.5.1	测量内聚性和耦合性	33
2.5.2	字符串和文本处理	35
2.6	探索提高可修改性的策略	37
2.6.1	提供显式接口	37
2.6.2	减少双向依赖	37
2.6.3	抽象出公共服务	38
2.6.4	使用继承技术	38
2.6.5	使用延迟绑定技术	42
2.7	度量——静态分析工具	43
2.7.1	什么是代码坏味道	43
2.7.2	圈复杂度——McCabe 度量	44
2.7.3	度量结果测试	45
2.7.4	运行静态检查器	47
2.8	重构代码	53
2.8.1	降低复杂度	53
2.8.2	改善代码坏味道	55
2.8.3	改善风格上和编码上的问题	57

2.9 本章小结 57

第3章 可测试性——编写可测试的代码 58

3.1 理解可测试性 58

3.1.1 软件可测试性及相关属性 58

3.1.2 架构级的方方面面 59

3.1.3 策略 60

3.2 白盒测试原理 65

3.2.1 单元测试 65

3.2.2 操作中的单元测试 66

3.2.3 单元测试模块 nose2 69

3.2.4 用 py.test 进行测试 70

3.2.5 代码覆盖 72

3.2.6 仿制一些东西 74

3.2.7 文档中的内联测试——doctest 78

3.2.8 集成测试 81

3.2.9 测试自动化 83

3.3 测试驱动开发 84

3.4 有回文的 TDD 85

3.5 本章小结 90

第4章 好的性能就是回报 92

4.1 什么是性能 93

4.2 软件性能工程 93

4.3 性能测试和度量工具 94

4.4 性能复杂度 95

4.5 度量性能 96

4.5.1 使用上下文管理器度量时间 97

4.5.2 使用 timeit 模块来计时代码 99

4.5.3 使用 timeit 度量代码的性能 100

4.5.4 揭示时间复杂度——各种图 102

4.5.5 使用 timeit 度量 CPU 时间 106

4.6 剖析 107

4.6.1 确定性剖析 107

4.6.2	使用 cProfile 和 profile 进行剖析	108
4.6.3	收集和报告统计数据	111
4.6.4	第三方剖析器	113
4.7	其他工具	119
4.7.1	objgraph	120
4.7.2	pympler	121
4.8	程序设计性能——数据结构	123
4.8.1	可变容器——链表、字典和集合	123
4.8.2	不可变容器——元组	124
4.8.3	高性能容器——集合模块	125
4.8.4	概率数据结构——布隆过滤器	131
4.9	本章小结	134

第 5 章 开发可扩展的应用 136

5.1	可扩展性和性能	137
5.2	并发性	139
5.2.1	并发性与并行性	140
5.2.2	Python 中的并发性——多线程机制	141
5.3	缩略图产生器	141
5.3.1	缩略图产生器——生产者 / 消费者架构	143
5.3.2	缩略图产生器——使用锁的资源约束	147
5.3.3	缩略图产生器——使用信号量的资源约束	150
5.3.4	资源约束——信号量和锁比较	153
5.3.5	缩略图产生器——使用条件的 URL 速率控制器	153
5.4	多线程机制——Python 和 GIL	160
5.4.1	Python 中的并发性——多进程机制	160
5.4.2	质数检查器	161
5.4.3	排序磁盘文件	163
5.5	多线程与多进程比较	168
5.6	先入为主的与合作的多任务处理	170
5.7	Python 中的 asyncio 模块	173
5.8	等待 future 对象——async 和 await	175
5.9	concurrent.future——高级并发处理	178
5.9.1	磁盘缩略图产生器	179

5.9.2	并发选项——如何选择?	181
5.10	并行处理库	182
5.10.1	joblib	182
5.10.2	PyMP	183
5.10.3	fractals —— Mandelbrot 集	184
5.11	Web 扩展	189
5.11.1	扩展工作流——消息队列和任务队列	189
5.11.2	Celery —— 一种分布式任务队列	190
5.11.3	在 Web 上使用 Python 服务——WSGI	194
5.12	可扩展架构	197
5.12.1	垂直可扩展架构	197
5.12.2	水平扩展架构	198
5.13	本章小结	201

第 6 章 安全性——编写安全代码 202

6.1	信息安全架构	202
6.2	安全编码	203
6.3	常见的安全漏洞	204
6.4	Python 安全吗?	208
6.4.1	读取输入	209
6.4.2	任意输入求值	211
6.4.3	溢出错误	214
6.4.4	序列化对象	216
6.5	Web 应用的安全问题	219
6.5.1	服务器端模板注入	220
6.5.2	服务器端模板注入——回避	222
6.5.3	服务拒绝	223
6.5.4	跨站脚本攻击	226
6.5.5	回避——DoS 和 XSS	227
6.6	Python 中的安全策略	228
6.7	安全编码策略	234
6.8	本章小结	234

第7章 Python 设计模式 236

7.1 设计模式——元素 237

7.2 设计模式分类 237

7.2.1 可插拔的散列算法 238

7.2.2 可插拔的散列算法总结 242

7.3 Python 模式——创建模式 242

7.3.1 单例模式 242

7.3.2 工厂模式 248

7.3.3 原型模式 250

7.3.4 建造者模式 256

7.4 Python 模式——结构化模式 261

7.4.1 适配器模式 261

7.4.2 外观模式 269

7.4.3 代理模式 275

7.5 Python 模式——行为模式 279

7.5.1 迭代器模式 279

7.5.2 观察者模式 282

7.5.3 状态模式 288

7.6 本章小结 293

第8章 Python 架构模式 295

8.1 MVC 概述 296

8.1.1 模型模板视图——Django 297

8.1.2 Django 管理——自动的以模型为中心的视图 297

8.1.3 灵活的微框架——Flask 299

8.2 事件驱动编程 300

8.2.1 采用 I/O 多路复用 select 模块的聊天服务器和客户端 300

8.2.2 事件驱动与并发编程 306

8.2.3 Twisted 306

8.2.4 Eventlet 313

8.2.5 Greenlet 和 Gevent 314

8.3 微服务架构 316

8.3.1 Python 中的微服务框架 317

8.3.2 微服务实例——餐馆预订 317

8.3.3 微服务的优点 320

8.4 管道和过滤器架构 320

8.5 本章小结 325

第9章 部署 Python 应用程序 326

9.1 可部署性 327

9.1.1 影响可部署性的因素 327

9.1.2 软件部署架构的层次 328

9.2 Python 软件部署 329

9.2.1 给 Python 代码打包 329

9.2.2 使用 Fabric 进行远程部署 340

9.2.3 使用 Ansible 进行远程部署 341

9.2.4 使用 Supervisor 管理远程守护进程 342

9.3 部署——模式和最佳实践 343

9.4 本章小结 344

第10章 各种用于调试的技术 346

10.1 最大子阵列问题 347

10.1.1 “print” 的力量 348

10.1.2 分析和重写 349

10.1.3 代码计时和代码优化 351

10.2 简单的调试技巧和技术 353

10.2.1 词搜索器程序 353

10.2.2 词搜索器程序——调试步骤 1 354

10.2.3 词搜索器程序——调试步骤 2 355

10.2.4 词搜索器程序——最终代码 356

10.2.5 跳过代码块 357

10.2.6 停止执行 357

10.2.7 使用 wrapper 来控制外部依赖 358

10.2.8 用函数的返回值或数据来替换函数（模拟） 359

10.3 作为一种调试技术的日志记录 367

10.3.1 简单的应用程序日志记录 368

10.3.2 高级日志记录——日志记录器对象 369