EFFECTIVE 系列丛书

HZ BOOKS 华章科技

从基本原则、惯用法、语法、库、设计模式、内部机制、开发工具和性能优化8方面深入探讨
编写高质量Python代码的技巧、禁忌和最佳实践

Writing Solid Python Code
91 Suggestions to Improve Your Python Program

编写高质量代码
改善Python程序的91个建议

张 颖 赖勇浩 著

机械工业出版社
China Machine Press

EFFECTIVE系列丛书

Writing Solid Python Code
91 Suggestions to Improve Your Python Program

编写高质量代码

改善Python程序的91个建议

张颖 赖勇浩 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

编写高质量代码：改善 Python 程序的 91 个建议 / 张颖，赖勇浩著 . —北京：机械工业出版社，2014.6
（Effective 系列丛书）

ISBN 978-7-111-46704-5

I. 编… II. ①张… ②赖… 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 100711 号

在通往“Python 技术殿堂”的路上，本书将为你编写健壮、优雅、高质量的 Python 代码提供切实帮助！内容全部由 Python 编码的最佳实践组成，从基本原则、惯用法、语法、库、设计模式、内部机制、开发工具和性能优化 8 个方面深入探讨了编写高质量 Python 代码的技巧与禁忌，一共总结出 91 条宝贵的建议。每条建议对应 Python 程序员可能会遇到的一个问题。本书不仅以建议的方式从正反两方面给出了被实践证明为十分优秀的解决方案或非常糟糕的解决方案，而且分析了问题产生的根源，会使人有一种醍醐灌顶的感觉，豁然开朗。

本书针对每个问题所选择的应用场景都非常典型，给出的建议也都与实践紧密结合。书中的每一条建议都可能在你的下一行代码、下一个应用或下一个项目中显露锋芒。建议你将本书搁置在手边，随时查阅，相信这么做一定能使你的学习和开发工作事半功倍。

编写高质量代码：
改善 Python 程序的 91 个建议

张 颖 等著

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：孙海亮 责任校对：殷 虹
印 刷：藁城市京瑞印刷有限公司 版 次：2014 年 6 月第 1 版第 1 次印刷
开 本：186mm × 240mm 1/16 印 张：17
书 号：ISBN 978-7-111-46704-5 定 价：59.00 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

华章计算机
HZBOOKS | Computer Science and Technology

Preface 前　言

为什么要写这本书

当这本书的写作接近尾声的时候，回过头来看看这一年多的写作历程，不由得心生感叹，这是一个痛并快乐着的过程。不必说牺牲了多少个周末，也不必计算多少个夜晚伏案写作，单是克服写作过程中因疲劳而迸发出来的彷徨、犹豫和动摇等情绪都觉得是件不容易的事情。但不管怎么说，这最终是个沉淀和收获的过程，写作的同时我也和读者们一样在进步。为什么要写这本书？可以说是机缘巧合。机械工业出版社的杨福川老师联系到我，说他们打算策划一本关于高质量 Python 编程方面的书籍，问我有没有兴趣加入。实话实说，最开始我是持否定态度的，一则因为业余时间实在有限，无法保证我“工作和生活要平衡”的理念；二则觉得自己水平有限，在学习 Python 的道路上我和千千万万读者一样，只是一个普通的“朝圣者”，我也有迷惑不解的时候，在没有修炼到大彻大悟之前拿什么来给人传道授业？是赖勇浩老师的加入给我注入了一针强心剂，他丰富的 Python 项目经验以及长期活跃于 Python 社区所积累下来的名望无形中给了我一份信心。杨老师的鼓励和支持也更加坚定了我的态度，经过反复考虑和调整自己的心态，最终我决定和赖老师一起完成这本书。因为我也经历过从零开始的 Python 学习过程，我也遇到过各种困惑，经历过不同的曲折，这些可能也正是每一个学习 Python 的人从最初到进阶这一过程中都会遇到的问题。抱着分享自己在学习和工作中所积累的一点微薄经验的心态，我开始了本书的写作之旅。这个过程也被我当作是对自己学过的知识的一种梳理。如果与此同时，还能够给读者带来一些启示和思索，那将是这本书所能带给我的最大收获了。

读者对象

- ❑ 有一定的 Python 基础，希望通过项目最佳实践来提升自己的相关 Python 人员。
- ❑ 希望进一步掌握 Python 相关内部机制的技术人员。

- 希望写出更高质量、更 Pythonic 代码的编程人员。
- 开设相关课程的大专院校师生。

如何阅读本书

首先需要注意的是，本书并不是入门级的语法介绍类的书籍，因此在阅读本书之前假定你已经掌握了最基础的 Python 语法。如果没有，也没有关系，你可以先找一本最简单的介绍 Python 语法的书籍看看，尝试写几个 Python 小程序之后再来阅读本书。

本书分为 8 章，主要从编程惯用法、基础语法、库、设计模式、内部机制、开发工具、性能剖析与优化等方面解读如何编写高质量的 Python 程序。每个章节的内容都以建议的形式呈现，这些建议或源于实际项目应用经验，或源于对 Python 本质的理解和探讨，或源于社区推荐的做法。它们能够帮助读者快速完成从入门到进阶的这个过程。

由于各个章节相对独立，因此无须花费整段的时间从头开始阅读。你可以在空闲的时候选取任意感兴趣的小节阅读。为了减轻读者负担，本书代码尽量保持完整，阅读过程中无须额外下载其他相关代码。

勘误和支持

由于作者的水平有限，加之编写时间仓促，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正。如果你在阅读过程中遇到任何问题或者发现任何错误，欢迎发送邮件至邮箱 highqualitypython@163com，我们会尽量一一解答直到你满意。期待能够得到你的真挚反馈。

致谢

首先要感谢机械工业出版社华章公司的杨福川老师，因为有了你的鼓励才使我有勇气开始这本书。还要感谢机械工业出版社的孙海亮编辑，在这一年多的时间中始终支持我的写作，是你的鼓励和帮助引导我顺利完成全部书稿。当然也要感谢我的搭档赖老师，和你合作是一件非常愉快的事情，也让我收获颇多。

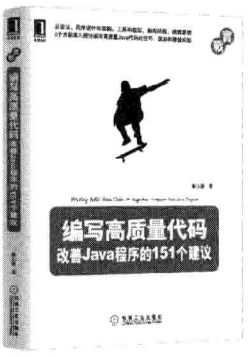
其次要感谢我的家人，是你们的宽容、支持和理解给了我完成本书的动力，也是你们无微不至的照顾让我不必为生活中的琐事烦心，从而能全身心地投入到写作中去。

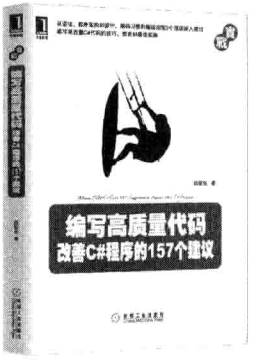
最后，我想提前感谢一下本书的读者，谢谢你们能够选择阅读这本书，这将是作为作者的我们最大的荣幸。

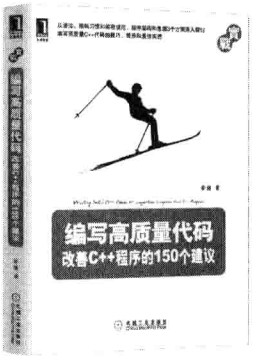
谨以此书献给所有热爱 Python 的朋友们！

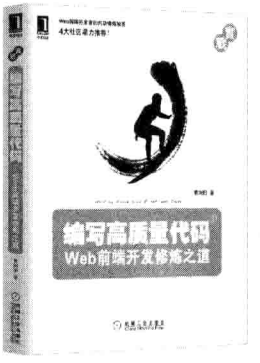
张　颖

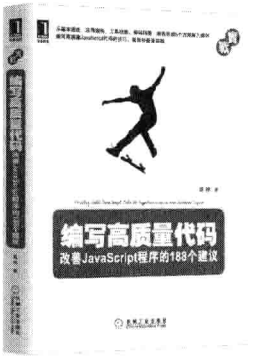
推荐阅读

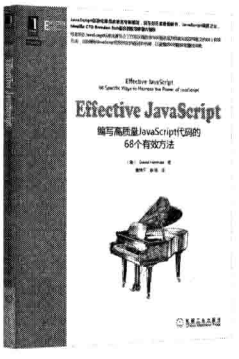

编写高质量代码
改善Java程序的151个建议

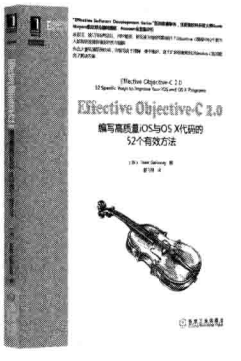

编写高质量代码
改善C#程序的157个建议

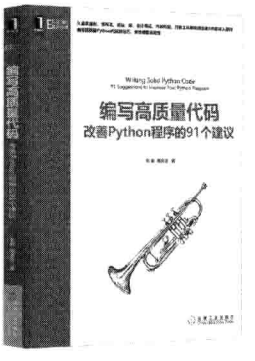

编写高质量代码
改善C++程序的150个建议


编写高质量代码
Web前端开发修炼之道


编写高质量代码
改善JavaScript程序的188个建议

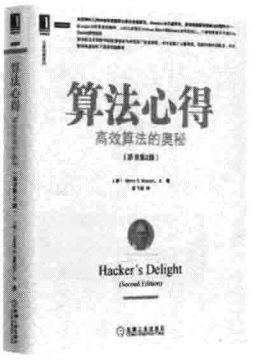

Effective JavaScript
编写高质量JavaScript代码的
68个有效方法

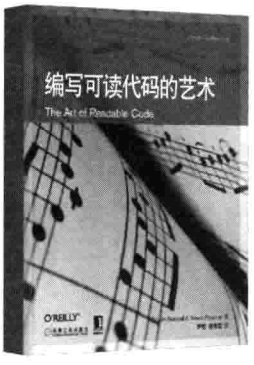

Effective Objective-C 2.0
编写高质量iOS与OS X代码的
52个有效方法

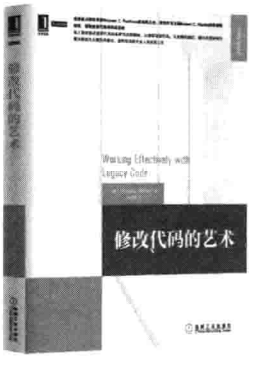

编写高质量代码
改善Python程序的91个建议

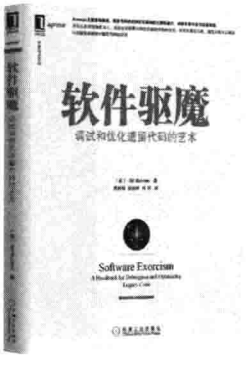
推荐阅读

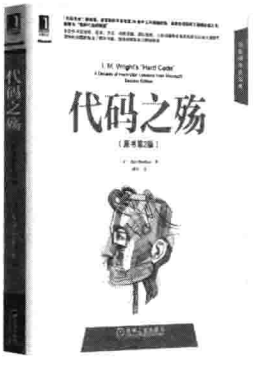

Mastering Algorithms with C
算法精解
C语言描述
O'REILLY


算法心得
高效算法的奥秘
Hacker's Delight
(Second Edition)


编写可读代码的艺术
The Art of Readable Code
O'REILLY


Working Effectively with Legacy Code
修改代码的艺术


软件驱魔
调试和优化遗留代码的艺术
Software Exorcism


代码之殇
[原书第2版]

Contents 目　录

第 1 章 Chapter 1

引　论

“罗马不是一天建成的”，编写代码水平的提升也不可能一蹴而就，通过一点一滴的积累，才能达成从量变到质变的飞跃。这种积累可以从很多方面取得，如一些语言层面的使用技巧、常见的注意事项、编程风格等。本章主要探讨 Python 中常见的编程准则，从而帮助读者进一步理解 Pythonic 的本质。本章内容包括如何编写 Pythonic 代码、在实际应用中需要注意的一些事项和值得提倡的一些做法。希望读者通过对本章的学习，可以在实际应用 Pythonic 的过程中得到启发和帮助。

建议 1：理解 Pythonic 概念

什么是 Pythonic？这是很难定义的，这就是为什么大家无法通过搜索引擎找到准确答案的原因。但很难定义的概念绝非意味着其定义没有价值，尤其不能否定它对编写优美的 Python 代码的指导作用。

对于 Pythonic 的概念，众人各有自己的看法，但大家心目之中都认同一个更具体的指南，那就是 Tim Peters 的《The Zen of Python》（Python 之禅）。在这一充满着禅意的诗篇中，有几点非常深入人心：

- 美胜丑，显胜隐，简胜杂，杂胜乱，平胜陡，疏胜密。
- 找到简单问题的一个方法，最好是唯一的方法（正确的解决之道）。
- 难以解释的实现，源自不好的主意；如有非常棒的主意，它的实现肯定易于解释。

不仅这几点，其实《Python 之禅》中的每一句都可作为编程的信条。是的，不仅是作为编写 Python 代码的信条，以它为信条编写出的其他语言的代码也会非常漂亮。

（1）Pythonic 的定义

遵循 Pythonic 的代码，看起来就像是伪代码。其实，所有的伪代码都可以轻易地转换为可执行的 Python 代码。比如在 Wikipedia 的快速排序[⊖]条目中有如下伪代码：

```
function quicksort('array')
    if length('array') ≤ 1
        return 'array'  // an array of zero or one elements is already sorted
    select and remove a pivot element 'pivot' from 'array'  // see 'Choice of pivot' below
    create empty lists 'less' and 'greater'
    for each 'x' in 'array'
        if 'x' ≤ 'pivot' then append 'x' to 'less'
        else append 'x' to 'greater'
     return concatenate(quicksort('less'), list('pivot'), quicksort('greater'))
        // two recursive calls
```

实际上，它可以转化为以下同等行数的可以执行的 Python 代码：

```
def quicksort(array):
    less = []; greater = []
    if len(array) <= 1:
        return array
    pivot = array.pop()
    for x in array:
        if x <= pivot: less.append(x)
        else: greater.append(x)
    return quicksort(less)+[pivot]+quicksort(greater)
```

看，行数一样的 Python 代码甚至可读性比伪代码还要好吧？但它真的可以运行，结果如下：

```
>>>quicksort([9,8,4,5,32,64,2,1,0,10,19,27])
[0,1,2,4,5,8,9,10,19,27,32,64]
```

所以，综合这个例子来说，Pythonic 也许可以定义为：充分体现 Python 自身特色的代码风格。接下来就看看这样的代码风格在实际中是如何体现的。

（2）代码风格

对于风格，光说是没有用的，最好是通过例子来看看，因为例子看得见，会显得更真实。下面以语法、库和应用程序为例给大家介绍。

在语法上，代码风格要充分表现 Python 自身特色。举个最常见的例子，在其他的语言（如 C 语言）中，两个变量交换需要如下的代码：

```
int a = 1, b = 2;
int tmp = a;
a = b;
b = tmp;
```

利用 Python 的 packaging/unpackaging 机制，Pythonic 的代码只需要以下一行：

⊖ http://en.wikipedia.org/wiki/Quicksort。

```
a, b = b, a
```

还有，在遍历一个容器的时候，类似其他编程语言的代码如下：

```
length = len(alist)
i = 0
while i < length:
    do_sth_with(alist[i])
    i += 1
```

而 Pythonic 的代码如下：

```
for i in alist:
    do_sth_with(i)
```

灵活地使用迭代器是一种 Python 风格。又比如，需要安全地关闭文件描述符，可以使用以下 with 语句：

```
with open(path, 'r') as f:
    do_sth_with(f)
```

通过上述代码的对比，能让大家清晰地认识到 Pythonic 的一个要求，就是对 Python 语法本身的充分发挥，写出来的代码带着 Python 味儿，而不是看着像 C 语言代码，或者 Java 代码。

应当追求的是充分利用 Python 语法，但不应当过分地使用奇技淫巧，比如利用 Python 的 Slice 语法，可以写出如下代码：

```
a = [1,2,3,4]
c = 'abcdef'
print a[::-1]
print c[::-1]
```

如果不是同样追求每一个语法细节的“老鸟”，这段代码的作用恐怕不能一眼就看出来。实际上，这个时候更好地体现 Pythonic 的代码是充分利用 Python 库里 reversed() 函数的代码。

```
print list(reversed(a))
print list(reversed(c))
```

（3）标准库

写 Pythonic 程序需要对标准库有充分的理解，特别是内置函数和内置数据类型。比如，对于字符串格式化，一般这样写：

```
print 'Hello %s!' % ('Tom',)
```

其实 %s 是非常影响可读性的，因为数量多了以后，很难清楚哪一个占位符对应哪一个实参。所以相对应的 Pythonic 代码是这样的：

```
print 'Hello %(name)s!' % {'name': 'Tom'}
```

这样在参数比较多的情况下尤其有用。

```
# 字符串
value = {'greet': 'Hello world', 'language': 'Python'}
print '%(greet)s from %(language)s.' % value
```

% 占位符来自于大家的先验知识，其实对于新手而言，有点“莫名其妙”，所以更具有 Pythonic 风格的代码如下：

```
print '{greet} from {language}.'.format(greet = 'Hello world', language = 'Python')
```

str.format() 方法非常清晰地表明了这条语句的意图，而且模板的使用也减少了许多不必要的字符，使可读性得到了很大的提升。事实上，str.format() 也成了 Python 最为推荐的字符串格式化方法，当然也是最 Pythonic 的。

（4）Pythonic 的库或框架

编写应用程序的时候的要求会更高一些。因为编写应用程序一般需要团队合作，那么可能你编写的那一部分正好是团队的另一成员需要调用的接口，换言之，你可能正在编写库或框架。

程序员利用 Pythonic 的库或框架能更加容易、更加自然地完成任务。如果用 Python 编写的库或框架迫使程序员编写累赘的或不推荐的代码，那么可以说它并不 Pythonic。现在业内通常认为 Flask 这个框架是比较 Pythonic 的，它的一个 Hello world 级别的用例如下：

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello():
    return "Hello world!"

if __name__ == "__main__":
    app.run()
```

稍有编程经验的人都可以通过上例认识到利用 Python 编程极为容易这一事实。一个 Pythonic 的框架不会对已经通过惯用法完成的东西重复发明“轮子”，而且它也遵循常用的 Python 惯例。创建 Pythonic 的框架极其困难，什么理念更酷、更符合语言习惯对此毫无帮助，事实上这些年来优秀的 Python 代码的特性也在不断演化。比如现在认为像 generators 之类的特性尤为 Pythonic。

另一个有关新趋势的例子是：Python 的包和模块结构日益规范化。现在的库或框架跟随了以下潮流：

- 包和模块的命名采用小写、单数形式，而且短小。
- 包通常仅作为命名空间，如只包含空的 __init__.py 文件。

建议 2：编写 Pythonic 代码

如何编写更加 Pythonic 的代码，与定义什么是 Pythonic 一样困难。在这里，只能给出一些经验之谈，希望对大家有所帮助。

（1）要避免劣化代码

与优化代码对应，劣化代码就是一开始写出来就是不合理的代码，比如不合适的变量命名等。通常有以下几个值得注意的地方：

1）**避免只用大小写来区分不同的对象。**如 a 是一个数值类型变量，A 是 String 类型，虽然在编码过程中很容易区分二者的含义，但这样做毫无益处，它不会给其他阅读代码的人带来多少便利。

2）**避免使用容易引起混淆的名称。**容易引起混淆的名称的使用情形包括：重复使用已经存在于上下文中的变量名来表示不同的类型；误用了内建名称来表示其他含义的名称而使之在当前命名空间被屏蔽；没有构建新的数据类型的情况下使用类似于 element、list、dict 等作为变量名；使用 o（字母 O 小写的形式，容易与数值 0 混淆）、l（字母 L 小写的形式，容易与数值 1 混淆）等作为变量名。因此推荐变量名与所要解决的问题域一致。有如下两个示例，示例二比示例一更好。

示例一：

```
>>> def funA(list,num):
...     for element in list:
...             if num==element:
...                     return True
...             else:
...                     pass
...
```

示例二：

```
>>> def find_num(searchList,num):
...     for listValue in searchList:
...             if num==listValue:
...                     return True
...             else:
...                     pass
...
```

3）**不要害怕过长的变量名。**为了使程序更容易理解和阅读，有的时候长变量名是必要的。不要为了少写几个字母而过分缩写。下例是一个用来保存用户信息的字典结构，变量名 person_info 比 pi 的可读性要强得多。

```
>>> person_info={'name':'Jon','IDCard':'200304','address':'Num203,Monday Road',
    'email':'test@gail.com'}
```