《程序设计实践》

图书基本信息

书名：《程序设计实践》

13位ISBN编号：978711540786X

出版时间：2016-1-1

作者：Brian W. Kernighan,Rob Pike

页数：251

《程序设计实践》

内容概要

本书是计算机科学方面的经典名著，由计算机界极具影响力的两位专家Brian W. Kernighan和Rob Pike合著。书的内容围绕程序设计实践中的一系列问题展开，讲述对于程序员有共性的知识，以帮助各程序员写出更高效的程序。本书从排错、测试、性能、可移植性、设计、界面、风格和记法等方面，讨论了程序设计中既具有实际意义又具有广泛意义的思想、技术和方法。
本书值得每位梦想并努力成为程序员的人参考，值得每位计算机专业的学生和计算机工作者阅读，也适合作为程序设计高级课程的教材或参考书。

《程序设计实践》

作者简介

作者:[美] 布莱恩 W. 克尼汉（Brian W. Kernighan）罗勃·派克（Rob Pike） 译者:无
Brain Kernighan计算机科学家，曾与UNIX的缔造者Ken Thompson和Dennis Ritchie一起在贝尔实验室工作。他也是AWK和AMPL程序设计语言的共同作者。“ K&R C ”和“ AWK ”中的“ K ”都是指“ Kernighan ”。2000年起，他在普林斯顿大学计算机科学系任教授，并任本科部代表。
Rob Pike 软件工程师。他在贝尔实验室任职期间，作为UNIX小组成员参与开发了Plan 9和Inferno操作系统以及Limbo程序设计语言。目前他在Google公司工作，参与了Go和Sawzall程序设计语言的开发。

《程序设计实践》

书籍目录

目录

《程序设计实践》

精彩短评

1、Simplicity & Clarity; Generality; Evolution; Interface; Automation; Notation;
2、之前发言过于不理性，现在重新来过，对之前的发言深表遗憾。与出版社工作人员沟通得知，这次封面设计不用原书封面，删减index是版权方要求所致。对index要求高的同学有准备就好了。仅此……3.17
3、养成良好的编程习惯, 学习优秀的编程手法. 本书可以说是涉及编程各阶段的技术与思路的引子, 可以按图索骥找具体的技术进行研究.
4、一般

《程序设计实践》

章节试读

1、《程序设计实践》的笔记-第3页

程序命名:

1. Use descriptive names for globals, short names for locals
* 针对全局作用域的数据/方法, 需要使用具有具体描述性的名称描述性是指: 通过名字可以知道如何进行编程使用.* 针对局部作用域的数据/方法, 建议使用短小的名称:i,j 用于循环遍历的下标; p,q 用于指针; s,t用于字符串
2. Be consistent
* 在名字(其实就是特定上下文的领域名称)的选取时, 需要保持前后的一致性.
* 在名字的使用时, 防止冗余: queue.length 优于 queue.queueLength. 这块在Golang上表现明显, 其名字的选择特别一致并且简洁.

3. Use active names for functions
* 使用动宾短语表示方法.
  * getTime
  * putChar
* 针对boolean型返回数值的方法, 建议使用is开头.
  * isFinish

4. Be accurate
* 名称表述准确: 一个名字就是一个让其他程序员看的接口, 只有准确的名字才不会引起人的误解

2、《程序设计实践》的笔记-第103页

接口的设计原则:

1. Hide implementation details. 隐藏实现细节, 主要方法是OO思想(信息隐藏, 封装, 抽象, 模块化)
2. Choose a small orthogonal set of primitives. 接口设计上选用一个最小的正交功能集合. 一个接口尽量完成少的功能, 功能之间是正交(不同的功能维度)的.
3. Don't reach behind the user's back. 不要触碰用户完全无法理解的地方, 例如通过一些硬编码的隐藏文件进行信息交换/直接修改一些全局变量, 这种远离了接口上下文的操作与数据读写需要减少并告知使用者.
接口在实现过程中, 如果涉及数据操作, 则尽量自包含与管理.
4. Do the same thing the same way everywhere. 保持接口实现的一致性, 不论是功能实现模式/命名等各个地方. 常见的地方: 命名, 注释, 代码编写风格.

3、《程序设计实践》的笔记-215 Notation

超越单一语言层面的一些通用编程技法: 数据格式化, 正则表达式, 脚本, 各种复杂指令集合的解释器/编译器/虚拟机等.

1. Formatting Data / 数据格式化216
  * Golang 的fmt的定义模式

4、《程序设计实践》的笔记-第17页

宏这种语法最早出现在C系语言中, 在初期主要是进行高调用的短小函数的编写. 但随着现代编译器的出现 , 这类问题已经被下层解决了.

1. Avoid function macros
不要使用宏. 由于宏只是简单的文本替换, 在不正确的使用下, 会导致很多问题. 例如:
#define isupper(c) ((c)>='A' && (c)<='Z')
//由于是直接替换, 导致直接的函数错误
while(isupper(c=getchar()))
.....

#define ROUND_TO_INT(x) ((int) ((x)+(((x)>0)?0.5:-0.5)))
//导致调用资源的浪费.
size = ROUND_TO_INT(sqrt(dx*dx+dy*dy))

<原文开始></原文结束>

2. Parenthesize the macro body and arguments
如果使用宏, 则利用()等将宏中的主体与参数进行包装, 以防止意外.
#define square(x) (x) * (x)
1 / square(x)
//导致如下错误
? 1 / (x)*(x)

5、《程序设计实践》的笔记-第109页

错误处理涉及到: 中止, 重试 与 失败. 目前针对这些错误, 程序的基础框架中提供了如下API: 中止(exit, assert), 重试与失败(try-catch).
针对接口与模块的提供方, 无法预知使用者的调用方法, 如果武断地直接中止程序将导致一些不好的后果(文字处理程序在路上前最好做一个数据的备份, 以防止退出后的数据重录). 因此错误处理最好交给使用者最后裁决.

1. Detect errors at a low level, handle them at a high level. 接口内部可以很早发现错误, 但最终是交给调用方进行错误处理.
2. Use exceptions only for exceptional situations.
异常机制的添加是为了让正常逻辑更有效的组织, 一般实现中异常处理部分只放置异常处理类事务.

《程序设计实践》

6、《程序设计实践》的笔记-第29页

我想这大概是一个基本的面试指南:搜索和排序，数组、列表、树和哈希表，这些都不会的话,out

7、《程序设计实践》的笔记-第113页

用户UI设计涉及交互设计等多个方面, 合理的提示会便于程序的使用.
不要指望用户会认真学习使用手册与用户说明.

8、《程序设计实践》的笔记-第19页

魔数是直接使用硬数字类型编码的如下使用: 常量, 数组长度(length), 字符串索引下标(index), 计算标量(km,mi) conversion factors, 以及其他出现在代码中的数字字面量.

1. Give names to magic numbers
给所有的魔数起个使用时的名字.

2. Define numbers as constatns, not macros.
将数字定义成常量, 而不是宏.

3. Use character constants, not integers.
? if (c>= 65 && c <= 90)
? .....
if (c>= 'A' && c <= 'Z')
.....

9、《程序设计实践》的笔记-第162页

1. Test as You Write the Code 像对待编码一样重视测试
* Test code at its boundaries. 边界测试
* Max-Int/Min-Int
* Test pre- and post-conditions. 测试前置/后置条件
* 期望的调用前环境
* Use assertions. 使用断言保证接口参数的正确性.
* Program defensively. 防御性编程. 将不可能发生的情况也涉及进来进行判断.
* 添加各种实际中不会出现的情况.
* Check error returns. 检查错误信息.
* errno
* Exception
* Compile warning
2. Systematic Testing. 系统性测试.
* Test incrementally.
* 循序渐进的进行测试
* Test simple parts first.
* 从简单的模块开始测
* Know what output to expect.

    * 知道程序的期望输出
  * Verify conservation properties.
    * 验证上下文属性
  * Compare independent implementations.
    * 对比不同的实现方式
  * Measure test coverages.
    * 进行测试覆盖率统计
3. Test Automation
  * Automate regression testing.
    * 优先进行回归测试的自动化.
  * Create self-contained tests.
    * 制作可自己产生上下文的测试用例, 各种mock.
4. Test Scaffolds.
5. Stress Tests
  * 高并发
  * 大数据
  * 边界变更叠加
6. Tips for Testing 一些有经验的测试人员设计的测试(下文中可以看到, 很多经验是对C/C++这类语言展开的. 现代语言, 特别是Golang, 结合本书的经验进行了大量的修补)
  * 检查数据边界
  * 保证测试过程的可复现性:
    * Hash输出一致
    * 让内存分配可以产生OOM, self-define-malloc
    * 保证各种变量有明确的初始化数值, 最好是使用更有价值的初始化数据(例如:Null-Object), 而不是简单的置0.
    * API调用时, 明确各参数, 以方便后续复现用例
    * 随机测试, 请记录开始时的seed
  * 发布版本前, 关闭一些测试时添加的代码, 以提高性能.
  * 及早构建可运行的测试用例, 提前验证用例本身的有效性.
  * 在发现问题后, 优先处理bug, 而不是赶工与做用例.
7. Who Does the Testing?
  * 白盒
  * 黑盒
  * alpha/beta 版本用户: 灰度用户

10、《程序设计实践》的笔记-第85页

设计的本质就是去平衡要完成目标与现实的约束.
作者给出了在进行设计时可以考虑的点:
* Interfaces: what services and access are provided? The interface is in effect a contract between supplier and customner. Ther desire is to provide services that are uniform and convenient, with enough functionality to be easy to use but not so much as to become unwieldy.
* Information hiding: what information is visible and what is private? An interface must provide straightforward access to the components while hiding details of the implementation so they can be changed without affecting users.
* Resource management: who is responsible for managing memory and other limited resources? Here, the main problems are allocating and freeing storage, and managing shared copies of information.
* Error handling: who detects errors, who reports them, and how? When an error is detected, what recovry is

attempted?
* 接口: 关注提供了哪些服务与访问请求. 这些服务与访问请求, 如同合同一样(统一/方便/简单易用但又不至于太过庞大(API/参数众多))是开发者与使用者之间的桥梁.
* 隐藏细节: 接口的使用要直截了当, 但内部实现可以自由变化且不影响现有用户的使用. 你变, 他不变.
* 资源管理: 内存等其他限制性资源(CPU/硬盘存储/共享变量等公共资源)由誰来管理需要明确.
* 错误处理: 针对程序错误中的检查/报告/处理/恢复由谁来负责需要明确.

11、《程序设计实践》的笔记-第212页

12、《程序设计实践》的笔记-第187页

13、《程序设计实践》的笔记-第61页

先确定需要什么数据，数据怎样组织(数据结构)，再考虑合适的算法，整个程序的设计呼之欲出。

14、《程序设计实践》的笔记-第6页

表达式与具体语句

1. Indent to show structure.
使用缩进来突显程序结构.
同样在这里Golang是典型的工程语言, 其内含了自动缩进与统一的format过程.

2. Use the natural form for expressions.
使用正常的格式(如果你平时读书一样丝滑)书写表达式.
Eg: 将含有not的表达转换成非not的

3. Parenthesize to resolve ambiguity.
使用括号来消除可能产生的歧义.
由于在实际开发实践中, 大量的程序人员对语法基础掌握不足, 导致像操作符(<,<=,&,||)优先级等问题上容易出现错误.

4. Break up complex expressions.
将复杂的语句进行多频简化.
例如: # ?? *x += (*xp = (2*k < (n-m)) ? c[k+1] : d[k--])

```
# change to
  if(2*k &lt; n-m)
     *xp = c[k+1];
  else
     *xp = d[k--];
  *x += *xp;
```

5. Be clear.
保持清晰. To write clear code, not clever code.
Clarity(清晰) is not the same as brevity(简洁).
清晰的代码应该是: 独立句子很短, 复杂含义通过多条语句表达. 其核心目标是为了便于其他人理解.

6. Be carefull wtih side effects.
与第3条类似, 防止一些操作符(i++,++i)产生的歧义.
甚至于针对下述不好的代码, 不同的编译器(下层程序员针对一些标准没有规范清楚的问题时, 自己定义了一些行为)可能会表现出不一样的输出结果.
不好的代码
str[i++]=str[i++]=' ';
array[i++] = i;
scanf("%d %d", &yr, &profit[yr]);

15、《程序设计实践》的笔记-第106页

数据资源管理

Free a resource in the same layer that allocated it. 一同一个层中申请与释放资源.

这里的资源有很多种类: 各级(内存,cache)存储器(calloc/free, new/delete, GC), IO(file,socket)句柄(open/close)

涉及到资源的操作, 则需要考虑: 资源容量能力(线程池, 数据 库连接池, 文件句柄池等), 操作的管理(同步与互斥), 数据的管理(申请与释放).
在涉及资源的接口设计时, 考虑到现代编程模型(多线程), 需要考虑接口的可重入性.

16、《程序设计实践》的笔记-第58页

算法与数据结构.
文中主要提及了以下数据结构与算法:

1. 检索(搜索): 线性搜索, 二分搜索
2. 排序: 快递排序
3. 自增长数组(通过平摊分析后, 在空间换时间的情况下, 自增长可以达到O(1))
4. 表
5. 树
6. 散列表

针对上述数据结构, 主要涉及如下操作: 增加元素, 查找, 删除等

17、《程序设计实践》的笔记-第10页

1. Use a consistent indentation and brace style.
请在使用缩进与{}时, 保持代码各处风格一致.

坚持使用{}来消除一些不必要的代码错误.

2. Use idioms for consistency.
使用一些行业标准的写法来保证代码的一致性.
例如如下代码样例:

```
for (int i = 0; i &lt; n; i++) {
   array[i]=1.0
}
for (p = list; p != NULL ; p = p-&gt;next) {
   ......
}

//用于无限循环
for(;;){
}

while ((c = getchar()) != EOF) {
   putchar(c);
}

//小心strlen用于字符串时不包含结束的'\0'
p = malloc(strlen(buf)+1);
strcpy(p, buf);
```

3. Use else-ifs for multi-way decisions
使用多段的if-else语句来描述多路选择.

```
if (argc != 3) {
   printf("Usage: cp inputfile outputfile\n");
} else if (NULL==(fin=fopen(argv[1],"r"))) {
   printf(" Can't open input file %s\n", argv[1]);
} else if (NULL==(fout=fopen(argv[2],"r"))) {
   printf(" Can't open output file %s\n", argv[2]);
} else {
   while ((c = getc(fin)) != EOF){
      putc(c, fout);
   }
   fclose(fin);
   fclose(fout);
}
```

如果使用switch-case语句, 请使用如下的样式:

```
switch(c) {
case '-':
   sign = -1;
   /* fall through */
case '+':
   c = getchar();
   break;
case '.';
```

```
    break;
default:
    if(!isdigit(c)){
        return 0;
    }
    break;
}
```

在其他语言中, 针对这块的结构设计有所不同:

* Ruby与golang: 允许各种类型与判断, 类似于if-else的变种. 并且在运行结束后, 不会向下一个指定项继续. https://golang.org/doc/effective_go.html#switch

18、《程序设计实践》的笔记-第117页

参见书目: 软件调试修炼之道 https://book.douban.com/subject/6398127/ 相关读书手记: http://www.5wpc.info/it/technical/debug/2013/04/21/HowToDebug/

1. 调试是一门艺术!
2. Good Clues, Easy Bugs 好的线索让bug浮出水面.
    1. Look for familiar patterns. 寻找常见的错误模式.
    2. Examine the most recent change. 查验最近的修改.
    3. Don't make the same mistake twice. 不要让错误重复出现.
    4. Debug it now, not later. 发现问题立刻跟进.
    5. Get a stack trace.
    6. Read before typing. 先检查, 再修改.
    7. Explain your code to someone else.
3. No Clues, Hard Bugs 没有线索, 大海捞针.
    1. Make the bug reproducible. 让Bug可以复现.
    2. Study the numerology of failures. 学习与分析过去的错误现场.
    3. Display output to localize your search. 利用特征现象进行分析.
    4. Write self-checking code. 添加自检查代码, 例如断言机制的使用.
    5. Write a log file. 添加日志进行分析.
    6. Draw a picture. 如果不知道程序发生了错误, 考虑将具体的数据dump出来, 通过具体数据进行分析.
    7. Use tools.
    8. Keep records. 像侦探一样整理已知线索, 回顾已经试过的路径.
4. Last resorts. 最后的手段.
    单步调试跟进.
5. Non-reproducible Bugs
    如果Bug无法复现, 就涉及一些特定的上下文环境:
    * 多线程与运行时序
    * 未初始化的数据变量
    * 大流量
    * 边界数据

19、《程序设计实践》的笔记-第23页

1. Don't belabor the obvious.
不要对显而易见含义的代码进行无效注释.
/* return SUCCESS */
return SUCCESS;
2. Don't comment bad code, rewrite it.
请重写垃圾代码, 而不是简单的注释.

3. Don't contradict the code.
不要在注释时, 文不对题. 此类问题多出在代码进行了重构, 但注释没有更新.

4. Clarify, don't confuse.
注释表述清晰, 不要含糊.

? int strcmp(char *s1, char *s2)
? /* string comparion routine returns -1 if s1 is above s2 in an ascending order list, 0 if equal */

/*
* strcmp: return < 0 if s1<s2, >0 if s1>s2, 0 if equal
*/
int strcmp(const char *s1, const char *s2)

20、《程序设计实践》的笔记-第82页

设计与实现

本章通过举例实现一个文章自动生成器的例子, 展示 了设计的重要性; 通过使用多种语言进行实现, 体现了不同语言在相同设计情况下性能与实现复杂度的关系.

《程序设计实践》

版权说明