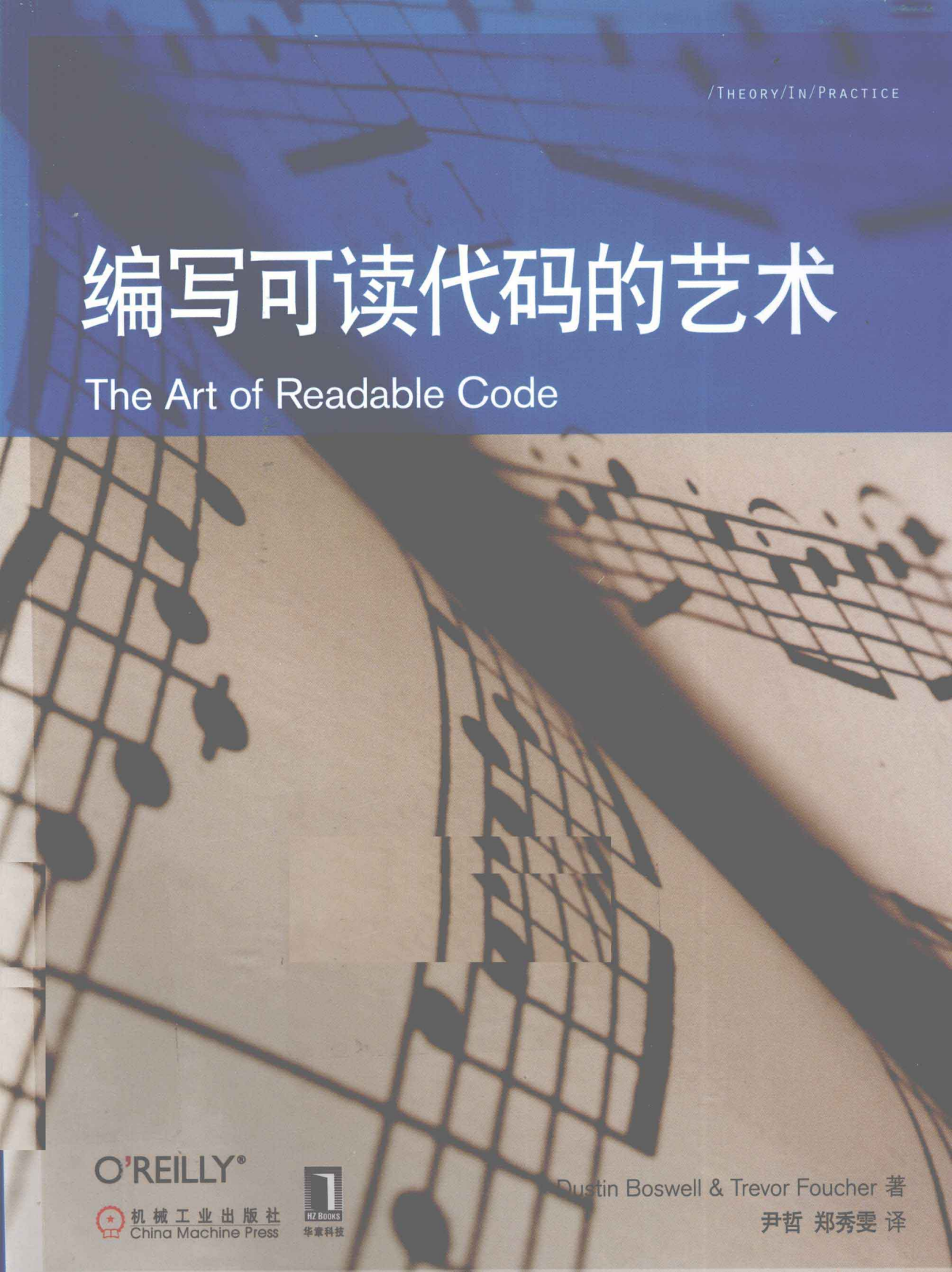
/THEORY/IN/PRACTICE

编写可读代码的艺术

The Art of Readable Code



Dustin Boswell & Trevor Foucher 著
尹哲 郑秀雯 译

O'REILLY®
机械工业出版社 China Machine Press
HZ BOOKS 华章科技

编写可读代码的艺术

Dustin Boswell & Trevor Foucher 著

尹哲 郑秀雯 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

O'Reilly Media, Inc.授权机械工业出版社出版

机械工业出版社

图书在版编目（CIP）数据

编写可读代码的艺术/（美）鲍斯维尔（Boswell, D.），富歇（Foucher, T.）著；尹哲，郑秀雯译. －北京：机械工业出版社，2012.7
（O'Reilly精品图书系列）
书名原文：The Art of Readable Code
ISBN 978-7-111-38544-8

I. 编… II. ①鲍… ②富… ③尹… ④郑… III. 代码－程序设计 IV. TP311.11
中国版本图书馆CIP数据核字（2012）第109081号

北京市版权局著作权合同登记
图字：01-2012-1275号

书　　名/ 编写可读代码的艺术
书　　号/ ISBN 978-7-111-38544-8
责任编辑/ 谢晓芳
封面设计/ Susan Thompson，张健
出版发行/ 机械工业出版社
地　　址/ 北京市西城区百万庄大街22号（邮政编码100037）
印　　刷/ 北京京北印刷有限公司
开　　本/ 178毫米×233毫米　16开本　12.25印张
版　　次/ 2012年7月第1版　2012年7月第1次印刷
定　　价/ 59.00元（册）

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
购书热线：(010) 68326294; 88379649; 68995259
投稿热线：(010) 88379604
读者信箱：hzjsj@hzbook.com

O'Reilly Media, Inc.介绍

O'Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始，O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了Make杂志，从而成为DIY革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

“O'Reilly凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照Yogi Berra的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去Tim似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

推荐序

这是一本关注编码细节的书。或许你会认为本书所讲皆为小道，诸如方法命名、变量定义、语句组织、任务分解等内容，俱是细枝末节，微不足道。然而，对于一个整体的软件系统而言，既需要宏观的架构决策、设计与指导原则，也必须重视微观的代码细节。正如作文，提纲主旨是文章的根与枝，但一词一句，也需精雕细作，才能立起文章的精气神。所谓“细节决定成败”，在软件历史中，有许多影响深远的重大失败，其根由往往是编码细节出现了疏漏。

我一直坚持“代码即架构”的观点，正如小说需要角色来说话一般，软件系统的质量好坏，归根结底还是需要代码来告知。代码的优劣不仅直接决定了软件的质量，还将直接影响软件成本。Yourdon和Constantine在其著作《Structured Design》中写道：软件成本由开发成本与维护成本组成，而往往维护成本要远高于开发成本。这其中耗费的主要成本就是由于理解代码和修改代码造成的。正如本书的书名所表示的含义，好的代码常常是可阅读的，要做到这一点，则近似于一种艺术之美了。

与本书相似的一本书是Robert C. Martin的《Clean Code》。该书在业界已经得到了广泛赞誉。如果你还在为写出“丑陋”的代码而烦恼，必须阅读该书。它带给你的冲击，好似《阿凡达》那无与伦比的3D电影带给你感官上的震撼。在某种程度上，本书几乎可以与《Clean Code》比肩。或许本书在深度上与《Clean Code》相比还有所不及，但在内容广度上，却远远超过了《Clean Code》。因为它关注编码本身，所以并不局限于某一种语言，而是列举了大量C++、Python、JavaScript和Java代码，涵盖了主流的静态语言和动态语言。这就使得本书的内容具有更强的普适性。

本书给出了许多改善编码质量的技巧，尤其它结合了大量真实案例，给出了具体的代码片段，并从正反两面对案例进行分析，这就使得作者的讲解不再流于空洞，既让人信服，又有助于读者理解。例如，在讲解命名如何表达意图时，作者给出了Google代码中的一个反面教材。在Google的一段代码中定义了一个宏，用于禁止“邪恶”的构造函数：

```
class ClassName {
 private:
  DISALLOW_EVIL_CONSTRUCTORS(ClassName);

 public:
  ...
};
```

宏的定义如下：

```
#define DISALLOW_EVIL_CONSTRUCTORS(ClassName) \
  ClassName(const ClassName&); \
  void operator=(const ClassName&);
```

这个宏禁止了构造函数和Copy构造函数（即“=”操作）。然而从宏的名称来看，这个含义是不明确的，会让读者认为仅仅禁用了构造函数。只需要改个名字，意图就可以变得更加清晰：

```
#define DISALLOW_COPY_AND_ASSIGN(ClassName) [...]
```

书中各章的案例非常翔实，并且总是先给出糟糕的版本，逐步分析推导，最后给出好的实现，作为直观鲜明的对照。为避免读者陷入相对独立而散乱的小案例中，作者又另辟一章内容，讲解了一个完整的案例Minute/Hour Counter。首先从问题域的提出开始，分析了接口的设计与实现，深入剖析了实现方法的命名乃至注释，抽丝剥茧，条分缕析。作者的分析显得好整以暇，有条不紊，先后给出了3个解决方案，渐进地对编码实现进行了改善，使之在性能、灵活性上都有了很好的改观，类的职责更为清晰，代码结构简洁而又易于理解。最后，作者还给出了3个解决方案的比较，从代码行数、时间复杂度、内存消耗和准确率4个因素出发，全面权衡了各个解决方案的优劣，以此来印证作者在本书中一直推崇的编码技巧。

本书作者并不满足于通过文字和代码来彰显这些技巧的力量，书中附带的漫画插图起到了很好的辅助作用。如果将本书比作一盘精美的佳肴，这些漫画就起到了调料的作用。对于技术书籍来说，这种图文并茂的方式实不多见，它为全书增添了亮色。我们赏阅漫画时的会意一笑，心底其实充满了如遇知音的喜悦。

阅读本书时，那种代码从丑陋到美丽的蜕变总是让人振奋；但是，我们不能满足于结果的获得，而应该享受这个过程。我的建议是，在阅读时，多思考作者给出的反面教材，不要急于了解结果，而应掩卷遐思，分析这段代码的问题，并结合自身经验与能力给出自己的方案。然后再比较作者的方案，两相印证，辨别两个方案各自的优劣之处。最后再仔细阅读作者的分析过程，如此才能更好地理解本书，提升自己的编码技能。

全书200页左右，与那些瀚如烟海的高文大册相比，本书显得轻而薄，但它胜在精专。作者没有囊括所有编码技巧的野心，更没有卖弄地展现自己的设计技巧和博识广学。它的专注可能会因此失去一大部分读者群，但这样的书才是我们程序员真正需要的。希望你能喜欢它！

张逸

ThoughtWorks高级咨询师

译者序

在做IT的公司里，尤其是软件开发部门，一般不会要求工程师衣着正式。在我工作过的一些环境相对宽松的公司里，很多程序员的衣着连得体都算不上（搞笑的T恤、短裤、拖鞋或者干脆不穿鞋）。我想，我本人也在这个行列里面。虽然我现在改行做软件开发方面的咨询工作，但还是改不了这副德性。衣着体面的其中一个积极方面是它体现了对周围人的尊重，以及对所从事工作的尊重。比如，那些研究市场的人要表现出对客户的尊重。而大多数程序员基本上每天主要的工作就是和其他程序员打交道。那么这说明程序员之间就不用互相尊重吗？而且也不用尊重自己的工作吗？

程序员之间的互相尊重体现在他所写的代码中。他们对工作的尊重也体现在那里。

在《Clean Code》一书中Bob大叔认为在代码阅读过程中人们说脏话的频率是衡量代码质量的唯一标准。这也是同样的道理。

这样，代码最重要的读者就不再是编译器、解释器或者电脑了，而是人。写出的代码能让人快速理解、轻松维护、容易扩展的程序员才是专业的程序员。

当然，为了达到这些目的，仅有编写程序的礼节是不够的，还需要很多相关的知识。这些知识既不属于编程技巧，也不属于算法设计，并且和单元测试或者测试驱动开发这些话题也相对独立。这些知识往往只能在公司无人问津的编程规范中才有所提及。这是我所见的仅把代码可读性作为主题的一本书，而且这本书写得很有趣！

既然是“艺术”，难免会有观点上的多样性。译者本身作为程序员观点更加“极端”一些。然而两位作者见多识广，轻易不会给出极端的建议，如“函数必须要小于10行”或者“注释不可以用于解释代码在做什么而只能解释为什么这样做”等语句很少出现在本书中。相反，作者给出目标以及判断的标准。

翻译书是件费时费力的事情，好在本书恰好涉及我感兴趣的话题。但翻译本书有一点点自相矛盾的地方，因为书中相当的篇幅是在讲如何写出易读的英语。当然这里的“英语”大多数的时候只是指“自然语言”，对于中文同样适用。但鉴于大多数编程语言都是基于英语的（至少到目前为止），而且要求很多程序员用英语来注释，在这种情况下努力学好英语也是必要的。

感谢机械工业出版社的各位编辑帮助我接触和完成这本书的翻译。这本译作基本上可以说是在高铁和飞机上完成的（我此时正在新加坡飞往香港的飞机上）。因此家庭的支持是非常重要的。尤其是我的妻子郑秀雯（是的，新加坡的海关人员也对她的名字感兴趣），她是全书的审校者。还有我“上有的老人”和“下有的小孩”，他们给予我帮助和关怀以及不断前进的动力。

尹哲

目录

前言 1

第1章 代码应当易于理解 5

是什么让代码变得“更好” 6

可读性基本定理 7

总是越小越好吗 7

理解代码所需的时间是否与其他目标有冲突 8

最难的部分 8

第一部分 表面层次的改进 9

第2章 把信息装到名字里 11

选择专业的词 12

避免像tmp和retval这样泛泛的名字 14

用具体的名字代替抽象的名字 17

为名字附带更多信息 19

名字应该有多长 22

利用名字的格式来传递含义 24

总结 25

第3章 不会误解的名字 27
例子：Filter() 28
例子：Clip(text, length) 28
推荐用first和last来表示包含的范围 29
推荐用begin和end来表示包含/排除范围 30
给布尔值命名 30
与使用者的期望相匹配 31
例子：如何权衡多个备选名字 33
总结 34

第4章 审美 36
为什么审美这么重要 37
重新安排换行来保持一致和紧凑 38
用方法来整理不规则的东西 40
在需要时使用列对齐 41
选一个有意义的顺序，始终一致地使用它 42
把声明按块组织起来 43
把代码分成“段落” 44
个人风格与一致性 45
总结 46

第5章 该写什么样的注释 47
什么不需要注释 49
记录你的思想 52
站在读者的角度 54
最后的思考——克服“作者心理阻滞” 58
总结 59

第6章 写出言简意赅的注释 60
让注释保持紧凑 61
避免使用不明确的代词 61
润色粗糙的句子 62

精确地描述函数的行为 62
用输入/输出例子来说明特别的情况 63
声明代码的意图 64
“具名函数参数”的注释 64
采用信息含量高的词 65
总结 66

第二部分 简化循环和逻辑 67

第7章 把控制流变得易读 69
条件语句中参数的顺序 70
if/else语句块的顺序 71
?:条件表达式（又名“三目运算符”） 73
避免do/while循环 74
从函数中提前返回 76
臭名昭著的goto 76
最小化嵌套 77
你能理解执行的流程吗 80
总结 81

第8章 拆分超长的表达式 82
用做解释的变量 83
总结变量 83
使用德摩根定理 84
滥用短路逻辑 84
例子：与复杂的逻辑战斗 85
拆分巨大的语句 87
另一个简化表达式的创意方法 88
总结 89

第9章 变量与可读性 91
减少变量 92
缩小变量的作用域 94

只写一次的变量更好 100
最后的例子 101
总结 103

第三部分 重新组织代码 105

第10章 抽取不相关的子问题 107
介绍性的例子：findClosestLocation() 108
纯工具代码 109
其他多用途代码 110
创建大量通用代码 112
项目专有的功能 112
简化已有接口 113
按需重塑接口 114
过犹不及 115
总结 116

第11章 一次只做一件事 117
任务可以很小 119
从对象中抽取值 120
更大型的例子 124
总结 126

第12章 把想法变成代码 127
清楚地描述逻辑 128
了解函数库是有帮助的 129
把这个方法应用于更大的问题 130
总结 133

第13章 少写代码 135
别费神实现那个功能——你不会需要它 136
质疑和拆分你的需求 136
保持小代码库 138

熟悉你周边的库 139
例子：使用Unix工具而非编写代码 140
总结 141

第四部分 精选话题 143

第14章 测试与可读性 145
使测试易于阅读和维护 146
这段测试什么地方不对 146
使这个测试更可读 147
让错误消息具有可读性 150
选择好的测试输入 152
为测试函数命名 154
那个测试有什么地方不对 155
对测试较好的开发方式 156
走得太远 158
总结 158

第15章 设计并改进“分钟/小时计数器” 160
问题 161
定义类接口 161
尝试1：一个幼稚的方案 164
尝试2：传送带设计方案 166
尝试3：时间桶设计方案 169
比较三种方案 173
总结 174

附录 深入阅读 175

前言

我们曾经在非常成功的软件公司中和出色的工程师一起工作，然而我们所遇到的代码仍有很大的改进空间。实际上，我们曾见到一些很难看的代码，你可能也见过。

但是当我们看到写得很漂亮的代码时，会很受启发。好代码会很明确告诉你它在做什么。使用它会很有趣，并且会鼓励你把自己的代码写得更好。

本书旨在帮助你把代码写得更好。当我们说“代码”时，指的就是你在编辑器里面要写的一行一行的代码。我们不会讨论项目的整体架构，或者所选择的设计模式。当然那些很重要，但我们的经验是程序员的日常工作的大部分时间都花在一些“基本”的事情上，像是给变量命名、写循环以及在函数级别解决问题。并且这其中很大的一部分是阅读和编辑已有的代码。我们希望本书对你每天的编程工作有很多帮助，并且希望你把本书推荐给你团队中的每个人。

本书内容安排

这是一本关于如何编写具有高可读性代码的书。本书的关键思想是**代码应该写得容易理解**。确切地说，使别人用最短的时间理解你的代码。

本书解释了这种思想，并且用不同语言的大量例子来讲解，包括C++、Python、JavaScript和Java。我们避免使用某种高级的语言特性，所以即使你不是对所有的语言都了解，也能很容易看懂。（以我们的经验，反正可读性的大部分概念都是和语言不相关的。）

每一章都会深入编程的某个方面来讨论如何使代码更容易理解。本书分成四部分：

表面层次上的改进

命名、注释以及审美——可以用于代码库每一行的小提示。

简化循环和逻辑

在程序中定义循环、逻辑和变量，从而使得代码更容易理解。

重新组织你的代码

在更高层次上组织大的代码块以及在功能层次上解决问题的方法。

精选话题

把"易于理解"的思想应用于测试以及大数据结构代码的例子。

如何阅读本书

我们希望本书读起来愉快而又轻松。我们希望大部分读者在一两周之内读完全书。

章节是按照"难度"来排序的：基本的话题在前面，更高级的话题在后面。然而，每章都是独立的。因此如果你想跳着读也可以。

代码示例的使用

本书旨在帮助你完成你的工作。一般来说，可以在程序和文档中使用本书的代码。如果你复制了代码的关键部分，那么你就需要联系我们获得许可。例如，利用本书的几段代码编写程序是不需要许可的。售卖或出版O'Reilly书中示例的D-ROM需要我们的许可。引用本书回答问题以及引用示例代码不需要我们的许可。将本书的大量示例代码用于你的产品文档中需要许可。

如果你在参考文献中提到我们，我们会非常感激，但并不强求。参考文献通常包括标题、作者、出版社和ISBN。例如："《The Art of Readable Code》by Dustin Boswell, and Trevor Foucher.©2012 Dustin Boswell, and Trevor Foucher，978-0-596-80229-5。"

如果你认为对代码示例的使用已经超出以上的许可范围，我们很欢迎你通过*permissions@oreilly.com*联系我们。

联系我们

有关本书的任何建议和疑问，可以通过下列方式与我们取得联系：

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街2号成铭大厦C座807室（100035）
奥莱利技术咨询（北京）有限公司

我们会在本书的网页中列出勘误表、示例和其他信息。可以通过*http://oreilly.com/product/9780596802301.do*访问该页面。

要评论或询问本书的技术问题，请发送邮件到：

bookquestions@oreilly.com

有关我们的书籍、会议、资源中心以及O'Reilly网络，可以访问我们的网站：

http://www.oreilly.com

http://www.oreilly.com.cn

在Facebook上联系我们：*http://facebook.com/oreilly*

在Twitter上联系我们：*http://twitter.com/oreillymedia*

在You Tube上联系我们：*http://youtube.com/oreillymedia*

致谢

我们要感谢那些花时间审阅全书书稿的同事，包括Alan Davidson、Josh Ehrlich、Rob Konigsberg、Archie Russell、Gabe W.，以及Asaph Zemach。如果书里有任何错误都是他们的过失（开玩笑）。

我们感激那些对书中不同部分的草稿给了具体反馈的很多审阅者，包括Michael Hunger、George Heinenman以及Chuck Hudson。

我们还从下面这些人那里得到了大量的想法和反馈：John Blackburn、Tim Dasilva、Dennis Geels、Steve Gerding、Chris Harris、Josh Hyman、Joel Ingram、Erik Mavrinac、Greg Miller、Anatole Paine和Nick White。

感谢O'Reilly团队无限的耐心和支持，他们是Mary Treseler（编辑）、Teresa Elsey（产品编辑）、Nancy Kotary（文字编辑）、Rob Romano（插图画家）、Jessica Hosman（工具）以及Abby Fox（工具）。还有我们的漫画家Dave Allred，他把我们疯狂的卡通想法展现了出来。

最后，我们想感谢Melissa和Suzanne，他们一直鼓励我们，并给我们创建条件来滔滔不绝地谈论编程话题。