

/ c++虚函数

/ 二叉树、四叉树、八叉树的应用

+ OS  
第 0 步, 共 1 步

— [https://blog.csdn.net/csdn\\_chai/article/details/78002202](https://blog.csdn.net/csdn_chai/article/details/78002202)

+ 数论算法

+ 迭代器失效  
第 0 步, 共 1 步

— [https://blog.csdn.net/qq\\_37964547/article/details/81160505](https://blog.csdn.net/qq_37964547/article/details/81160505)

+ 堆、堆排序、优先队列

+ 红黑树、平衡二叉树、b树（区别，各自优劣，应用（如stl map用了红黑树，DB中用了B树））

/ 进程和线程的区别，多线程

+ c++ stl

+ multimap

+ c++模板  
第 0 步, 共 1 步

— [https://github.com/wuye9036/CppTemplateTutorial/blob/master/ReadMe.md?](https://github.com/wuye9036/CppTemplateTutorial/blob/master/ReadMe.md?tdsourcetag=s_pctim_aiomsg)  
tdsourcetag=s\_pctim\_aiomsg

/ tcp3次握手，和UDP区别  
第 0 步, 共 1 步

— UDP 只要发出去就可以，TCP 也是这样吗？这个问题刚开始没有太理解，当时不是，因为 TCP 按收到报文顺序不一定是发送的顺序。面试官说这是 TCP 内部的实现。然后我问是 socket 编程的时候的区别吗，然后 TCP 和 UDP 名字不一样，和...

/ c++ static

- / 操作系统调度、页表、缓存算法 (LRU, RR等)
- / IP, 部分地址的默认功能、掩码、DNS等等
- / 假设有两个很大的数组, 每个几万条数据, 如何快速求交集? 用Redis的Set可以做, 或者用BitMa...
- + 排序算法的优劣, 稳定与不稳定
- + DFS&BFS ★
- / 字符串压缩->哈夫曼编码
- + c++ 指针和引用
- / c++多态 (虚函数)
- / c++11 新特性: 智能指针等
- + 手撕快排, 利用快排结构优化某些问题
- + 字符串匹配
- + kd树
- + 红黑树
- + 卡马克算法
- + 并查集

+ 全局解释器锁

+ python内存管理

/ new、malloc的区别

第 0 步, 共 7 步 · 0

- <https://blog.csdn.net/weibo1230123/article/details/81980889>
- malloc开辟空间类型大小需手动计算, new是由编译器自己计算;
- malloc返回类型为void\*,必须强制类型转换对应类型指针, new则直接返回对应类型指针;
- malloc开辟内存时返回内存地址要检查判空, 因为若它可能开辟失败会返回NULL; new则不用判断, 因为内存分配失败时, 它会抛出异常bad\_alloc,可以使用异常机制;
- malloc/free为函数只是开辟空间并释放, new/delete则不仅会开辟空间, 并调用构造函数和析构函数进行初始化和清理
- new/delete在开辟内存空间时, 不仅会开辟空间, 还会调用构造函数和析构函数, 在返回地址时则会调用析构函数, 而在delete时则会查看内存上对象个数, 从而根据个数count确定调用几次析构函数, 从而完全清理内存上所有对象
- 字节, 而在delete时则会查看内存上对象个数, 从而根据个数count确定调用几次析构函数, 从而完全用new分配的变量是存放于堆内存中的, 但是返回的指针变量是存放在栈中的。当我们在一个子函数中new了一个变量, 但是在函数返回时既没有保存new返回的指针

/ inline、define的区别

第 0 步, 共 3 步

- inline: 直接在class类定义中定义各函数成员, 系统将他们作为内联函数处理; 成员函数是内联函数, 意味着: 每个对象都有该函数一份独立的拷贝
- inline进行参数有效性检测及使用C++类的成员访问控制。
- inline: 内联函数对编译器提出建议, 是否进行宏替换, 编译器有权拒绝

+ 多态的实现原理

+ 虚析构函数(原理、使用)

+ gcc里在main函数之前执行代码的方法-\_\_attribute\_\_关键字

第 0 步, 共 1 步

- [https://blog.csdn.net/king\\_cpp\\_py/article/details/79435351](https://blog.csdn.net/king_cpp_py/article/details/79435351)

+ delete this之后会发生什么? delete this的使用场景

第 0 步, 共 1 步

- <https://www.v2ex.com/t/559047>

- + 0x5f3759df
- + 如何实现只能创建在堆或者栈上的类  
第 0 步, 共 4 步
  - <https://blog.csdn.net/KingOfMyHeart/article/details/90344362>
  - <https://blog.csdn.net/dove1202ly/article/details/81747437>
  - 栈: 通过private重载禁用new
  - 堆: 通过private或protect禁用析构函数
- + vector、map的底层实现
- + 模板偏特化、SFINAE ★
- + volatile的原子性  
第 0 步, 共 1 步
  - <https://www.jianshu.com/p/f74044782927>
- + c++编译过程, 如何查看编译过程中生成的中间文件 ★  
第 0 步, 共 1 步
  - <https://blog.csdn.net/u012662731/article/details/78520349>
- / c++程序内存布局、const变量存储在那个区域
- + 名字修饰、如何查看符号表
- + 引用和指针的区别 ★
- + 什么情况下出现重定义错误, 如何避免
- / extern关键字的用法  
第 0 步, 共 1 步
  - extern 'c' 要求编译器按照c风格编译函数名 (e.g. 解决跨平台时相同函数由于编译器原因名称不一样导致的链接错误)

+ 内存泄漏出现的场景和解决方法

/ 智能指针的原理和使用，为什么要有weak\_ptr  
第 0 步，共 3 步



- <https://www.cnblogs.com/TianFang/archive/2008/09/20/1294590.html>
- boost::weak\_ptr与boost::share\_ptr或boost::weak\_ptr转类同不，这也说明，进行该对象的内存管理的是那个强引用的boost::share\_ptr。boost::weak\_ptr只是提供了对管理对象的一个访问函数
- expired()用于检测所管理的对象是否已经释放；lock()用于获取所管理的对象的强引用指针。

/ python的GC机制  
第 0 步，共 1 步



- <https://zhuanlan.zhihu.com/p/83251959>

+ 浏览器输入网址的处理流程，arp协议处理流程



+ time\_wait出现的时机，过多时处理措施

+ close\_wait出现的时机

+ 网络相关的shell命令

/ 拥塞避免的详细实现  
第 0 步，共 1 步



- <https://zhuanlan.zhihu.com/p/37379780>

+ 流量控制和可靠传输的实现原理，滑动窗口的原理



+ UDP和TCP的区别



+ keepalive、SO\_REUSEADDR、SO\_REUSEPORT

+ HTTP的常用请求方法，option的作用



- + HTTP的错误码及含义
- + HTTP常用header
- + HTTP报文格式, 如何使用C++填充HTTP报文 ★
- + linux查看系统占用的命令  
第 0 步, 共 1 步
  - top
- + epoll的实现, epoll处理过程中是否加锁, 什么时候加锁 ★
- + epoll边缘触发和水平触发的区别  
第 0 步, 共 2 步
  - epoll有EPOLLIT和EPOLLET两种触发模式, LT是默认的模式, ET是“高速”模式。LT模式下, 只要这个fd还有数据可读, 每次 `epoll_wait` 都会返回它的事件, 提醒用户程序去操作, 而在ET (边缘触发) 模式
  - <https://blog.csdn.net/crazywdy/article/details/33790945>
- + 阻塞、非阻塞、同步、异步的特点 ★
- + linux容器的实现, cgroup和namespace
- / grep命令的使用, 从日志中过滤耗时过长的日志  
第 0 步, 共 1 步
  - <http://www.zsythink.net/archives/1733/>
- + cfs调度算法的特点和实现原理
- + 硬连接、软连接的区别, inode的内容
- + EAGAIN的含义和出现的场景
- + 进程和线程的区别 ★

- / 线程进行同步的机制 ★
- + IPC机制
- / pcb中的内容 ★
- + Linux系统中的锁
- + MySQL数据组织(索引段、回滚段、数据段, 页面、行记录等3
- + innodb和MyISAM的区别
- + hash索引和b+树索引的特点
- + 索引失效条件、联合索引的最左匹配
- + 辅助索引
- + hash join的原理
- + left join
- + 红黑树的特点, 和B+树的对比
- / 模板编程 ★
- + 字符串匹配(朴素匹配算法) ★
- + awk
  - 第 0 步, 共 2 步
    - linux中awk进行日志文本的处理过滤  
<https://blog.csdn.net/DittyChen/article/details/78087963>

— <https://www.cnblogs.com/ftl1012/p/9250541.html>

/ socket  
第 0 步, 共 1 步

— [https://blog.csdn.net/weixin\\_39634961/article/details/80236161](https://blog.csdn.net/weixin_39634961/article/details/80236161)

/ fork-exec

+ <https://github.com/NLP-LOVE/ML-NLP>

/ 图形学  
第 0 步, 共 2 步

— [https://blog.csdn.net/csdn\\_chai/article/details/78028344](https://blog.csdn.net/csdn_chai/article/details/78028344)

— <https://blog.csdn.net/zhaotianyu950323/article/details/79955805>

+ 模取幂运算 ( $a^b \bmod c$ )  
第 0 步, 共 3 步

— <https://blog.csdn.net/dremi/article/details/1568221>

— <https://blog.csdn.net/najiuzheyangbaacm/article/details/80456951>

— [https://blog.csdn.net/DBC\\_121/article/details/77646508](https://blog.csdn.net/DBC_121/article/details/77646508)

+ 线段树  
第 0 步, 共 1 步

— <https://baijiahao.baidu.com/s?id=1605870136961096251&wfr=spider&for=pc>

+ <https://github.com/OUCMachineLearning/OUCML/tree/master/%E4%BB%A3%E7%A0%81%E9%...>

+ 数据结构与算法-C++实现动态变长数组  
第 0 步, 共 1 步

— <https://www.cnblogs.com/trialley/p/11248890.html>

/ 四种cast  
第 0 步, 共 4 步

— 很棒 <https://www.jianshu.com/p/5163a2678171>

— static\_cast<引用、指针等>(要cast的变量or对象) 不安全



- `dynamic_cast<类对象的引用、指向对象的指针>()` 安全
- `int *c=const_cast<int*>(&b)` b为const int&解除b的const属性，即可以使用c修改b的值

/ C++提供虚继承机制，防止类继承关系中成员访问的二义性

/ static  
第0步，共4步

- 在域作用符内：保证每一次调用时该变量保留上一次的值（存储于全局数据区）第一次调用时初始化!!
- 在域作用符外，声明该变量/函数仅属于本文件
- 静态成员（变量/函数）：类的一部分（不属于对象，属于类本身）1、静态成员之间可以互相访问；2、对象可以任意访问静态成员；3、静态成员不可访问对象
- [https://blog.csdn.net/weixin\\_40311211/article/details/82851300](https://blog.csdn.net/weixin_40311211/article/details/82851300)

/ 变量的内存分布  
第0步，共4步

- 栈区：由编译器自动分配释放，像局部变量，函数参数，都是在栈区。会随着作用于退出而释放空间。
- 堆区：程序员分配并释放的区域，像`malloc(c)`,`new(c++)`
- 全局数据区(静态区)：全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。程序结束释放。
- 代码区

/ 智能指针 解环  
第0步，共1步

- [https://blog.csdn.net/weixin\\_36750623/article/details/84831969](https://blog.csdn.net/weixin_36750623/article/details/84831969)

/ delete this  
第0步，共2步

- 普通成员函数中会执行完，但是后续调用对象内容会输出未初始化的内容
- 析构函数会无限调用并无限执行delete this之前的代码，直到栈溢出

+ 网络上客户端与服务器在公网下通讯

+ 传输结构体

/ 结构体大小  
第0步，共2步

— <https://www.cnblogs.com/smile-812/p/7897187.html>

小的数据模数的内存地址是0时，数据是对齐的。例如：WORD值应该是总是从被2除尽的地址开始，而DWORD值应该总是从被4除尽的地址开始，数据对齐不是内存结构的一部分，而是CPU结构的一部分

int大小

第0步，共1步

— <https://www.cnblogs.com/downey-blog/p/10469977.html>

指针大小

第0步，共1步

— <https://www.cnblogs.com/downey-blog/p/10469977.html>

堆栈默认大小——1MB

字节对齐

第0步，共1步



— [https://blog.csdn.net/qq\\_34342154/article/details/79242188](https://blog.csdn.net/qq_34342154/article/details/79242188)

fork-exec

第0步，共2步



fork:系统让新的进程与旧的进程使用同一个代码段，因为它们的程序还是相同的，对于数据段和堆栈段，系统则复制一份给新的进程（修改时分离）

exec: 一个进程一旦调用exec类函数，它本身就“死亡”了，系统把代码段替换成新的程序的代码，废弃原有的数据段和堆栈段，并为新程序分配新的数据段与堆栈段，唯一留下的，就是进程号

struct和class的区别

第0步，共2步

struct D.A //公有继承

— {

public:

默认访问权限: struct作为数据结构的实现体，它默认的数据访问控制是public的，而class作为对象的实现体，它默认的成员变量访问控制是private的

+ 进程终止&僵尸进程 p52



int类型和指针的大小

+ 第0步，共5步



— <https://www.cnblogs.com/downey-blog/p/10469977.html>

统中，int型为16位大小，两字节

— 32位系统中，int型为32位大小，四字节

— 32位系统中，long型为32位大小，4字节

— 64位系统中，long型为64位大小，8字节

— 在32位系统下指针类型为32位，在64位系统下指针类型为64位



- 指针本质上是变量，它的值是内存中的地址，既然需要通过指针能够访问当内存当中所有的数据，那么这个指针类型的宽度至少要大于等于地址总线的宽度

## / define和const的区别 第 0 步，共 2 步

- [https://blog.csdn.net/ZYZMZM\\_/article/details/83302084](https://blog.csdn.net/ZYZMZM_/article/details/83302084)
- 首先，本质上两者不同，define修饰出来的是常量！并且是真常量！
- 而const修饰的是假常量，它本质还是变量！只不过编译器不让你修改！

## / const成员函数 第 0 步，共 2 步

- <https://www.cnblogs.com/cthon/p/9178701.html>
- 本质上，const指针修饰的是被隐藏的this指针所指向的内存空间，修饰的是this指针

## / trie树 第 0 步，共 3 步

- 可以对Trie字典树做些限制，比如每个节点只能有3个子节点，左边的节点是小于父节点的，中间的节点是等于父节点的，右边的子节点是大于父节点的，这就是三分搜索Trie字典树(Ternary Search)
- 例如插入单词 "pain"，如果用普通字典树，需要插入 4 个节点，但是用三分搜索Trie字典树，只需要插入 3 个节点，所以如果往Trie中插入一个单词，需要把该单词的最后一个字符的节点的 isWord 设置为 true
- Trie查找操作就比较简单了，遍历带查找的字符串的字符，如果每个节点都存在，并且待查找字符串的最后一个字符对应的Node的 isWord 属性为 true，则表示该单词存在

## + epoll 第 0 步，共 1 步

- <https://www.jianshu.com/p/127e10ff2340>

## + 管道!!!



## + ssl

## + 服务端开发 第 0 步，共 3 步



- 服务器缓存
- 网络
- 数据库