代码结构

CODE Structure

魏家明◎编著

程序员必备
打造完美代码

中国工信出版集团

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

代码结构

魏家明　编著

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书对如何优化代码结构做了深入的探讨，分为5个部分：编程问题与原则、编程格式与风格、让代码更容易读、如何做代码重构和C语言一些要素。

本书对这些部分做了重点的探讨：代码中存在的各种问题，编程时要遵循的原则，编程时要注重的格式、注释和名字，如何让表达式和控制流简单易读，如何消除代码中的重复冗余，如何切分代码和少写代码等。另外，本书还探讨了C语言设计中的一些要素和常见问题。

本书适合具有一定编程基础的人阅读，读者通过阅读本书，可以规范代码设计，提高设计能力，优化代码结构，提高代码的可读性，从而提高代码的各种特性。

图书在版编目（CIP）数据

代码结构/魏家明编著. —北京：电子工业出版社，2016.8
ISBN 978-7-121-29603-1

Ⅰ. ①代… Ⅱ. ①魏… Ⅲ. ①程序语言 Ⅳ. ①TP312

中国版本图书馆CIP数据核字（2016）第179694号

策划编辑：牛平月
责任编辑：张 剑
文字编辑：牛平月
印 刷：北京京科印刷有限公司
装 订：三河市良远印务有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×1 092 1/16 印张：17.5 字数：450 千字
版 次：2016 年 8 月第 1 版
印 次：2016 年 8 月第 1 次印刷
印 数：3 000 册 定价：49.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）88254454。

序言

通过编程语言可以设计两种系统：软件系统和硬件系统。例如，使用 C 语言设计操作系统和应用软件，使用 Verilog 设计 ASIC 芯片和 FPGA 应用。

系统的生命周期包含两个阶段：开发期和运营期。在开发期，公司投入大量的人力和物力用于开发系统，所以总是希望系统能发挥最大的能力；在运营期，为了尽可能地延长系统的运营时间，就需要投入大量的工作进行系统维护。

系统维护一般包括三大类：一是纠正性维护，用来修改系统中存在的错误和缺陷；二是适应性维护，为了让系统适应不断变化的环境，需要对系统进行修改和扩充；三是完善性维护，为了让系统提升性能、扩大功能、拓展应用范围，需要对系统进行扩充和移植。

在这三大类的系统维护工作中，第二类和第三类维护所占的份额最大，约占总维护工作的 80%左右。所以，系统的可维护性是首要考虑的问题，系统的运营过程就是维护系统生命价值的过程。根据调查表明，系统维护成本已占到系统生命周期成本的 70%以上。这表明系统维护的难度越来越大，已成为目前系统开发所面临的最大问题。

影响系统维护的方面有很多：系统架构设计不合理、不灵活，难以修改和扩充，编写代码时不注意代码的内在质量，代码可读性和可理解性差，缺乏相关的文档资料，代码和文档不相符合，员工频繁离职导致工作脱节等。

随着系统开发越来越复杂，就需要采取科学的管理方法和优秀的设计技术，严格把控系统设计的质量，使得系统设计按照有条不紊的方式进行，提高代码编写的生产率，提高代码的可靠性、可读性、可理解性和可维护性，从而降低系统开发和维护的成本。

为了设计易于维护的系统，就要设计出通用、灵活、易于维护的系统架构，系统设计模块化，模块高内聚低耦合，遵守编码风格，优化代码结构，提高代码设计的质量，保证代码的可读性、可理解性、可测试性、可维护性、可移植性等内在特性，从而降低系统的维护难度，延长系统运营的生命周期。

作者根据上面这些开发和维护的需求，对实际代码设计进行分析总结，同时阅读多位编程大师的书籍，查找大量的资料，从而完成了本书的编写。作者在本书中讨论了这些内容：代码中存在的各种问题，编程时要遵循的原则，编程时要注重格式、注释和名字，如何让表达式和控制流简单易读，如何消除重复冗余的代码，如何切分代码和少写代码，C 语言的一些要素等。

读者通过阅读本书，可以规范代码设计，提高设计能力，优化代码结构，提高代码的可读性，从而提高代码的各种特性。

GigaDevice Semiconductor (Beijing) Inc.
北京兆易创新科技股份有限公司
副总经理 曹堪宇博士
2016 年 5 月 12 日

前言

这是一本向编程大师致敬的书，也是一本对代码设计思考的书，还是一本对实际代码设计有帮助的书。

Martin Fowler said：“Any fool can write code that a computer can understand. Good programmers write code that humans can understand。”

有些人花了很大的精力编写代码，但只是满足于代码能工作就行，从未想过让代码有好的结构和可读性，也就不会花费精力调整代码，从而导致代码很糟糕。其实，没什么比糟糕的代码更差劲的了，因为糟糕的代码只会降低代码的可读性，只会拖团队的后腿，只会让系统的成本增加。

系统的成本在于长期维护的成本，这是因为为了尽量降低维护时出现缺陷和错误，就有必要透彻地理解系统在干什么。当系统变得越来越复杂，设计人员就需要花费越来越多的时间来理解系统，但还是可能会有很多的困惑和误解，而代码中存在的各种各样的问题还会加剧设计人员对代码的困惑和误解。

代码设计中存在很多问题，例如命名混乱、名实不副、格式混乱、注释混乱、重复冗余、臃肿庞大、晦涩难懂、过度耦合、滥用变量、嵌套太深、代码混杂、太多警告、过度设计、陈旧腐败等，这些都是在代码结构设计上出了问题，这些问题就像“死气沉沉的沼泽”一样，让设计人员痛苦不堪，艰难地跋涉。

代码设计中存在的各种问题，会影响代码的设计质量，影响代码的灵活性、可读性、可理解性、可维护性、可测试性、可移植性、可重用性等，还会影响设计人员的干劲与活力，甚至影响设计人员间的协作精神。

所以代码设计应该具有良好的代码结构，具有良好的可读性和表达力，能够清晰地表达设计者的意图，同时也是对工作的负责，对团队的尊敬。设计人员在理解代码时花费的时间越少，就越能减少设计缺陷和错误，越能减少维护成本，同时设计者本人也会赢得其他人员的尊敬。

本书针对如何优化代码结构，通过以下 5 个部分进行讨论。

1）编程问题与原则：本部分讨论代码质量、代码问题、人员问题、编程原则和编程之道，因为只有清楚了存在的问题，掌握了编程原则，才有可能写出更好的代码。

2）编程格式与风格：本部分讨论格式优美、注释合理、名字定义等方面，还介绍 Emacs 的使用，这些方面对设计人员非常重要。

3）让代码更容易读：本部分讨论如何让编写出的代码更容易阅读，包括消除警告、精心用变量、表达式易读、控制流易读、设计好函数。

4）如何做代码重构：本部分讨论代码重构的好处和方法，包括消除重复、代码切分、少

写代码、简化代码，还讨论代码生成、代码测试。

5）C 语言一些要素：本部分讨论 C 语言的一些要素，讨论一些容易混淆和出错的地方，还讨论 C 语言的一些好用法。

如果你没有优化代码结构的意愿，那么说再多也没用，当你看见自己混乱的代码时，你也不会感到害臊；当你看见别人优美的代码时，你也不会有任何感觉。

如果你有优化代码结构的意愿，就要注重编码格式、名字定义、代码易读、消除冗余、代码重构等方面的学习，并积极实践这些方法，从一点一滴做起，最后就会获得很好的代码结构。

本书举例主要以 C 语言为主，但也会有 Verilog 和 Perl 的例子。有些人觉得它们是三种完全不同应用的语言，竟然在一本书里描述，可能觉得不可接受，但是代码设计都有共性，都需要注意代码结构，关键在于灵活掌握这些设计原则，那么对任何编程语言你都会游刃有余。

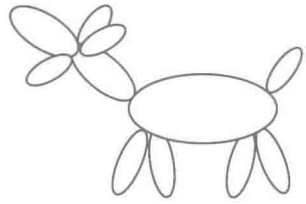
衷心地感谢曹堪宇博士，他在百忙之中为本书题写了序言，他带领我们做系统设计的时候，我深深地被他的精益求精的精神所打动。

衷心地感谢我的朋友燕雪松、张茜歌等人帮我审稿，帮我在书中找到好多的缺陷和错误。

衷心地感谢网上的朋友们，因为我采用了很多的网上资料，感谢你们的无私奉献。

衷心地感谢电子工业出版社的支持，正是由于策划编辑牛平月的密切联系和各位编辑的认真工作，才使得本书得以顺利地与读者见面。

如果您在本书中发现有缺陷或者错误的地方，或者您对本书存有模糊或者疑惑的地方，请通过 QQ 或者邮件与我联系，我的 QQ 号码是 943609120，您的任何反馈都是令人欢迎的。我还建立了一个名为“代码结构”的 QQ 群，群号码是 482433665，欢迎大家加入，共同探讨各种与代码设计有关的各种问题。



魏家明

2016 年 6 月 30 日

目录

第一部分 编程问题与原则

第二部分 编程格式与风格

第三部分 让代码更容易读

第四部分 如何做代码重构

第五部分 C语言一些要素

第一部分

编程问题与原则

有些人花了很大的精力编写代码，但只是满足于让代码能工作就行，从没有想过让代码有好的结构和好的可读性，也就不会花费精力调整代码，从而导致代码很糟糕。其实，没什么比糟糕的代码更差劲的了，因为糟糕的代码只会降低代码的可读性，只会拖团队的后腿，让系统的成本增加。

本部分讨论代码质量、代码问题、人员问题、编程原则和编程之道，因为只有清楚了编程中存在的问题，掌握了编程原则，才有可能写出更好的代码。

第 1 章
美的设计

1.1 美学观点

“程序设计是一门艺术”这句话有两个意思：一方面是说，程序设计像艺术设计一样，深不可测，奥妙无穷；另一方面是说，程序员像艺术家一样，也有发挥创造性的无限空间[14]。

Donald Knuth 认为“计算机科学”不是科学，而是一门艺术。它们的区别在于：艺术是人创造的，而科学不是；艺术是可以无止境提高的，而科学不能；艺术创造需要天赋，而科学不需要。所以 Donald Knuth 把他的 4 卷本巨著命名为《计算机程序设计艺术》（*The Art of Computer Programming*）。

Donald Knuth 不仅是计算机学家、数学家，而且是作家、音乐家、作曲家、管风琴设计师。他的独特的审美感使他拥有广泛的兴趣、多方面的造诣，他的传奇般的创造力也是源于这一点。对于 Donald Knuth 来说，衡量计算机程序是否完整的标准不仅在于它是否能够运行，他认为计算机程序应该是雅致的，甚至可以说是美的。计算机程序设计应该是一门艺术，一个算法应该像一段音乐，而一个好的程序就应该像一部优秀的文学作品。

Bjarne Stroustrup，C++语言发明者，说“我喜欢优雅高效的代码。代码逻辑应当直截了当，让缺陷难以隐藏；应当减少依赖关系，使之便于维护；应当依据分层战略，完善错误处理；应当把性能调至最优，省得引诱别人做没规矩的优化，搞出一堆混乱来”。他特别使用“优雅”这个词来说明“令人愉悦的优美、精致和简单”[18]。

一个人的美学观点会影响他的程序设计，因为 Knuth 有这么多的艺术爱好，所以他把程序设计看成艺术设计，在程序设计中要体现出程序的美。同样，当 Bjarne Stroustrup 编写优雅且高效的代码的时候，他也是在程序设计中寻求美。

有些人的美学观点是简单和谐、整洁有序；有些人的美学观点是宏大华丽、空洞无味；还有些人的美学观点是乱七八糟、凑合了事。你的美学观点是什么呢？有些人颇为自负，自我感觉良好，以为领悟到了编程的真谛，看到代码可以运行，就洋洋得意，可是却对自己造成的混乱熟视无睹。那堆“可以运行”的程序，就在眼皮底下慢慢腐坏，然后废弃扔掉。

所以程序设计要遵从自己的方法论，要体现自己的奇思妙想，要让设计有更长的生命力，而不是最后沦为豆腐渣工程。程序设计应该像设计艺术作品一样，要寻求美、设计美，要精雕细琢、仔细打磨，要经历痛苦与无奈，还要经历快乐与自得。

1.2 代码可读

在人们眼中，程序员大部分时间是用在编写和调试代码上，但是其实程序员大部分时间是用在阅读和理解代码上，包括自己写的代码，也包括别人写的代码。为什么要花这么多时

间阅读和理解代码呢？因为一个系统不是从零开始设计的，代码有各种来源，有自己设计的，有别人设计的，也有从别的系统里拿过来的。一方面要让自己设计的代码正常工作，另一方面要消化吸收其他的代码。这就要求程序员不停地测试，不停地“Debug”，不停地阅读理解代码。因此代码要具有很好的可读性，才能减少阅读理解的时间。

如果有人对你说：“Hi，过来帮我看一下这个问题出在哪里。”然后他给你展现的是一堆乱七八糟的代码。你心里怎么想，你是不是在想：“这位老兄是在逗我玩吗？竟然好意思拿这样的代码给我看。”在阅读代码的过程中，人们说脏话的频率是衡量代码内在质量的唯一标准，所以代码最重要的读者不是编译器、解释器或者电脑，而是人，好的代码应该能让人快速阅读并理解。

现实总是那么不尽如人意，很多人的代码中都有着一些阻碍代码可读的问题，这些问题包括：命名混乱、名实不副、格式混乱、注释混乱、重复冗余、臃肿庞大、晦涩难懂、过度耦合、滥用变量、嵌套太深、代码混杂、太多警告、过度设计、陈旧腐败等，反正就是各种各样的混乱。这些问题是现实存在的，是显而易见的，但是对于这些问题很多人都毫不在意。

当你意识到你的代码中存在的问题后，你才能想办法规范自己的代码设计，提高代码的可读性，从而提高代码的可理解性、可扩展性、可测试性、可维护性、可移植性，从而你才会有这种感觉：“恢恢乎其于游刃必有余地矣……提刀而立，为之四顾，为之踌躇满志，善刀而藏之。”

1.3　适用范围

本书主要探讨这些问题：编程问题与原则、编程格式与风格、让代码更容易读、如何做代码重构和 C 语言一些要素。

本书举例主要以 C 语言为主，但是并不局限于 C 语言，也会举一些 Perl 和 Verilog 的例子。有些人会觉得这三种语言跨度太大了，包含了软件语言、脚本语言和硬件语言。但是编程语言和编程方法都具有共性，本书所探讨的代码结构也具有共性。例如名字定义、格式优美、消除冗余、代码切分，C 语言可以这么做，Verilog 可以这么做，其他语言同样也可以这么做。

编写代码不只是用来设计软件，还包括用 Verilog 设计硬件电路，也包括用 MatLab 做科学仿真，所以本书很少使用“软件”这个词，而是使用“系统”表示所有可以用代码实现的东西，这样更具有实在的意义；所以本书所说的代码设计是广义的，所讨论的原则是普遍适用的；所以本书的程序员是广义的，只要是编写代码的人，都可以称为程序员。

C 和 Perl 程序构成的基本单位是函数，Verilog 程序构成的基本单位是模块，但是也夹杂着一些函数和任务。本书以讨论函数为主，以讨论模块为辅，因为设计思想是相同的，所以很多提到函数的地方也适用于模块。

本书不只适用于软件开发人员，也适用于所有编写代码的人员。本书讨论的方法简单明了、切实可行，会对你有所帮助，会帮助你规范代码设计，优化代码结构，提高代码的可读性。

第 2 章
代码质量

对于一个完整的系统来说，需求和架构是宏观的，设计和测试是微观的。只有做到宏观和微观的协调统一，才能做出一个完美的系统，否则最后只能是一个烂尾的系统。系统的质量归根结底反映到代码的质量上，代码的质量不仅决定系统的质量，还直接影响系统的成本。代码同时拥有外在和内在的质量特性，我们在设计时需要着重关注它们。

2.1 外在特性

外在特性指的是该系统的用户所能感受到的部分，包括下列内容：[19]

1）正确性：指系统在规范、设计和实现方面错误的稀少程度。
2）易用性：指系统在学习和使用上的容易程度。
3）高效率：指系统是否能尽可能少地占用资源，包括内存使用和执行时间。
4）可靠性：指系统是否能够可靠地运行，应该有很长的平均无故障时间。
5）完整性：指系统既能确保数据能够正确地访问，又能阻止不正确和未经授权的访问。
6）适应性：指系统在不做修改的情况下，能够在其他应用或者环境中使用的范围。
7）精确性：指系统输出结果的误差程度，尤其当输出结果是数量值的时候。
8）健壮性：指系统在接收无效输入或者处于压力环境时继续正常运行的能力。

在以上这些特性中，有一些特性是相互重叠的，但它们都有不同的含义，并且在不同的场合下，重要性也有所不同。

外在特性是用户关心的唯一特性，用户只会关心系统是否正确运行，只会关心系统是否容易使用，而不会关心代码是否具有很好的可读性，不会关心代码修改起来是否很容易，不会关心代码是否具有很好的结构。

2.2 内在特性

内在特性指的是系统内在的质量特性，包括下列内容：[19]

1）可读性：指阅读并理解代码的难易程度，尤其是在细节语句的层次上。
2）可理解性：指在系统组织和细节语句的层次上理解整个系统的难易程度。与可读性相比，可理解性对系统提出了更高的内在一致性要求。
3）可维护性：指是否能够很容易地对系统修改缺陷和错误，提高运行性能，增加新的功能。
4）可测试性：指的是可以进行何种程度的单元测试或者系统测试，以及在何种程度上验

证系统是否符合要求。

5）可移植性：指一个系统是为特定用途或者环境而设计的，那么当该系统被用于其他用途或者环境的时候，需要对系统修改的难易程度。

6）可重用性：指系统的某些部分可被应用到其他系统的程度，以及此项工作的难易程度。

内在特性和外在特性并不能完全割裂开来，因为在某些层次上，内在特性会影响某些外在特性。如果一个系统无法从内部理解或者维护，那么其缺陷也是很难修正的，而这又会影响其正确性和可靠性等外在特性。如果一个系统很刻板，无法根据用户的需要进行修改，那么就会影响其可用性这一外在特性。

我们需要清楚的是：对于一个系统需要什么样的特性，这些特性间在什么时候会发生什么样的相互作用。让所有的特性都表现得尽善尽美是很困难的，需要根据这些相互竞争的目标寻找出一套最优化的解决方案。

2.3 一个故事

20 世纪 80 年代末，有家公司编写了一个很流行的应用软件，许多专业人士都买来用。但是，后来的发布周期就开始拉长，缺陷总是不能修复，装载时间越来越久，崩溃的概率也越来越大。Martin 至今还记得他在某天沮丧地关掉这个软件之后，从此就不再用它。在那之后不久，该公司就关门大吉了[18]。

20 年后，Martin 见到那家公司的一位早期雇员，问他当年发生了什么事。他的回答让 Martin 越发地感到恐惧。原来当时他们赶着推出产品，代码写得乱七八糟，特性越加越多，代码也越来越乱，最后他们再也没有办法管理这些代码。所以，是糟糕的代码毁了这家公司。

在某些公司中，代码质量被认为是次要目标，快速而糟糕（quick and dirty）的编程成了普遍的现象，那些胡乱堆砌劣质代码并能“快速完成工作”的程序员受到公司的重视，而那些编写出高质量代码并在发布之前完成工作的程序员则受到公司的轻视。既然这样，谁还会把代码质量当作他们的头等大事呢？

如果你是一位有经验的程序员，肯定遭遇过这种困境，我们有专用的词用来形容这种困境：沼泽（wading）。当我们阅读修改调试那些糟糕代码的时候，我们就如同在杂草丛生、水潭密布、泥沼暗藏、望不到尽头的沼泽地里跋涉，我们拼命地想找到出路，期望有点什么线索能启发我们到底发生了什么事情，但是目光所及，只是越来越多的死气沉沉的代码。

2.4 提升质量

代码的内在质量要靠编程规范来保证，但是编程规范在很多公司里无人问津。很多人一边在为糟糕的代码烦恼，却又一边继续编写着糟糕的代码。程序员之间的相互尊重体现在他们所编写的代码上，他们对工作的负责也体现在这里，而糟糕的代码能体现出对同事的尊重和对工作的负责吗？

我们都希望编写出高质量的代码，通往高质量代码的方法有两种：一种是事后修补，另一种是从开始就关注质量。

前者属于常见的工作方法，就是传统的瀑布式开发方法，它要求进行大量的测试，当你以为已经完工了，到头来你会发现还有许多工作要做，需要反反复复地分析、设计、编码和测试。

后者是一种可以让你对系统有信心、可以长年维护系统的方法，它要求根据系统的外在和内在特性，明确定义出代码质量的目标，在设计和编码中要遵循众多规则，同时不断地进行着代码重构。