HZ BOOKS
华章科技
PEARSON

全球资深Python专家Doug Hellmann作序鼎力推荐，Amazon全五星评价，Python领域最有影响力的著作之一

从设计模式、并发技术和程序库角度，围绕Python编程的核心问题，系统而详细地讲解各种实用Python编程技术和技巧，并以3个完整的案例展示“设计–实现–优化”的全过程，带你领略Python语言之美，提升Python编程水平

华章程序员书库

Python in Practice

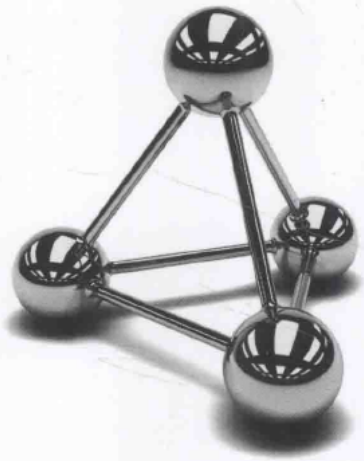
Create Better Programs Using Concurrency, Libraries, and Patterns

Python编程实战

运用设计模式、并发和程序库创建高质量程序

（美） Mark Summerfield 著

爱飞翔 译



机械工业出版社
China Machine Press

华章程序员书库

Python in Practice

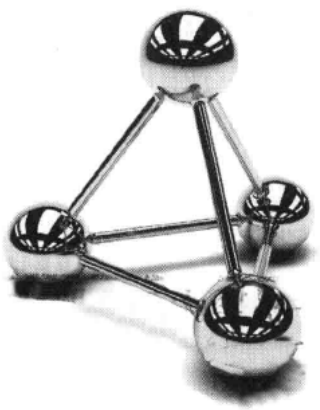
Create Better Programs Using Concurrency, Libraries, and Patterns

Python编程实战

运用设计模式、并发和程序库创建高质量程序

（美） Mark Summerfield 著

爱飞翔 译



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Python 编程实战：运用设计模式、并发和程序库创建高质量程序 /（美）萨默菲尔德（Summerfield, M.）著；爱飞翔译 . —北京：机械工业出版社，2014.7
（华章程序员书库）
书名原文：Python in Practice: Create Better Programs Using Concurrency, Libraries, and Patterns

ISBN 978-7-111-47394-7

I. P… II. ①萨… ②爱… III. 软件工具－程序设计 IV. TP311.56

中国版本图书馆 CIP 数据核字（2014）第 161735 号

本书版权登记号：图字：01-2014-3566

Python 编程实战：
运用设计模式、并发和程序库创建高质量程序

[美] Mark Summerfield 著

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：关 敏　　责任校对：殷 虹
印　刷：北京市荣盛彩色印刷有限公司　　版　次：2014 年 8 月第 1 版第 1 次印刷
开　本：186mm × 240mm 1/16　　印　张：16.75
书　号：ISBN 978-7-111-47394-7　　定　价：69.00 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）88378991 88361066　　投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259　　读者信箱：hzjsj@hzbook.com

The Translator's Words 译 者 序

由于 Python 语言的写法非常简洁，而且应用范围又很广泛，所以近年来吸引了很多开发者积极投身其中。Python 语言的基础教程种类繁多，开发者的入门过程也特别快。在掌握了基础知识之后，许多程序员还想进一步提升自己的 Python 编程水平。这时主要会遇到三大问题。

第一是如何运用设计模式来规划代码结构，使之既易于修改，又易于维护；第二是如何通过并发及 Cython 等技术提升代码执行速度；第三是如何利用各种 Python 程序库来快速开发具体的应用程序和游戏。

针对这三大问题，本书都做了非常精彩的解答。作者把设计模式分为“创建型”、“结构型”与“行为型”三类，并分别比较了每种设计模式的传统用法与它在 Python 中的用法究竟有何异同。读者通过这部分讲解，可以学会如何在 Python 中简化或扩充传统的设计模式，也能明白为何有些模式在 Python 中无须使用。

许多开发者都想通过并发来提升程序性能，但由于要处理“资源竞争”、“加锁”等复杂的问题，所以并发式应用程序很容易出错，而本书作者则会告诉我们怎样利用 `queue` 及 `future` 等高级数据结构来避免这些错误，此外，他还辨析了多进程技术与多线程技术的适用场合。其后，作者又讲解了如何用 Python 来调用 C 或 C++，并通过实例证明了 Python 并不是一门效率低下的语言——只要合理运用并发及 Cython 技术，照样可以写出速度很快的程序。

在具体的应用领域中，作者着重讲解了如何利用 Python 程序库来简化网络及图形编程，并把图形编程又细分为用户界面编程和三维图形编程。由于 Python 的程序库非常丰富，所以读者只要把学到的解题思路灵活地运用到自己的工作领域，并辅以适当的程序库，就能快速开发出符合需求的程序。

本书不仅提供了大量实用的范例代码，而且还有三个完整的案例研究。这三个案例再现

了“设计—实现—优化”的全过程，并使我们领略到 Python 语言之美。

尽管本书作者认为读者应该有一定的 Python 基础，但 Python 其实是一门亲和力很高的语言，即便你对 Python 知之甚少，但只要用其他语言编过程序，也依然可以从 `for x in range(1,10): print(x)`、`(2, 3, 4) * 2`、`[9,8,7,6][1:3]` 等浅显易懂的写法中轻松得知循环、元组及列表等功能的用法。如果你想快速掌握 Python 语言，那不妨在开发文档的帮助下，试着直接读这本书。

总之，无论你是老手还是新手，本书都是学习 Python 的良好伴侣。相信大家在读完本书之后，应该能用其中的技术和思路把 Python 程序写得更加优秀。

翻译过程中，我得到了机械工业出版社华章公司诸位编辑与工作人员的帮助，在此深表感谢。

本书附有“中英文词汇对照表”，其中列出了相关词汇的各种译法，供大家参考。有兴趣的朋友可至华章公司网站 www.hzbook.com 下载。由于时间仓促，译者水平有限，错误与疏漏之处在所难免，敬请读者批评指正。大家可发邮件至 eastarstormlee@gmail.com 与我联系，也可访问网页 http://agilemobidev.com/eastarlee/book/python-in-practice 留言。

爱飞翔

Preface 序

这 15 年来，我用 Python 编程语言开发过各种应用程序，在此过程中，Python 开发者社区也逐渐成熟与壮大起来。从前，我们必须向管理人员“推销”Python 语言，以便其允许开发者在相关工作项目中使用它。而现在则不同了，懂 Python 语言的程序员在就业市场中备受青睐，而参加各种 Python 技术会议的人也络绎不绝，有些是区域性的会议，有些是大型的国内或国际会议。如 OpenStack 这样的项目既能拓宽 Python 语言的应用领域，又能吸引开发好手投入 Python 阵营。社区壮大之后，优秀的 Python 书籍也比原来多了。

Mark Summerfield 是 Python 开发者社区里的知名技术作家，写有 Qt 及 Python 等方面的著作。我是乔治亚州亚特兰大市 Python 开发者聚会的组织人，经常有开发者问我：学 Python 语言看什么书好？在我列出的推荐书目中，Mark 所著的《 Programming in Python 3》名列榜首。Mark 的这本新作当然也在书单里，只是目标读者有些不同。

大部分编程书籍都位于两个极端：要么是向某门语言初学者（或刚开始学习编程之人）做一些简单介绍，要么就是专门讲解 Web 开发、图形用户界面应用程序开发、生物信息学等特定的高端话题。在写作《 The Python Standard Library by Example 》[⊖]一书时，我想填补两极之间的空白，也就是想写给那些有一定基础的程序员或计算机通才，这部分读者既想提升知识水平，又不愿局限在某个狭小的应用范围内。编辑请我审阅本书的出版提案时，我发现这本书的目标读者恰与我自己的书一致，于是颇感欣慰。

我始终在寻找这样一本书：它里面的内容不针对某个特定的框架或程序库，读者在阅读过程中一旦产生某个想法，就立刻能将其运用于目前正在做的项目中。过去几年里我一直在开发一套系统，用于计量 OpenStack 云服务。在这个过程中，开发团队发现：计费系统所收集到的数据也可以有其他用途，比如，报表系统和监视系统也能用到。于是，在设计系统时，我们通过一条管线来传递样本数据，以便将其转换并发布为各种格式，这样一来，就能把数据传给多个数据消费端了。当这条管线的代码快要写好时，我开始参与这本书的技术评审工作了。读完草稿第 3 章前面几节之后，我清楚地意识到，当初把管线实现得太过复杂

⊖ 中译《Python 标准库》，刘炽等译，机械工业出版社 2012 年出版。——译者注

了。Mark 在书中演示的“协程链”技术非常优雅，而且易于理解，所以，我立刻在项目的“路线图”里加了个任务，计划在下个发行周期里用该技术来修改管线设计。

本书有很多实用的建议及范例代码，大家可以像上面所说的那样，用它们来改进自己的项目。此外，书中还介绍了许多有趣的编程技法，即便大家像我一样读过很多代码，也依然有可能发现一些原来不知道的技巧。无论是很有经验的老程序员，还是正在寻求自我提升的新手，本书都会引领大家从不同的角度来观察问题，使你能用学到的技术创建出更为高效的解决方案。

Doug Hellmann
DreamHost 公司资深开发者
2013 年 5 月

Preface 前　言

本书面向有志于拓展及深化 Python 知识的读者，它将教你如何改进 Python 程序的质量、可靠性、速度、可维护性以及可用性。书中包含大量实用的范例与思路，可帮助大家提升 Python 编程水平。

本书有四大主题：用设计模式编写出优雅的代码、用并发和“编译过的 Python”（也就是 Cython）提升处理速度、高级网络编程，以及图形。

《Design Patterns: Elements of Reusable Object-Oriented Software》（详情参见附录 B）一书虽然早在 1995 年就出版了，然而时至今日，依然深刻地影响着面向对象编程这一领域。本书从 Python 语言的角度重新审视前书所提到的每种设计模式，给出实用的 Python 范例，并解释为何 Python 程序员用不到某些模式。这些模式在第 1 章、第 2 章及第 3 章中讲解。

Python 的 GIL（Global Interpreter Lock，全局解释器锁）会阻止 Python 代码同时在多个处理器核心上运行[⊖]。于是有人就误以为 Python 不支持多线程，或无法发挥多核硬件的优势。对于“计算密集型”（CPU-bound）程序来说，可以用 `multiprocessing` 模块实现并发，该模块不受 GIL 限制，可以完全利用每个核心。这样一来，处理速度就很容易提高了（大致同 CPU 的核心数成正比）。对于“I/O 密集型”程序来说，我们既可以用 `multiprocessing` 模块来做，也可以用 `threading` 模块或 `concurrent.futures` 模块来做。实际上，使用 `threading` 模块来编写 I/O 密集型程序时，并不用担心由 GIL 所带来的开销，因为网络延迟的影响更大。

遗憾的是，在编写并发程序时，如果采用低级与中级方式[⊜]，那么非常容易出错，任

⊖ CPython 有此限制。CPython 是大部分 Python 程序员所使用的“参考实现”（reference implementation）。某些 Python 实现没有这一限制，这其中最有名的就是 Jython（也就是用 Java 语言所实现的 Python）。

⊜ 这里所说的“低级”（low-level）或“中级”（medium-level）方式，是指封装程度不那么高、偏底层的方式。——译者注

何编程语言都有这种问题。要想少出错，就不要使用“显式锁”（explicit lock），而是改用 Python 的 queue 及 multiprocessing 模块，这些模块提供了封装程度较高的“队列”（queue），此外，也可以改用 concurrent.futures 模块来做。第 4 章会告诉大家如何用封装程度较高的并发技术来大幅提高程序性能。

某些程序员之所以使用 C、C++ 或其他“编译型语言”（compiled language）来编程，是因为他们还有另外一个错误的想法，那就是 Python 程序运行得很慢。一般来说，Python 确实要比编译型语言慢，但在目前的硬件上面，用 Python 语言所编写的绝大部分应用程序的运行速度都足够快。即便有时 Python 程序真的不够快，我们也可以一边享受用 Python 编程所带来的好处，一边想办法提升其运行速度。

如果要给某些长期运行的程序提速，那么可以使用 PyPy 这款 Python 解释器（网址是 pypy.org）。这是一种“即时编译器”（just-in-time compiler），可以极大提升程序执行速度。另外一种优化执行效率的方式是调用运行速度与编译后的 C 程序相仿的代码，对于“计算密集型”程序来说，用这种代码改写后，其执行速度很容易变成原来的 100 倍。要想使 Python 程序运行得和 C 程序一样快，最简单的办法就是调用那种底层以 C 语言来实现的 Python 模块。比方说，标准库里的 array 模块或第三方 numpy 模块都能飞快地处理数组，并且很省内存（多维数组可以用 numpy 来处理）。除此之外，还可以使用标准库的 cProfile 模块来探查程序的瓶颈，并用 Cython 来写对速度要求很高的那部分代码。这种写法实际上就是一套“增强版 Python 语言”（enhanced Python）：写好的程序可以编译成纯 C，从而使运行速度提升到极致。

当然，有时候我们所需的功能已经由现成的 C 或 C++ 库实现好了，或者由采用“C 语言调用约定”（C calling convention）的其他语言程序库实现好了。在大多数情况下，都能找到第三方 Python 模块来访问我们所需的那些程序库，这些模块可以在 Python Package Index（简称 PyPI，网址是 pypi.python.org）里找到。不过在个别情况下可能找不到这种模块，此时可以用标准库的 ctypes 模块或第三方的 Cython 包来调用 C 程序库里的功能。采用已经实现好的 C 程序库来编写代码能够极大减少开发时间，而且 C 代码的运行速度也相当快。第 5 章讲解 ctypes 与 Cython。

Python 标准库提供了许多用于网络编程的模块，比如底层的 socket 模块、中层的 socketserver 模块，以及高层的 xmlrpclib 模块。把用其他语言所写的代码移植到 Python 时，可能会用到底层与中层网络模块，然而直接用 Python 编程时，通常不需要理会那些底层的细节，只需要用高层模块来实现所需的网络功能即可。第 6 章讲解如何使用标准库中的 xmlrpclib 模块以及功能强大且易用的第三方 RPyC 模块。

每个程序差不多都要提供某种用户界面，使用户可通过它来向程序下达指令。可以用

argparse 模块来编写 Python 程序，使其支持命令行界面；也可以用其他模块来编写，使其支持完整的终端用户界面（例如，在 Unix 系统上，可用第三方 urwid 包实现这种界面，此包的网址为 excess.org/urwid）。此外，有许多 Web 框架能够实现出 Web 界面，比如轻量级的 bottle 框架（网址是 bottlepy.org）、重量级的 Django 框架（网址是 www.djangoproject.com）与 Pyramid 框架（网址是 www.pylonsproject.org）。当然，除了上面说的这几种界面外，还可以创建具有“图形用户界面”（Graphical User Interface，GUI）的 Python 应用程序。

经常听到“ Web 程序将取代 GUI 程序”这种说法，不过现在还没发展到那一步。实际上，用户可能更喜欢 GUI 程序，而不是 Web 程序。比方说，在 21 世纪初智能手机刚开始流行时，用户总是爱用专门制作好的“ app”而不是浏览器中的网页来处理日常事务。有许多第三方 Python 包都可用来编写 GUI 程序。本书第 7 章要介绍的 Tkinter 包位于 Python 标准库里，该章会告诉大家如何用它创建样式新潮的 GUI 程序。

目前大多数计算机（包括笔记本电脑及部分智能手机）都配有功能强大的图形渲染硬件，这种硬件通常是独立的 GPU（ Graphics Processing Unit，图形处理单元），能够绘制出绚丽的二维及三维图形。而大多数 GPU 都支持 OpenGL API，所以 Python 程序员可以通过第三方包来调用这套 API。第 8 章将会讲解怎样用 OpenGL 绘制三维图形。

本书旨在演示如何编写更好的 Python 程序，如何写出效率高、易维护且易于使用的 Python 代码。阅读之前，需要有 Python 编程基础，因为此书是写给已经学会 Python 语言用法的读者看的，大家应该已经读过 Python 的开发文档或是类似教程了，比如《 Programming in Python 3, Second Edition》（详情参见附录 B）等书。而这本书将提供一些有助于提升 Python 编程水平的思路、灵感与实用技巧。

本书全部范例代码都在 Linux 系统的 Python 3.3 版本下测试通过（笔者也尽量在 Python 3.2 及 Python 3.1 版本下测试过了），绝大部分代码还能在 OS X 与 Windows 操作系统中运行。可从本书网站 www.qtrac.eu/pipbook.html 下载范例代码，这些代码也应该能在后续的 Python 3.x 版本下运行。

致谢

与写其他技术书籍时一样，笔者写这本书时也得到了大家的诸多建议、帮助及鼓励，在此深表谢意。

Nick Coghlan 从 2005 年起成为 Python 的核心开发者，他提供了大量建设性的批评意见，并展示了许多想法及代码片段，以此表明书中所讲的某些内容还有更好的实现方式。Nick 对

笔者改进本书内容帮助极大，尤其是前面几章。

Doug Hellmann 是资深 Python 开发者与技术作者，他给笔者写了许多条非常有用的评论，从成书之前的出版提案到成书之后正文里的每一章都是如此。Doug 还给笔者提供了许多思路，并为本书撰写了序。

两位友人 Jasmin Blanchette 与 Trenton Schulz 都是有经验的 Python 程序员，他们各自的研究方向迥然不同，但都位于本书所讲的范围之内。Jasmin 与 Trenton 反馈了许多意见，使笔者能够据此改写正文及范例代码中的许多不够明晰之处。

感谢策划编辑 Debra Williams Cauley，在成书过程中，他再次向我提供了支持和帮助。

感谢 Elizabeth Ryan 精心管理了本书的出版流程，感谢 Anna V. Popick 出色的校对工作。

最后，一如往常，感谢妻子 Andrea 的关爱与支持。

Contents 目　录

译者序
序
前言

第 1 章　Python 的创建型设计模式 ····· 1

1.1　抽象工厂模式 ····· 1

1.1.1　经典的抽象工厂模式 ····· 2

1.1.2　Python 风格的抽象工厂模式 ····· 4

1.2　建造者模式 ····· 6

1.3　工厂方法模式 ····· 12

1.4　原型模式 ····· 18

1.5　单例模式 ····· 19

第 2 章　Python 的结构型设计模式 ····· 21

2.1　适配器模式 ····· 21

2.2　桥接模式 ····· 26

2.3　组合模式 ····· 31

2.3.1　常规的“组合体 / 非组合体”式层级 ····· 32

2.3.2　只用一个类来表示组合体与非组合体 ····· 35

2.4　修饰器模式 ····· 37

2.4.1　函数修饰器与方法修饰器 ····· 38

2.4.2　类修饰器 ····· 42

2.5 外观模式 …… 47
2.6 享元模式 …… 52
2.7 代理模式 …… 54

第 3 章 Python 的行为型设计模式 …… 58

3.1 责任链模式 …… 58
3.1.1 用常规方式实现责任链 …… 59
3.1.2 基于协程的责任链 …… 60
3.2 命令模式 …… 63
3.3 解释器模式 …… 66
3.3.1 用 eval() 函数求表达式的值 …… 67
3.3.2 用 exec() 函数执行代码 …… 70
3.3.3 用子进程执行代码 …… 73
3.4 迭代器模式 …… 76
3.4.1 通过序列协议实现迭代器 …… 77
3.4.2 通过双参数 iter() 函数实现迭代器 …… 77
3.4.3 通过迭代器协议实现迭代器 …… 79
3.5 中介者模式 …… 81
3.5.1 用常规方式实现中介者 …… 82
3.5.2 基于协程的中介者 …… 85
3.6 备忘录模式 …… 87
3.7 观察者模式 …… 87
3.8 状态模式 …… 91
3.8.1 用同一套方法来处理不同的状态 …… 93
3.8.2 用不同的方法来处理不同的状态 …… 94
3.9 策略模式 …… 95
3.10 模板方法模式 …… 98
3.11 访问者模式 …… 101
3.12 案例研究：图像处理程序包 …… 102
3.12.1 通用的图像处理模块 …… 103
3.12.2 Xpm 模块概述 …… 111
3.12.3 PNG 包装器模块 …… 113

第 4 章　Python 的高级并发技术 …… 116

4.1　计算密集型并发 …… 119

4.1.1　用队列及多进程实现并发 …… 121

4.1.2　用 Future 及多进程实现并发 …… 126

4.2　I/O 密集型并发 …… 128

4.2.1　用队列及线程实现并发 …… 129

4.2.2　用 Future 及线程实现并发 …… 134

4.3　案例研究：并发式 GUI 应用程序 …… 136

4.3.1　创建 GUI …… 138

4.3.2　编写与工作线程配套的 ImageScale 模块 …… 144

4.3.3　在 GUI 中显示图像处理进度 …… 146

4.3.4　处理 GUI 程序终止时的相关事宜 …… 148

第 5 章　扩充 Python …… 150

5.1　用 ctypes 访问 C 程序库 …… 151

5.2　Cython 的用法 …… 159

5.2.1　用 Cython 访问 C 程序库 …… 159

5.2.2　编写 Cython 模块以进一步提升程序执行速度 …… 164

5.3　案例研究：用 Cython 优化图像处理程序包 …… 169

第 6 章　Python 高级网络编程 …… 173

6.1　编写 XML-RPC 应用程序 …… 174

6.1.1　数据包装器 …… 174

6.1.2　编写 XML-RPC 服务器 …… 178

6.1.3　编写 XML-RPC 客户端 …… 180

6.2　编写 RPyC 应用程序 …… 188

6.2.1　线程安全的数据包装器 …… 188

6.2.2　编写 RPyC 服务器 …… 193

6.2.3　编写 RPyC 客户端 …… 195

第 7 章　用 Tkinter 开发图形用户界面 …… 199

7.1　Tkinter 简介 …… 201

7.2　用 Tkinter 创建对话框 …… 203

7.2.1　创建对话框式应用程序 …… 205

7.2.2　创建应用程序中的对话框 …… 212

7.3　用 Tkinter 创建主窗口式应用程序 …… 220

7.3.1　创建主窗口 …… 222

7.3.2　创建菜单 …… 224

7.3.3　创建带计分器的状态栏 …… 226

第 8 章　用 OpenGL 绘制 3D 图形 …… 229

8.1　用透视投影法创建场景 …… 230

8.1.1　用 PyOpenGL 编写 Cylinder 程序 …… 231

8.1.2　用 `pyglet` 编写 Cylinder 程序 …… 235

8.2　用正交投影法制作游戏 …… 238

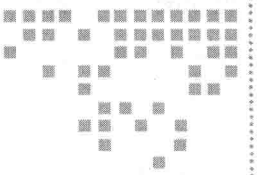
8.2.1　绘制游戏场景 …… 240

8.2.2　判断用户是否选中了场景里的物体 …… 242

8.2.3　处理用户操作 …… 244

附录 A　结束语 …… 248

附录 B　参考书目摘录 …… 250



第 1 章 Chapter 1

Python 的创建型设计模式

关乎对象创建方式的设计模式就是“创建型设计模式”（creational design pattern）。一般我们都是通过调用构造器（也就是用参数来调用类对象）来创建对象的，但有时候需要以更为灵活的方式来创建对象，而这正是创建型设计模式的用途。

对于 Python 程序员来说，其中某些设计模式彼此之间非常相似，而另外一些则根本用不到（稍后就要讲到）。有些设计模式主要是为 C++ 这种语言设计的，目的是绕开这些编程语言中的某些限制。而 Python 语言没有这些限制，所以就用不到它们了。

1.1 抽象工厂模式

“抽象工厂模式”（Abstract Factory Pattern）用来创建复杂的对象，这种对象由许多小对象组成，而这些小对象都属于某个特定的“系列”（family）。

比方说，在 GUI 系统里可以设计“抽象控件工厂”（abstract widget factory），并设计三个“具体子类工厂”（concrete subclass factory）[⊖]：`MacWidgetFactory`、`XfceWidgetFactory`、`WindowsWidgetFactor`，它们都提供创建同一种对象的方法（例如都提供创建按钮的 `make_button()` 方法，都提供创建数值调整框的 `make_spinbox()` 方法），而具体创建出来的对象的风格则与操作系统平台相符。我们可以编写 `create_dialog()` 函数，令其以“工厂实例”（factory instance）为参数来创建 OS X、Xfce 及 Windows 风格的对话框，对话框的具体风格取决于传进来的工厂参数。

⊖ concrete 一词也译为“具型”、“具象”、“实体”。——译者注

1.1.1 经典的抽象工厂模式

为了演示抽象工厂模式，我们来写一段程序，用以生成简单的“示意图”（diagram）。这段程序会用到两个“工厂”（factory）：一个用来生成纯文本格式的示意图，另一个用来生成 SVG（Scalable Vector Graphics，可缩放的矢量图）格式的示意图。图 1.1 列出了这两种格式。此程序有两种写法，`diagram1.py` 文件按照传统方式来运用抽象工厂模式，而 `diagram2.py` 则借助了 Python 的某些特性，这样写出来的程序比原来更短小、更清晰。这两个版本所生成的示意图都一样[⊖]。

```
+----------------------------+
|   +--------------------+   |
|   |%%%%%%%%%%%%%%%%%%%%|   |
|   |%%Abstract Factory%%|   |
|   |%%%%%%%%%%%%%%%%%%%%|   |
|   +--------------------+   |
+----------------------------+
```

图 1.1　纯文本格式与 SVG 格式的示意图

有一些代码是这两个版本都要用的，首先我们来看 `main()` 函数。

```
def main():
    ...
    txtDiagram = create_diagram(DiagramFactory())  ❶
    txtDiagram.save(textFilename)

    svgDiagram = create_diagram(SvgDiagramFactory())  ❷
    svgDiagram.save(svgFilename)
```

首先创建两个文件（上述范例代码中没有列出相关语句）。接下来，用默认的纯文本工厂（❶）创建示意图，并将其保存。然后，用 SVG 工厂（❷）来创建同样的示意图，也将其保存。

```
def create_diagram(factory):
    diagram = factory.make_diagram(30, 7)
    rectangle = factory.make_rectangle(4, 1, 22, 5, "yellow")
    text = factory.make_text(7, 3, "Abstract Factory")
    diagram.add(rectangle)
    diagram.add(text)
    return diagram
```

`create_diagram` 函数只有一个参数，就是绘图所用的工厂，该函数用这个工厂创建出所需的示意图。此函数并不知道工厂的具体类型，也无须关心这一点，它只需要知道工厂对象具备创建示意图所需的接口即可。以 `make` 开头的那些方法我们放在后面讲。

说完工厂的用法之后，我们来看工厂本身的写法。下面这个工厂类用来绘制纯文本示意图（该工厂也是其他工厂的基类）：

⊖ 本书全部范例代码均可从 www.qtrac.eu/pipbook.html 下载。