

# 编程技巧

陈元昊

2022 年 9 月 30 日

## 目录

<b>1</b>	<b>Linux</b>	<b>2</b>
1.1	指令技巧	2
1.1.1	grep	2
1.1.2	gcc/g++	2
1.1.3	git	2
1.2	基本概念	2
1.2.1	进程管理	2
1.2.2	中断、信号	2
1.3	WSL2	2
1.3.1	网络	2
<b>2</b>	<b>C/C++</b>	<b>3</b>
2.1	算法技巧	3
2.1.1	预处理	3
2.1.2	算法执行	3
2.1.3	算法评估	3
2.2	语言特性	3
2.2.1	基本语法	3
2.2.2	面向对象	4
2.3	Qt	4
2.3.1	绘图	4
2.3.2	原理	4
<b>3</b>	<b>Python</b>	<b>5</b>
3.1	语言规范	5
3.1.1	程序结构	5
3.2	语言特性	5
3.2.1	运行特性	5
3.3	具体应用	5
3.3.1	正则表达式	5
<b>4</b>	<b>Javascript</b>	<b>6</b>
4.1	语言特性	6
4.1.1	函数	6
4.1.2	对象与原型	6

# 1 Linux

## 1.1 指令技巧

### 1.1.1 grep

1. 加-E选项后，可以在使用正则匹配时不用给括号转义

### 1.1.2 gcc/g++

1. 加-E选项后仅执行到预处理，文件后缀.i
2. 加-S选项后仅执行到编译，文件后缀.s
3. 加-C选项后仅执行到汇编，文件后缀.o

### 1.1.3 git

1. 当因为token的原因（一般存在于报错）无法clone时，可尝试设置一个在网站上设置一个具有权限的token，复制之，然后在Windows凭据管理器上新建/修改一个普通凭据，注意密码应为token

## 1.2 基本概念

### 1.2.1 进程管理

1. 挂起，一般通过按ctrl+z实现，效果为暂停执行（前台或后台程序均可以），但可用fg或bg恢复执行
2. 后台运行，一般通过在命令行末尾加“&”符号实现，也可以通过挂起+后台恢复间接实现，效果为以不占用终端的方式运行
3. 由于后台运行不能让shell以阻塞方式等待，所以不能直接用waitpid的方式等待，而是要先使用信号通知shell某个子进程的结束，再进行waitpid

### 1.2.2 中断、信号

1. 信号处理程序中不能使用printf，原因是printf在更改缓冲区时会加锁，若主程序调用printf时，控制权离开并返回主程序，且返回主程序时发现有待处理且非blocked的信号，进入信号处理程序时也调用printf，则信号程序中的printf由于主程序的printf的锁不得不等待，又由于主程序和信号处理程序处在同一个进程/线程中，因此主程序printf的锁总是无法解除，从而导致死锁
2. 每个线程有独立的信号处理

## 1.3 WSL2

### 1.3.1 网络

1. 宿主机可以用127.0.0.1访问WSL2，反之则不行
2. 当代理软件（Clash）位于Windows上时，Windows配置代理仅需要set http(s)\_proxy="127.0.0.1:7890"，而WSL2在使用export http(s)\_proxy="宿主机IP:7890"之前，要先用cat /etc/resolv.conf | grep nameserver | awk 'print \$2'获取宿主机IP（此外，用hostname -I | awk 'print \$1'获取WSL2自身IP）

## 2 C/C++

### 2.1 算法技巧

#### 2.1.1 预处理

1. 二分前先使数组有序
2. 注意隐藏边界（长度为0,1等）
3. 先排序再计算往往可以简化计算过程
4. 有可能样例输入有序，测试点输入无序
5. 注意图的输入中的重边和自环，以及有向输入转化为无向图

#### 2.1.2 算法执行

1. 二分区间的开闭由具体问题决定（一般一边开一边闭）
2. 递归算法需要数组记录答案时不用“触底”时全部修改，然后利用一个全局的bool变量连续退出，而是可以回溯时逐步修改，从而减小代码复杂度
3. 注意浮点数计算的上下浮动
4. 尝试将 $n!$ （排列）转化为 $2^n$ （组合）

#### 2.1.3 算法评估

1. 计算递归算法复杂度可先计算递归实例的数量

### 2.2 语言特性

#### 2.2.1 基本语法

1. 注意循环嵌套中，循环变量i、j、k等不要重复使用
2. 循环体中的变量地址不变
3. 使用getchar前注意去除cin等留下的回车等干扰字符
4. 注意数组下标越界有可能完全无异常（越在其他变量内部）
5. switch分支结构注意用break
6. 函数调用计数可利用函数体中的局部静态变量
7. 可以使用位域直接操作内存中的位

## 2.2.2 面向对象

1. 对象内部局部变量需要初始化
2. 注意写public（默认为private）
3. 友元函数函数不是成员函数，不能加作用域符号
4. 引用本质只是别名，其创建时不会产生任何构造过程
5. 注意避免自身赋值
6. 当一个内部类或内部对象需要访问外部对象时，尽量通过外部对象成员变量的指针来访问，否则有可能出现构造顺序或访问权限的问题
7. 尽量不要创建野指针，如果不可避免要创建野指针，一定要初始化为nullptr
8. 移动构造、赋值前注意删除当前指针的内容，避免当前指针赋新值后内存泄漏
9. delete前对象最好指针最好不是nullptr，delete后对象指针最好置为nullptr
10. 在返回值和参数均可被析构时，先析构返回值，再析构参数（符合栈的顺序）
11. 虚函数/常量函数不能为静态函数，因为其调用/参数中需要/含有this指针
12. 模板函数将成员函数作为形参时，成员函数应设为静态函数，非静态成员函数因为有this指针形参，参数数量不一致，可能导致错误（sort）
13. std::move()对常引用无效
14. 派生类新定义的非虚函数和新定义的变量会在函数形参为值/引用/指针（所有情况）时被切片
15. 重写函数调用时，与所有当前形式类中的函数同名且参数不同的函数会被隐藏，然后按虚函数机制调用
16. 在派生类没有直接写出新函数的情况下，派生类不会自动生成新的虚函数继承版本，而是在虚函数表中沿用旧版本（注意与重写隐藏的关系）
17. 基类指针指向派生类对象时，调用被基类声明、派生类继承的虚函数不需要dynamic\_cast，调用派生类声明的函数需要dynamic\_cast
18. 模板的声明与实现需要在同一文件中（模板实例化在编译期确定）

## 2.3 Qt

### 2.3.1 绘图

1. Qt画圆的坐标原点为外界矩形的左上角点

### 2.3.2 原理

1. Qt画圆的坐标原点为外界矩形的左上角点

## 3 Python

### 3.1 语言规范

#### 3.1.1 程序结构

1. 引用原生库和手写库
2. 定义全局变量
3. 定义修饰器
4. 定义类（包括函数对象）
5. 定义函数（包括argparse）
6. 定义主函数

### 3.2 语言特性

#### 3.2.1 运行特性

1. 在使用import时正确的路径是针对main.py而言的，而不是针对当前文件而言的
2. global关键字的使用是为了在局部作用域中引用并修改全局变量
3. 闭包函数若要修改上级作用域中的变量，需要用nonlocal关键字
4. Python中只有模块(module)，类(class)以及函数(def、lambda)才会引入新的作用域，其它的代码块(如if、try、for等)不会引入新的作用域，因此在代码块外部可以直接引用代码块内声明的变量
5. Python的变量是动态声明的，未考虑到这点可能出现bug，例如在if语句中声明了变量，若该if语句条件为假，则不仅其内部语句不执行，其内部变量也不会被定义。因此，很多时候有必要在if语句之前声明变量

### 3.3 具体应用

#### 3.3.1 正则表达式

1. 在?、+、\*以及{n,m}后加?表示进行懒惰匹配（与默认的贪婪匹配相反）
2. \b、\$以及^匹配的是单词边界，而非字符（匹配的是“一条线”）

## 4 Javascript

### 4.1 语言特性

#### 4.1.1 函数

1. 闭包中的作用于整个函数的变量为引用，而在某个循环内部的变量为拷贝

#### 4.1.2 对象与原型

1. this永远指向最近的调用者