普通高等教育“十二五”重点规划教材 计算机系列

高级C语言编程

师敏华 沈玉龙 董学文 编著

科学出版社

普通高等教育“十二五”重点规划教材·计算机系列

高级C语言编程

师敏华　沈玉龙　董学文　编著

科学出版社

北京

内 容 简 介

作者总结多年的从业经验，讲解了C语言在实际工程中的高级应用方法及技巧，将实际应用中的难点通过清晰易懂的文字及简单明了的示例程序进行展现。

全书共10章：第1章介绍了编程风格及编码规范；第2章和第3章分别介绍了数据类型和字节序、字节对齐；第4章和第5章讲解了Socket编程和多进程、多线程编程；第6章到第9章介绍了一些代码优化、检查工具、调试及故障分析手段等方面的知识；第10章简单介绍了VMware虚拟机的应用。

在学习本书之前，读者应先学习C语言基础知识。本书可作为大专院校高年级学生或者研究生教材，尤其适合即将步入职场的毕业生，也可供计算机相关工程技术人员参考。

图书在版编目（CIP）数据

高级C语言编程/师敏华，沈玉龙，董学文编著. —北京：科学出版社，2016

（普通高等教育“十二五”重点规划教材•计算机系列）

ISBN 978-7-03-047235-9

I. ①高… II. ①师…②沈…③董… III. ①C语言-程序设计 IV. ①TP312

中国版本图书馆CIP数据核字（2016）第021774号

责任编辑：赵丽欣 / 责任校对：王万红

责任印制：吕春珉 / 封面设计：东方人华设计部

科学出版社 出版

北京东黄城根北街16号

邮政编码：100717

http://www.sciencep.com

百善印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2016年3月第 一 版 开本：787×1092 1/16

2016年3月第一次印刷 印张：16

字数：404 480

定价：38.00元

（如有印装质量问题，我社负责调换〈百善〉）

销售部电话 010-62136230 编辑部电话 010-62134021

前　言

C语言自诞生以来，就以其功能丰富、灵活高效的特点迅速风靡全球。使用C语言编写的程序更是不计其数，这其中就包括了大名鼎鼎的UNIX/Linux操作系统。另一方面，基于C语言还衍生出了C++语言，同样，使用C++语言也编写出了我们更为熟悉的Windows系列桌面操作系统。

然而，近年来随着移动互联技术的快速发展，各种移动互联的应用技术如雨后春笋般快速出现，以至于Java及其相关技术遍地开花，Java开发人员更是一时洛阳纸贵，而C语言却似乎大有没落之势，甚至有些计算机专业的学生都放弃对C语言的学习而直接学习Java语言。C语言真的会没落吗？答案当然是否定的。当我们在互联网中畅游时，有没有想过，所有这些信息可能需要经过无数的路由器、交换机、电信设备（包括基站、基站控制器、核心网等）等进行传输，而这些设备中运行的程序几乎90%以上是用C语言写成的；当我们使用手持终端在开心地交流或者娱乐时，有没有想过这些设备的底层操作系统以及设备硬件的驱动也基本是用C语言编写的。因此，到目前以及能预见的一段时间内，C语言不会没落，还将一直发展、应用下去。从另一个角度来看，学习C语言和学习Java语言并不矛盾，通过学习C语言应用可以对底层（操作系统层面、硬件层面以及协议栈层面等）有更为深刻的认识。正所谓根深才能叶茂，如果我们不去探究深层次的道理，而是直接使用现成的封装组件进行应用，那么我们可能永远只能停留在表面。因此，掌握C语言高级应用是一个优秀程序员必须所具备的能力。

本书没有过多探讨C语言本身的语法规则以及数据结构、算法等，而是着眼于实际的工程应用，将在工程开发中的常用技术以及难点和常见问题进行了讲解和总结。作者从事工业编程十多年，经常遇到这样的事情，就是同样的技术问题被不同的人一犯再犯。而通常的情况是编写代码的人清楚代码的业务逻辑，却使用了错误的技术，自己也找不出问题的原因。于是只好由另外一批并不太清楚业务逻辑的人花费巨大精力进行问题分析。这样不但增加了产品的成本，而且还使得代码编写人员可能会承受一定的压力。

本书用了大幅的篇章讲解了Socket编程、多进程、多线程、信号量等技术，因为这些技术是构建所有通用软件平台的基础技术。在讲解这些技术的同时，我们不得不提及伟大的IT巨匠威廉·理查德·史蒂文斯（William Richard Stevens）。理查德于1999年去世，终年48岁。他一生致力于UNIX系统及网络通信方面的理论和编程研究，为我们留下了《TCP/IP详解》三卷、《UNIX网络编程》两卷以及《UNIX环境高级编程》经典巨著，这些著作至今仍被视为UNIX及网络通信编程的“圣经”。但这些技术涉及的内容很多也很复杂，比如说TCP/IP通信技术，要想精通TCP/IP通信技术并不是一件容易的事。从原理到具体协议栈的实现以及应用就有厚厚的几本书需要研读。但在我们实际工程应用中，大多数的知识我们并不会用到。因此基于这个层面，本书重点介绍了这些技术在工程中的基本应用，并没有针对某个知识点做特别深入的探讨，这样能使读者

快速熟悉这些技术，尽快在工程中上手应用，在应用过程中根据场景继续进行深入学习。因此，本书特别适合大专院校的高年级学生或者研究生，尤其是即将步入职场的毕业生。通过本书的学习，可以掌握C程序大多高级应用，真正做到“知行合一”。

本书所举的程序都非常容易理解，部分程序可能并没有实际的应用意义，但这些程序一方面可以清楚说明和演示所讲述的知识点，另一方面这些程序就是实际工程应用的框架模板。将实际中的业务逻辑加入到这些框架中就成为了有意义的实际应用程序。

在本书的撰写和定稿中，为了确保内容的正确性，我们经常反复讨论和修改，但是限于作者的水平，错误和疏漏在所难免，希望广大读者朋友不吝指正。

目　录

第1章

编程风格与编程规范

1.1 引　言

Programs must be written for people to read, and only incidentally for machines to execute（程序必须为阅读它的人而编写，只是顺便用于机器执行）. ——H. Abelson and G. Sussman（in *The Structure and Interpretation of Computer Programs*）

Write program for people first, computer second（编写程序首先为人，其次为计算机）. ——Steve McConnell（in *Code Complete*）

诸如“软件维护期成本占整个生命周期成本的50%以上”、“解决软件使用过程中的问题所付出的成本远远高于开发过程中的成本”此类的话语似乎早已是“陈词滥调”了，但不幸的是，我们还必须得继续强调这样的“滥调”，没有为什么，事实就这样。

一般没有参与过大型系统编程的初学者，代码往往会写得很随意甚至故意炫耀技巧。尽管这些代码可以正常工作，但其可读性、可维护性往往不尽如人意。研究表明，优秀的程序员善于弥补其不足之处，使用有效的、最简单的技术，所编写的代码让自己和他人都易看懂，其中的错误也较少。

C语言是非常灵活的一种语言，灵活的同时就会出现一些让人理解起来比较困难的语句，本书不会去研究类似i++ + ++i或者 *p++这类语句或表达式，因为一个有着良好编程风格的程序员也不会使用这样的写法，他们会使用清晰的表达式或语句来表达自己的想法。

如果要问什么是优秀的代码？简单地说，清晰、简洁的代码就是优秀的代码。清晰指的是代码易于维护、易于重构；简洁指的是代码易于理解、易于实现。

1.2 编 程 风 格

什么是编程风格？可以理解为一种编程中的编排格式。编程规范更倾向于C语法的规范使用形式，编程风格更倾向于是一种框架、一种书写形式。我们都知道C语言的书写形式很随意，一行可以写很多条语句，有些循环或者判断并不需要使用花括号，代码行之间并不一定需要增加空行等。尽管这些程序能够被机器正确执行，但这样的程序看起来很不清晰。同样功能的程序，如果使用一定编程风格进行排列安排，那么程序看起

来将会有很强的层次错落感，也显得非常优雅，从而也易于理解和维护。

1.2.1 程序版式

1. 代码行独占原则

代码行独占原则包含两重含义：第一，一行最多包含一个声明或定义；第二，一行只能包含一条语句（for 条件判断语句不包含在内）。

一行多个变量，不便于就某个变量加注释。另外，一行多个变量可能会产生一些歧义。比如有如下定义：

```
char* ps8SrcStr,ps8DstStr;
```

编程者的本意是想定义两个指针，实际上却是定义了一个指针和一个字符型变量，而通过分行定义可以避免这样的非预期结果。

一行只能包含一条语句主要是为了提高程序可读性，使得程序看起来更有层次感，尤其是当控制语句嵌套较多时，如果代码行不独占，程序阅读起来就比较困难，而且这种冗长的语句容易隐含错误。

C 语言允许判断条件或者循环条件时，如果执行语句只有一句时可以不使用 { }。但这样的写法影响程序的可读性，也影响程序的优雅性。因此，一般都规定：if、for、while、case、switch、default 等语句也独占一行，且 if、for、do、while 等语句的执行语句部分无论多少都要加括号{}。但这里有个例外，就是 do...while 结构，在这个结构中，建议 do 独占一行，while 和最后的括号紧贴一起。值得注意的是，有些企业明文限定不建议使用 do...while 结构，因为任何 do...while 结构都可以用 for 结构或 while 结构来代替。

```
do
{
    ...
}while (u32MsgLen < MAX_MSG_LEN)
```

2. 空行原则

在程序中适当增加空行，不但能明显提升人们的视觉审美，而且可以提升代码的可读性。增加空行的原则是：在变量声明之后、相对独立的程序块之间增加空行；在 return 之前根据代码的拥挤程度决定是否增加空行。例如：

```
void  var_declare_and_prog_block_blank_line()
{
    U32  u32MsgLen = 0;
    U32  u32Temp = 0;

    u32MsgLen = get_msg_len();
    if (u32MsgLen > MAX_MSG_LEN)
    {
    /* Some Process  Code */
    ...
```

```
    }

    u32Temp = u32MsgLen;
    ...

    return;
}
```

3. 空格与缩进原则

在程序中使用适当的空格和缩进，除了可以使得程序错落有致外，更重要的是可以更清晰地显示程序的结构。空格与缩进主要有以下几个原则：

1）最顶级的代码紧贴编辑器的最左边，每一级代码（使用{}包括的代码段）应比上一级代码缩进4个空格，比如：

```
void code_indent_example()
{
    /* some variable declare */

    if (Expression1)
    {
        /* Some code */

        if (Expression2)
        {
            /* Some code */
        }
    }
    else
    {
        /* Some code */
    }

    return;
}
```

2）二元操作符前后应该增加一个空格。比如：

```
u32TotalLen = u32MsgHeadLen + u32MsgBodyLen;
```

3）一元操作符和数组及结构体运算符不能有空格。比如：

```
u32Array[1] = 0;
u32Loop++;
pstStudent->u32Age = 10;
stTeacher.u32Age = 35;
```

4）if、for、while等关键字后应该留一个空格再跟括号。

5）文件中不应该出现Tab符号，应全部使用空格（Tab符号在编辑器中如果没有进行特殊设置的话是看不出来的，大多数C编辑器都提供了将Tab使用空格代替的选项，

比如 UltraEdit、SoureSight 及 VC 等，在编辑器的设置中将 Tab 设置为使用 4 个空格替代，这样，在程序缩进的时候，就可以直接使用 Tab 键缩进，编辑器将自动替换为 4 个空格）。

4. 长行拆分

一行程序不宜过长，太长的话阅读起来非常不方便，那么一行程序应该多长合适呢？这没有确定的数字要求，有些建议是 80 个字符，有些建议是 120 个字符。本书更倾向于使用 80 个字符，如果超过这个长度则建议折行。其实这个原则需要灵活掌握，比如，代码长度就是 81 个字符，那就建议不折行。再比如某行有 110 个字符，必须在第 80 个字符处折行吗？这个不是必需的，应该观察在什么地方折行后更具有对称美观性、更易于人们阅读。因此，单纯长行拆分的原则在实际使用中应灵活一些。

在长行拆分中还有一种情况，就是 if 判断中如果判断语句过多、过长，这个时候建议拆分时将每一个判断条件独立一行，将条件判断之间的运算符放在行首，如下所示：

```
if ((u32MsgLen > g_stCommRecvInfo[1].stMsgInfo.u32MsgMaxLen)
|| (NULL == stMsg.stBody.pu8MsgContentBuf)
|| (u32MsgType == g_stCommRecvInfo[1].stMsgInfo.u32MsgExcepion))
{
    /* do something */
}
```

1.2.2 注释

程序的注释对提高程序的可读性和可维护性有很大的帮助，尤其是一些较为复杂的算法，如果注释清楚的话，则容易理解。但在一些应用程序中，由于程序员只更新代码并不更新注释，代码和注释已经不能匹配，甚至代码和注释含义相反，结果给后续维护人员带来巨大的烦恼。因此，编写良好风格程序的同时必须也配套有良好的注释，注释是程序的一部分。注释的原则是有助于对程序的阅读理解，在该加的地方都加了，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁。注释的量应该为多少比较合适呢？一般情况下给出的量化原则是程序总行（包括空行、注释行和代码行）的 20%～40%。

1. 注释符号

C99 中已经支持“//”注释符，但是有些编译器并不支持这个注释符，比如风河的 VxWorks 操作系统的编译器。因此，为了提高程序的可移植性，建议使用“/* */”注释符。

2. 注释原则

程序中增加注释主要有以下几个原则：

1）注释中不能嵌套注释符号。

2）注释的位置应与被描述的代码相邻，可以放在代码的上方或右方。

3）注释的内容要清楚、明了，含义准确，防止注释二义性。

4）边写代码边注释，修改代码的同时修改相应的注释，以保证注释与代码的一致性，不再有用的注释要删除。

5）注释与所描述内容进行同样的缩排。

6）对于已经能非常清楚表达意思的语句则不需要增加注释。

7）不增加没有意义的注释。

上边的几点原则比较容易理解，对于6、7两条示例如下：

```
if(ulMsgLen > MSG_MAX_LEN)  /* 消息长度太长 */
{
    ...
}
```

这个注释意义不大，因为语句中无论变量的名称还是宏定义的名称都是自我解释的，已经很清晰了，所以增加这个注释只能增加注释的统计量，并不能带来任何好处。

```
if(NULL == pstMsg)  /* 指针判空处理 */
{
    ...
}
```

这个注释毫无意义，犹如画蛇添足，类似这样的注释应该摒弃。

3. 函数头部注释

每个函数之外需要增加函数的注释说明，格式可能每个团队都不太一样，但应该达成统一风格。一般来讲，函数的注释中应该给出函数的功能、函数的参数说明、函数的返回值及备注说明等，下边给出一个可能的示例。

```
/*****************************************************************
* 函数名称:  func name
* 函数功能:  <描述函数实现的功能>
* 函数参数:
* 参数名称:    类型          输入/输出      描述
*  XXXXXX     XXXX            XXXX        XXXXXX
* 返回值:
* 函数说明:
* 修订记录:
*****************************************************************/
```

4. 文件头部注释

对于.h头文件和.c源文件都需要在文件的开始增加注释，描述该文件的版权、作者、功能、修订记录等。由于.h和.c文件还涉及文件包含等方面的内容，附录中将给出一个.h和.c文件示例。

1.2.3 工程文件组织形式

往往一个大型的应用程序在逻辑上是由很多子系统、功能模块组成的，同时也包含了很多文件。这些文件应该如何组织呢？最自然也最清晰的办法就是使用目录分级管

理，并且头文件和源文件放在不同的目录，下边给出一个比较典型的工程文件组织结构。请注意，图 1-1 所示的组织结构中，第一级结构中有一个 public 目录，下边每一级也都有自己的 public 目录，而且 public 目录里边只有 include 目录，public 里边只是用来放各自系统、模块的公共头文件。这样做可以让整个工程的头文件包含比较清晰，也可以避免交叉包含而导致的编译错误。

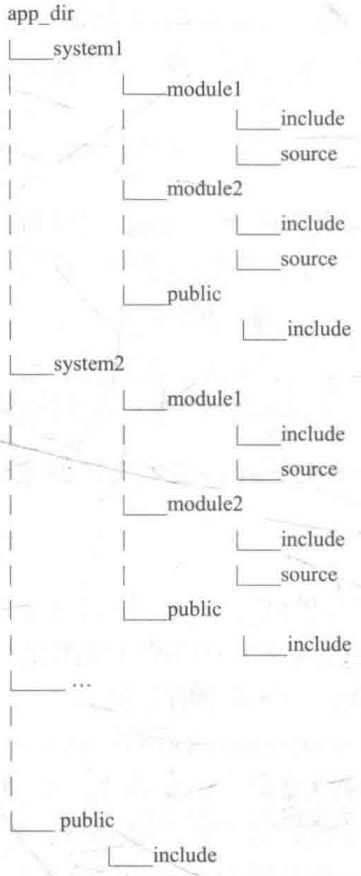

图 1-1　工程文件组织结构

1.2.4　文件命名方式

每一个.c 文件名称的格式应该是“子系统名称_模块名称_子模块名称_功能描述 1_功能描述 2.c”，文件名的长度不要超过 48 个字符。

1.2.5　工程编译

对于以这样的体系组织起来的工程来讲，编译并不是一件很容易的事情，需要手动做一些事情。我们在.c 源文件中不需要并且禁止使用以带路径的方式包含头文件，那么在编译时，就需要告诉编译器在什么地方找这些头文件，这就是所谓 makefile.mak 文件的作用。当然，这只是 makefile 的一个作用（一般称其为 makefile，实际编译时，命令行中执行 make 命令，make 命令就会找到 makefile.mak 文件，根据该文件指定的编译规则进行编译）。换句话说，其实 makefile 就是告诉编译器的编译方法和规则，因

此 makefile 文件中需要指定编译命令（比如 gcc）以及许许多多的编译选项、编译宏等。这么做的好处就是当头文件的路径或者范围发生变化时，不需要逐个修改代码文件，只需要修改 makfile 即可。makefile 文件的编写比较麻烦，在 VC 环境下，可以通过设置相关编译选项的图形界面从而免去编写 makefile 的麻烦。但如果需要在 Linux 或者 Solaris 等操作系统下编译的话，就不得不去亲自编写 makefile 了。关于如何编写 makefile，内容很多，有兴趣的读者可以参阅相关资料。下面简单介绍在 VC6 下如何编译一个工程。

在 VC 下首先建立一个类型为“Win32 Console application”的空的工程，然后在左栏中的文件视图里，在工程名称上单击右键，可以看到有“New Folder”选项，这个就可以建立新的目录。需要注意的是 VC 上的编译工程建立的目录并不需要和实际代码存放的目录对应。比如说，1.2.3 小节中示例的工程目录有 2 级目录，即系统+模块，在建立 VC 编译工程时是否也需要建立两级目录，这个需要根据代码文件的多少或者建立工程的人的习惯和喜好来决定，并没有规定标准。但是，VC6 建立多级目录的工程时，所有目录，包括其子目录的名称是不允许重复的，这个和实际文件存储目录的规则是不同的。实际文件存储时，不同的目录下可以有相同的名称的子目录，但 VC6 的工程则不允许。因此，一般情况下，在 VC6 中建立编译工程时，只需要建立一级目录，为每一个系统建立一个目录，然后将该系统下的所有.c 文件添加到这个目录即可。通过在建立的目录上单击右键，选择弹出菜单的“Add Files to Folder”选项就可以将文件加入到工程，如果多个.c 文件都实际存储在一个目录下的话，可以一次全选将所有文件加入到工程，但是如果文件存储在多个目录中的话，则需要逐个目录依次加入到工程。图 1-2 是建立好的一个示例工程，在 system1 的目录中，将 system1 中的 module1 和 module2 的源文件全部加入进来。

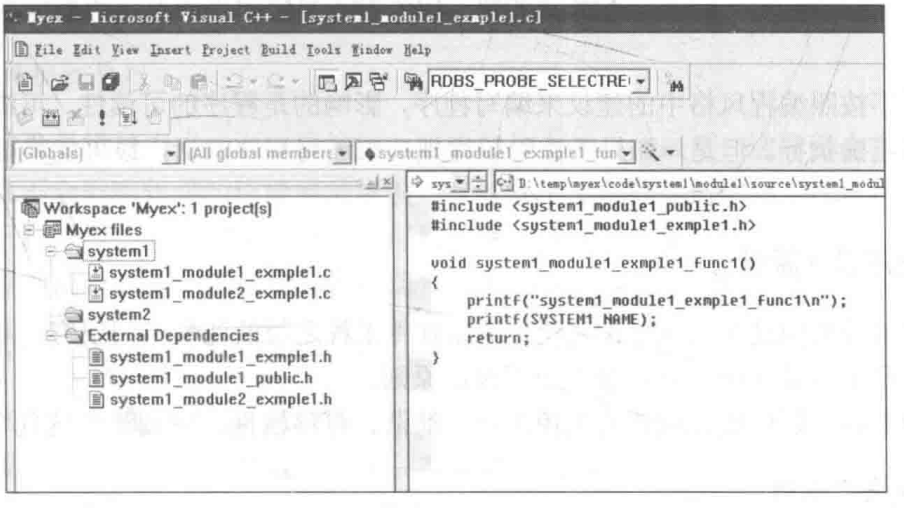

图 1-2　VC6 中一个工程示例

目录建立完成以及所有的文件都加入完成后，就需要指定头文件的路径，头文件的路径指定可以针对整个工程，也可以针对某一级目录，还可以针对某一个具体的文件。一般来讲，都是针对某一级目录来指定（这里的目录其实就是每一个子系统），当为这

一级目录指定了头文件路径后，其下所有的源文件都默认使用这个路径。

在目录上单击右键，然后选择弹出菜单的“Settings”选项来设置编译选项。在弹出对话框中，选择 C/C++页面中的 Preprocessor，如图 1-3 所示，图中方框中所示就是填写头文件的路径。

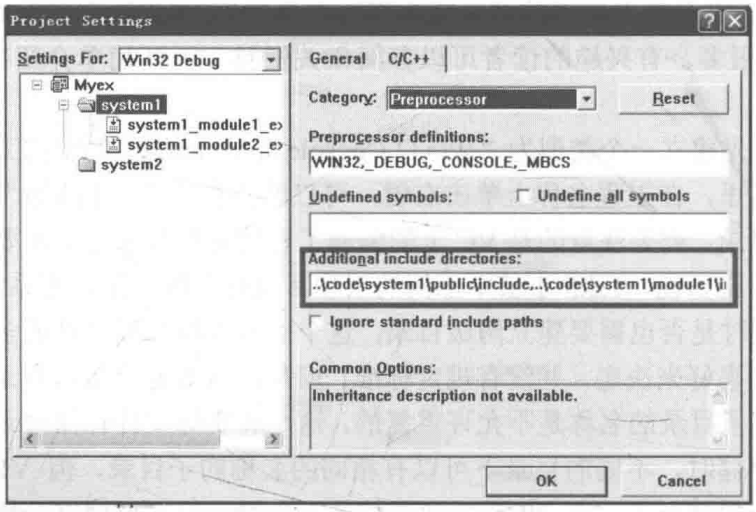

图 1-3 VC6 中设置工程的头文件路径

头文件路径可以是多个，中间使用逗号“,”隔开即可。在设置路径时一般情况下建议使用相对路径。相对路径的起点是 VC 工程文件（.dsp/.dsw）存放目录。“../”表示上一级目录，以此类推，“../..”表示上上级目录，“./”表示当前目录。

1.3 编 程 规 范

如果不按照编程风格中的建议来编写程序，影响的是程序的可读性、可维护性，但程序可以正确执行。但是，如果不按照编程规范来编写程序的话，很可能将影响程序的正确执行。

1.3.1 程序设计原则

程序设计原则这个问题非常宽泛，诸如软件工程之类的课程专门研究。本书对于这个问题立足于作者实际经验，给出一些设计原则。

一般来讲，程序设计应该遵循模块化、简洁、可移植性、可调测性这几个原则。

1. 模块化原则

模块化的原则是编程的首要原则，模块化原则有两个层面的含义：第一个层面是对软件功能进行拆分，拆分出一个个的功能块，每个功能块再拆分成更小的功能块，自顶向下逐步求精完成程序设计，目的是降低程序复杂度，使得程序设计、调试和维护简单化，是分治思想在程序设计中的应用。第二个层面是综合分析已有的各种产品和应用中

相似的功能模块，充分利用已有的模块，或加工出可复用的模块，目的是提高开发效率，降低开发成本，提升开发质量，是相似性原理在程序开发中的应用。

模块化拆分要按照正交性的原则进行拆分。正交性要求拆分出的功能单元或组件具有具体的良好定义的接口和职责，独立、相互隔离，改变其中之一，只要不改变接口，就不用担心对其他功能单元或组件造成影响。建议借鉴面向对象的思想进行拆分，拆分出的模块是一个对象，多个对象相互协作完成整个功能。

比较大的功能模块在实现模块时尽量分成两部分，把“接口”部分及“内核”部分分开开发，以提高“内核”部分的可移植性和可复用性。对不同产品中的某个功能相同的模块，其接口可能不同，若能做到内核部分完全一致，那么无论对产品的测试、维护，还是后续产品的开发都会有很大帮助。实现模块内核的功能时，要按照最大限度提高复用、降低重复的原则进行开发。需要开发出可复用的组件、功能单元、函数；充分利用已有的模块化成果，利用通用的算法和数据结构提高效率。

模块化最后一个问题是，建议模块化的时候也要考虑接口的标准化，各产品各项目使用模块化的组件、标准化的接口，开发出系列化的产品。

2. 简洁原则

简洁是程序是否良好的最重要的特征，简洁的程序稳定、可靠、易于理解，不会有意想不到的问题。简洁原则要求程序简单、整洁、不罗嗦、无冗余，但使程序简洁是一项复杂的工作。简洁的实现背后往往有大量不简单的思考，这是设计人员、编程人员重点要做的工作。要做到简洁需要综合考虑大量信息，要理清紧密交织在一起的关系，对软件进行正确的划分，选择恰当的算法，划清函数的职责和范围，对函数接口和实现进行仔细的设计。通常情况下，对于实现相同功能的软件，质量与软件的大小成反比；简洁的原则要求软件整体的思路上结构简单到不能再简单。

3. 可移植性原则

可移植性是设计出来的，需要按照一定的架构支持可移植性。这样做会在将来得到回报，而且不会影响代码的性能或清晰度，有时甚至还能提高清晰度。在设计阶段要尽早地考虑可移植性问题，如果没有考虑可移植性，随着工作平台环境的变化，一次次地重新调整以前的设计是一项代价高昂的工作，最后使得代码变得一团糟，不可维护。

4. 可调测性原则

程序必须具备可调测性。可调测性是指可以对程序是否按照设计工作进行检测，程序必须在设计阶段把可调测性设计进去。在设计的时候就要考虑，站在系统、子系统、模块的角度给出程序是否按照设计工作的检测方法，给代码编写以指导。程序在开发阶段我们可以依赖开发环境的 debug 手段来检测程序 Bug，但程序一旦商用，就需要依靠程序输出的日志、告警及一些后门手段来分析、定位程序 Bug。因此，在程序设计时候就要想到在商用环境中如何去分析解决程序的问题。