TURING

成长计划

系统程序员

李先静 编著

结合代码详细讲解程序开发方法

汇集丰富的软件开发思想

CSDN专家全新力作

人民邮电出版社
POSTS & TELECOM PRESS

成长计划

系统程序员

李先静 编著

人民邮电出版社
北 京

图书在版编目（CIP）数据

系统程序员成长计划 / 李先静编著. -- 北京 : 人民邮电出版社, 2010.4
ISBN 978-7-115-22401-9

Ⅰ. ①系… Ⅱ. ①李… Ⅲ. ①软件开发 Ⅳ. ①TP311.52

中国版本图书馆CIP数据核字(2010)第031151号

内 容 提 要

本书以生动的语言和丰富的代码示例，运用一些相对简单的例子分析开发系统程序中可能遇到的各种问题。作者把数年的开发经验和阅读大量书籍的体会，结合他在培训新员工过程中所积累的培养方法，融会贯通在这 12 章的内容中。书中介绍了链表、数组、栈、队列和散列表等基础数据结构，也介绍了并发、同步和内存管理等系统程序中常需注意的问题，还讲解了文本处理器等具体应用程序的设计方法。

本书是初涉系统程序开发领域的人不可多得的一本参考书。书中体现的思想对于其他各种软件开发人员、相关专业的在校学生以及软件开发爱好者都有启发意义。

系统程序员成长计划

◆ 编　　著　李先静
责任编辑　傅志红
执行编辑　傅尔也

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
中国铁道出版社印刷厂印刷

◆ 开本：800×1000　1/16
印张：17.5
字数：413千字　　2010年4月第1版
印数：1－3 000册　　2010年4月北京第1次印刷

ISBN 978-7-115-22401-9

定价：45.00元

读者服务热线：(010)51095186　印装质量热线：(010)67129223
反盗版热线：(010)67171154

李先静

CSDN 开源专家，有着十年 Linux 开发经验、五年手机开发经验。擅长嵌入式程序员培训，软件架构设计和技术写作。近几年负责 Broncho Linux 智能手机项目，致力于基于 Linux 的嵌入式系统的学习和研究。其 CSDN 博客（http://blog.csdn.net/absurd）连续三年被 CSDN 提名为最有价值的技术博客（MVB），他先后发表了近 500 篇技术类博文，博客文章被各大技术网站转载。在《程序员》杂志上发表过多篇文章。

站在巨人的肩上

Standing on Shoulders of Giants

www.turingbook.com

序

写作背景

在经历过几个大型的、失败的项目之后，我终于认识到了：没有什么比高素质的程序员更能决定项目的成功。无论采用什么开发过程，什么编程语言和开发工具，离开了高素质的程序员，都是白费力气。毫无疑问，人是软件开发中最重要的因素。但并非每个人都重要，也不是什么样的人都重要，在软件开发中，只有那些高素质的程序员和那些对项目有突出贡献的人才是重要的。

不过高素质的程序员并不多见，所以从我开始带人起，就一直在思考团队成员培养的问题。我做过很多尝试，从小组内学习到整个部门一起上大课，最后又回到对个人做单独的辅导；从通过Code Review（代码评审）做现场教育到制定一个宏伟的培训计划，最后又回到一个朴素的培训过程。其中遇到了很多问题，开始是培训不够系统，效果不甚理想，后来又因为计划过于“宏伟”而无法实施，等到最后形成一个朴素的、切实可行的培训方案，已经经过了好几年时间，直到去年，整个计划才趋于完善。我把这个培训计划称为系统程序员成长计划，而这正是我在本书中所要介绍的。

培训内容不是来源于某本书，毕业八年来，我坚持不懈地阅读有关书籍，所读过的300多本不同类型的著作装满了家中的7个大储物箱，而这些著作囊括了大部分经典的IT图书。当然培训的内容也不是全部源于书本，这几年我在开发开源软件的过程中所收获的感悟和积累的经验也融入其中。我的培训计划并不是要阐述什么高深的道理，相反，我这本书主要是针对应届毕业生和业余爱好者写的，目的就是为了让初学者进阶为一个专业的程序员。

为什么把这个培训计划叫做“系统程序员成长计划”，而不是“程序员成长计划”呢？程序员的范围太广了，虽然软件开发有很多相似之处，但是隔行如隔山，比如对于目前炙手可热的Web开发，我完全是外行。为了不造成“想什么都讲一点，结果是什么都没有讲清楚”的尴尬，我得把培训计划限定在我熟悉的范围之内。而所谓系统程序员，是指从事操作系统内核、DBMS、GUI系统、基础函数库、应用程序框架、编译器和虚拟机等基础软件开发的程序员。不过虽说这个培训计划叫“系统程序员成长计划”，其实这些内容同样适用于桌面软件和智能手机软件开发，

对其他软件开发也多少会有一些启发作用。

第一次在温伯格的《咨询的奥秘》[①]中看到草莓酱定律和果酱定律[②]时，我觉得非常有意思。当然《系统程序员成长计划》也无法脱离草莓酱定律的魔法，利用本书所讲的内容，我手把手地教了十多个同事，取得了良好的效果。但当有成百上千的读者读这些文章时，我不敢期望有同样的效果。不过在果酱定律的鼓励下，我相信本书中至少有部分内容的价值不会因为读者群的增大而消失，所以我最终决定写这本书，来分享我这些年来所积累下的经验。

中心思想

软件开发的困难在哪里？对于这个问题，不同的人有不同的答案，同一个人在不同职业阶段也会有不同的答案。作为一个系统程序员来说，我认为软件开发有两大难点。

一是控制软件的复杂度。软件的复杂度越来越高，而人类的智力基本保持不变，如何以有限的智力去控制无限膨胀的复杂度？在经历过几个大型项目，也分析过不少现有的开源软件后，我得出一个结论：单个难题和技术细节我们总是可以搞定的，而所有这些问题出现在一个项目中时，其呈指数增长的复杂度往往让我们束手无策。

二是隔离变化。用户需求在变化，应用环境在变化，新技术不断涌现，所有这些都要求软件开发能够射中移动的目标。即使是开发基础平台软件，在超过几年时间的开发周期之后，需求的变化也是相当惊人的。需求变化并不可怕，关键在于变化对系统的影响，有时这种变化会牵一发而动全身，一点小小的变化都可能对系统造成致命的影响。

为了解决这两个问题，方法学家们几十年来不断努力，他们改进或发明软件的开发过程和设计方法。系统程序员所面对的基础软件通常都是复杂的大型软件，其通用性也要求能容纳更多变化，解决这两个问题也是系统程序员成长计划的主要目标。

文章特色

以引导读者思考为主。培训可以制造合格的程序员，却无法造就一流的高手。因为培训是一个相对被动的过程，很难保证学习效果（我们都知道在大学里听课的效果），所以我不希望本书被视作一本单纯的培训教材，我们要做到变被动为主动，最大限度地提高学习的效果。大多数情况下，我会先提出问题让读者去思考，让读者尝试自行解决问题。能不能解决这个问题其实并不重要，重要的是在思考中提升自己。如果读者在一定时间内找不到解决问题的方法，本书也提供

① 已由清华大学出版社出版。

② 草莓酱定律：面积涂得越大，酱就越薄。它说明如果被过分引申，任何深刻的寓意都会被减弱。果酱定律：只要还有颗粒，酱就永远都不会被涂抹得过薄。说明寓意不会因为过分引申而消失。

了专业程序员的参考解决方案（或许不是最优的）。

以简单的例子讲述复杂的设计方法。我曾经制定过一个宏伟的培训计划，但不幸的是这个计划并没有带来成功的结果，原因很简单：我忘记了我在学会走路之前也曾艰难地爬行过。这次我吸取了教训，用简单的示例来讲述复杂的设计方法，而且不要求读者掌握许多背景知识。书中不会出现复杂的数据结构和算法，也不会引入大型软件来唬人。既包含足够的挑战，不会让读者感到乏味；又一切尽在掌握之中，不会让读者因为挫折而打击积极性。

技术能力与工作态度并重。古人云："德才兼备真君子。"同样，一流的程序员也应该是德才兼修的。当我手把手教别人的时候，我希望他不仅能学会我讲的知识点，更希望他能对我的工作态度和作为程序员的道德素养有所感悟。虽然有些东西只可意会不可言传，但我仍希望大家能成为德才兼备的程序员。

读者群

本书主要是针对初学者写的，这里所说的初学者，包括在校学生、应届毕业生和其他业余爱好者。拿我面试过的应届毕业生来说吧，他们大多数并不具备工作所需要的编程能力，只是对基本理论多少有一些了解。本书中的文章就是为需要提升编程能力的初学者们量身定制的。书中的内容经历了十余人的实践，取得过令人满意的效果：大多数参与过我的培训的人最开始可能连一行代码都写不出，但到了培训结束时，他们一般都能独立开发/维护一些有着几千行代码的小模块。不过学习效果还是要看个人的领悟能力和努力程度，但不管怎样，只要读完这些文章，你都能从中取得不少收获。

如何使用本书

温伯格说过，医生的药方包括需要服用的药物和服药的方法，两者缺一不可。同样的教材，如果使用不同的学习方法，最终效果也有很大差别。那么该如何学习本书中的文章呢？我建议大家先自己想办法去解决文中提出的问题，在思考的过程中可以自己查阅资料，至少经过两三个小时的思考之后，再继续阅读下去，最后再按学到的方法自己独立地将程序写一遍。要记住：**学习编程一定要多写多练，否则效果会大打折扣。**

Enjoy it!

致　谢

感谢我的上司老魏（魏政群），没有他的支持，我不可能写那么多BLOG，没有他的信任，我不可能在broncho团队推行这套培训课程。感谢broncho团队同事的支持，大部分同事都参加了这套培训课程，他们的反馈让我能不断完善这套课程。

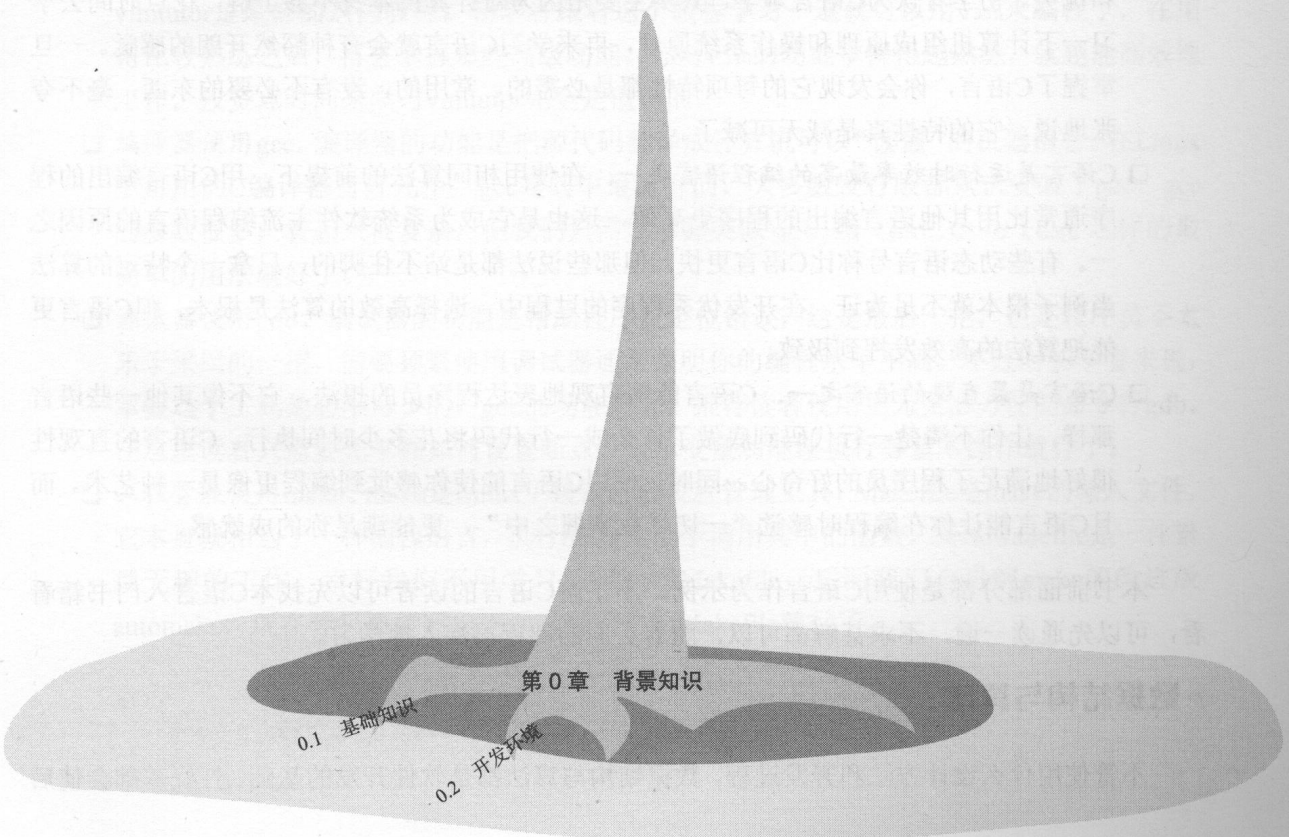
感谢我的老婆欢欢，没有她无微不至的照顾，我不可能有那么多时间去写BLOG和写书，没有她的鼓励支持，我不可能坚持下来。感谢岳父岳母，如果不是他们精心地照顾我的儿子，我不可能有精力去写作。感谢我的父母，他们的声音总是给我无穷的动力。

感谢网友们的支持，特别是Joey.Huang、Dig、haibin.yu和echo几位兄弟长期的支持，是你们的鼓励让我感觉在做一件有意义的事情。

目　录

第0章

背景知识



对于是否应该写这样一章，我犹豫了一段时间，最后考虑到本书主要是针对新手而写的，不应该对读者背景有过多要求，所以还是写了这一章介绍背景知识的内容。其实本章介绍的这些基础知识是每个程序员都必须掌握的。如果你已经了解它们，尽可以放心大胆地跳过本章。如果你是新手，那么请认真学习本章所讲述的内容。

0.1 基础知识

C语言

千万不要认为C语言过时了，它始终是开源社区，特别是系统软件和嵌入式系统开发中的王者，在可以预见的未来，C语言还将持续不断地焕发出生命力。有些不了解软件开发的人也许会认为C语言不适合开发大型软件，这种看法是不对的，事实上，操作系统内核、虚拟机、数据库管理系统、图形引擎和Web服务器等大型软件几乎都是用C语言开发的。C语言其实并不适合开发小程序，相较而言，脚本语言更适用于小程序的开发。C语言能经久不衰，自有它的道理。

- C语言是最简单的语言之一。大部分编程语言在刚出现时都以其简洁而获得好评，但几乎都随着时间的推移而变得越来越复杂。不过C语言历经数十年的发展，却始终保持其简洁和优美。初学者认为C语言难学，其实主要是因为对计算机本身不够了解，花点时间去学习一下计算机组成原理和操作系统原理，再来学习C语言就会有种豁然开朗的感觉。一旦掌握了C语言，你会发现它的每项特性都是必需的、常用的，没有不必要的东西，毫不夸张地说，它的特性真是减无可减了。
- C语言是运行时效率最高的编程语言之一。在使用相同算法的前提下，用C语言编出的程序通常比用其他语言编出的程序更高效，这也是它成为系统软件主流编程语言的原因之一。有些动态语言号称比C语言更快，但那些说法都是站不住脚的，只拿一个特定的算法当例子根本就不足为证。在开发优秀程序的过程中，选择高效的算法是根本，但C语言更能把算法的高效发挥到极致。
- C语言是最直观的语言之一。C语言能够直观地表达程序员的想法，它不像其他一些语言那样，让你不清楚一行代码到底做了什么或一行代码将花多少时间执行。C语言的直观性很好地满足了程序员的好奇心。同时，使用C语言能使你感觉到编程更像是一种艺术。而且C语言能让你在编程时感觉“一切尽在掌握之中”，更能满足你的成就感。

本书前面部分都是使用C语言作为示例，不了解C语言的读者可以先找本C语言入门书籍看看，可以先通读一遍，不求甚解都可以，随着后面的课程再深入地学习。

数据结构与算法

不管使用什么设计方法和开发过程，数据结构与算法都是软件开发的基础。打好基础会使后

续的开发工作事半功倍。后继课程也都是这些基本数据结构和算法为中心，讲述如何用这些基本的材料构建大型系统。读者暂时无需精通数据结构和算法，可以先找本书看看，了解一下双向链表、动态数组、队列、堆、栈、散列表、排序和查找的基本原理就行了，后面我们会以这些数据结构为主题反复加以练习。

0.2 开发环境

本书重点讲解软件开发的基础知识，这些知识并不依赖于特定的平台和开发环境，读者可以根据自己的喜好来选择，但我们推荐读者使用下列开发环境。

- 操作系统使用Linux。Linux是最适合程序员使用的操作系统，它是开源的，有多种不同的发行版可供免费使用，而这些发行版大多默认安装有开发工具。全面学习Linux需要一本专门的书，不过即便你从来没接触过Linux，也犯不着惊慌失措，其实学习本书的内容只需要你花几个小时学会十来个常用的命令就够了，其他有关Linux的内容可以以后慢慢再学。
- 编辑器使用vim。编辑器的功能是创建源文件，也就是把我们编写的代码输入到电脑中。vim和emacs是Linux下最流行的代码编辑器，vim更容易上手，而且功能也很强大。它支持查找、剪切、替换等基本编辑功能，也支持符号跳转和代码补全等高级编辑特性。vimtutor是最好的入门教程，初学者跟着这个教程学习一遍就可以用vim来编程了，在用得比较熟练之后，再去掌握那些高级功能。你对vim的功能掌握得越熟练，就越能高效地工作，投资点时间来学习vimtutor完全是值得的。
- 编译器使用gcc。编译器的功能是把源代码翻译成计算机可以“读懂”的机器语言。在Linux下可用的C编译器有好几个，gcc是其中最流行的，大多数发行版都默认安装了gcc。gcc的参数很多，看起来很复杂，但我们只需要掌握类似 `gcc -g test.c -o test` 这样的最简单的用法就好了。
- 调试器使用gdb。调试器的功能是帮助程序员定位错误，这是最后一招，也是程序员不太乐于采用的一招，需要频繁使用调试器通常说明你的编程水平不高。不过对初学者来说，掌握这个工具是必不可少的。gdb的功能强大，推荐读者使用更为灵活方便的命令行gdb。在这里读者只需要先掌握如何设置断点、显示变量和继续执行等基本操作就行了。
- 工程管理使用make。make是Linux下最流行的工程管理工具，Makefile是make的输入文件，它本身就相当于一种编程语言，执行make相当于调用其中的函数。编写Makefile是一件繁琐无趣的工作，幸好我们不用学习如何编写Makefile，后面我们会讲解make的改进版automake，现在你只要能写出下面这种简单的Makefile就行了。

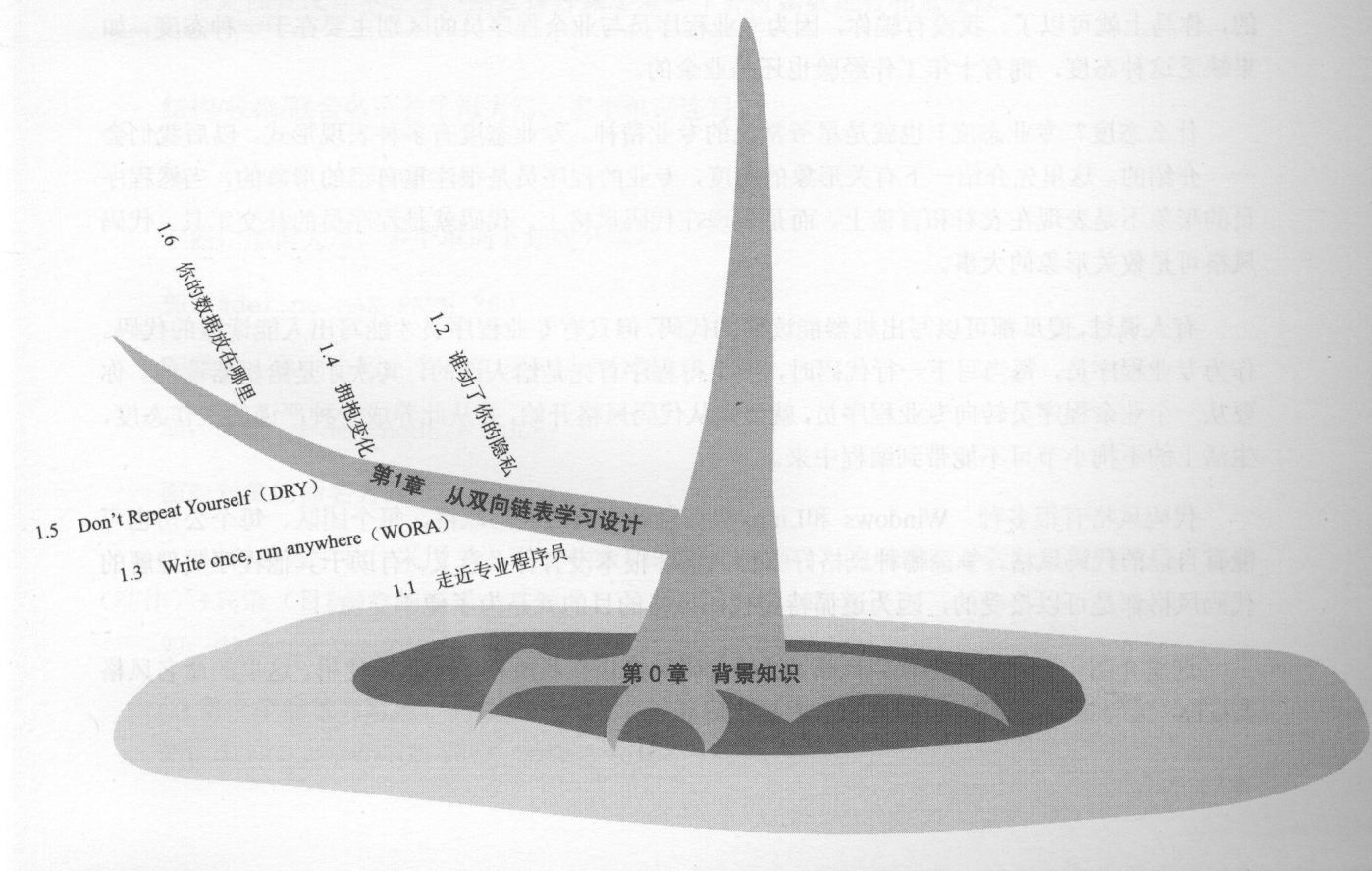
```
all:
    gcc -g test.c -o test
clean:
    rm -f test
```

在这里，你可以把all看作一个函数名，gcc -g test.c -o test是函数体（前面加tab），它的功能是将test.c编译成test，在命令行运行make all就相当于调用这个函数。clean是另外一个函数，它的功能是删除test。如果你有时间，多了解一下Makefile当然更好，如果没有时间，了解这么多也够了。

在培训初学者时，如果他从来没用过Linux，从来没有用C语言写过程序，我会给他2到4周的时间学习上述内容。如果读者也处于类似的水平，请不要急于了解后面的内容，最好先好好学习一下这里提到的知识。

第 1 章

从双向链表学习设计



1.1 走近专业程序员

需求简述

用C语言编写一个双向链表。如果你有一定的C语言编程经验，这自然是小菜一碟。有的读者可能连一个小程序都没有写过，那也不用担心，可以参考任何一本有关数据结构和C语言的书籍。先弄清楚基本概念，把书上的代码看明白，再把代码原封不动地输入到电脑里，保证编译通过，然后调试程序直到它能正常运行。重复这个过程，直到你能独立完成它为止。写第一行代码通常是很痛苦的，我培训过好几个同事，他们不是科班出身，刚开始时他们就算在电脑前坐一整天，也是连一行代码都敲不出来。其实我最早写程序时的情况也好不了多少，不过没有关系，迈出这第一步，以后的路也就好走很多了。

请先花1～3天时间完成这个任务，然后再继续往下阅读。记得多写多练，不要偷懒。

当你读到这里的时候，相信你已经独立写出了一个双向链表。恭喜你！迈出这一步可是值得庆祝的，现在你已经走在成为程序员的光明大道上了。不过你还是个业余程序员，那当然了，你才写出第一个程序呢！什么时候才能成为一个专业程序员呢？三年还是五年工作经验？其实不用的，你马上就可以了，我没有骗你，因为专业程序员与业余程序员的区别主要在于一种态度，如果缺乏这种态度，拥有十年工作经验也还是业余的。

什么态度？专业态度！也就是星爷常说的专业精神。专业态度有多种表现形式，以后我们会一一介绍的。这里先介绍一下有关形象的态度，专业的程序员是很注重自己的形象的，当然程序员的形象不是表现在衣着和言谈上，而是表现在代码风格上，代码就是程序员的社交工具，代码风格可是攸关形象的大事。

有人说过，傻瓜都可以写出机器能读懂的代码，但只有专业程序员才能写出人能读懂的代码。作为专业程序员，每当写下一行代码时，要记得程序首先是给人读的，其次才是给机器读的。你要从一个业余程序员转向专业程序员，就要先从代码风格开始，并从此养成一种严谨的工作态度，生活上的不拘小节可不能带到编程中来。

代码风格有很多种，Windows 和Linux都有自己主流的代码风格，每个团队、每个公司也可能有自己的代码风格，争论哪种风格好哪种风格坏根本没有什么意义。有助于其他程序员理解的代码风格都是可以接受的，因为遵循特定代码风格的目的就是为了便于交流。

这里介绍一下我喜欢的代码风格，这种代码风格也在我所在的团队中使用。这里的命名风格与GTK+代码相近，排版风格与Linux内核代码相近。

命名要展示对象的功能

文件名：单词小写，多个单词用下划线分隔。

如：dlist.c （这里d代表double，是通用的缩写方法。）

注意 文件名一定要能传达文件的内容信息，别人一看到文件名就能知道文件中放的是什么内容。把一个类的代码或者某一类代码放在一起是好的习惯，这样就很容易给文件取一个直观的名字。业余爱好者常常把很多没关系的代码糅到一个文件中，结果造成代码杂乱无章，也很难给它取一个恰当的名字。

函数名：单词小写，多个单词用下划线分隔。

如：find_node

注意 一个函数只完成单一功能。不要用代码的长度来衡量是否要把一段代码独立成一个函数。即使只有几行代码，只要这些代码完成的是一项独立的功能，都应该将其写为一个单独的函数，而函数名要能够直观地反应出它的功能。如果在给函数起名时遇到了困难，通常是函数设计不合理，则应该仔细思考一下并对函数进行相应修改。

结构/枚举/联合名：首字母大写，多个单词连写。

如：struct _DListNode;

宏名：单词大写，多个单词下划线分隔。

如：#define MAX_PATH 260

变量名：单词小写，多个单词下划线分隔。

如：DListNode* node = NULL;

面向对象的命名方式

(1) 以对象为中心，采用“主语（对象）+谓语（动作）”的形式来命名，取代传统的“谓语（动作）+宾语（目标）”的形式。

如：dlist_append

(2) 第一个参数为对象，并用thiz命名。

如：dlist_append(DList* thiz, void* value);