- 进程和线程的区别
  1. 进程是资源分配的基本单位，线程是程序执行的基本单位
  2. 进程有自己的资源，多个线程共享进程资源
  3. 进程的系统开销远大于线程
  4. 进程的切换成本更高

  同一进程中的多个线程共享代码段（代码和常量），数据段（全局变量和静态变量），扩展段（堆存储）。但是每个线程拥有自己的栈段，栈段又叫运行时段，用来存放所有局部变量和临时变量
- 协程和线程的区别
  1. 进程和线程都是同步机制，协程是异步机制
  2. 协程能保留上一次执行的状态
  3. 一个线程可以有多个协程
  4. 更轻量级
  5. 非抢占
- 进程通信方式

  信号量，消息传递，共享内存，管道，socket

  管道可以分为两类：匿名管道和命名管道。匿名管道是单向的，只能在有亲缘关系的进程间通信；命名管道以磁盘文件的方式存在，可以实现本机任意两个进程通信。

  Socket 可用于不同机器间的进程消息传递。
- TCP 连接握手不是 2 次或 4 次
  1. 若客户端发送 A 报文，丢失。然后超时后发送 B，被服务器接收后，正常建立连接后关闭连接。此时若 A 报文到达服务器，服务器收到后进入连接状态，发送确认报文，但因为客户端已经进入关闭状态，所以会造成服务器长时间单方面等待。
  2. 三次连接才能让双方确实双方的发送和接受能力正常
  3. 告知对方自己的初始序列号，并且确认对方受到自己的初始序列号

  第一次握手服务器确认“自己收，客户端发”正常

  第二次客户端确认“自己发，对方收，自己收，对方发”正常

  第三次服务器确认呢“自己发，对方收”正常。
- 四次挥手的原因

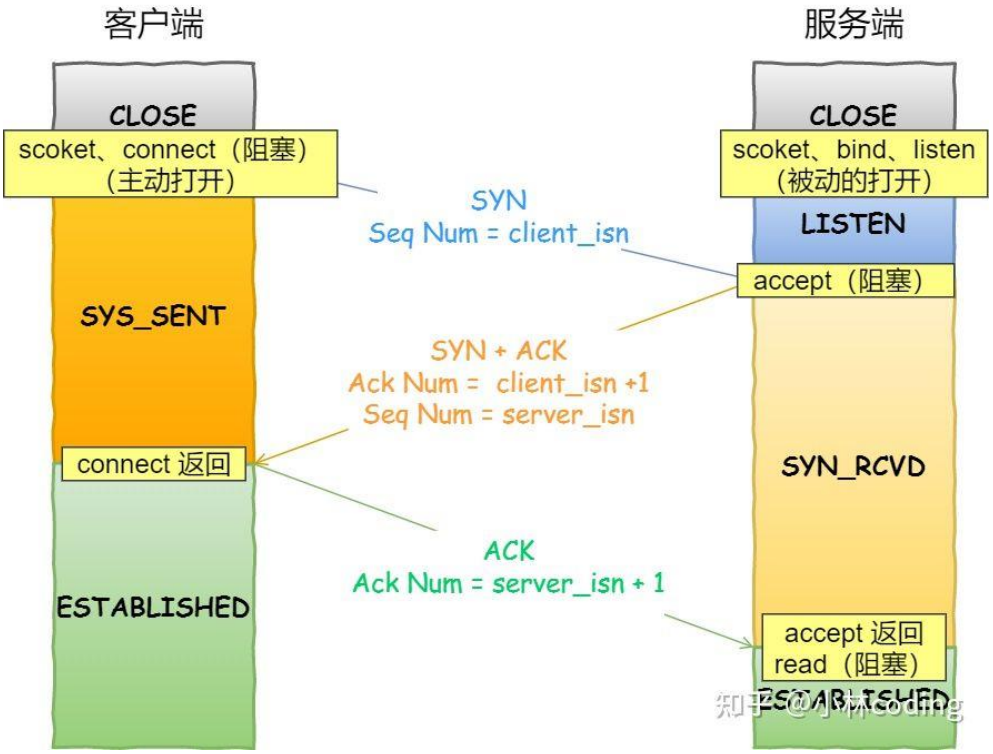
  服务器受到 FIN 后可能还有数据没传输完成，但先返回 ACK，需要等到数据传输完成后再返回一次 FIN。
- TIME_WAIT
  1. 要确保服务器受到报文后关闭，如果客户端的 ACK 丢失了，那么 2 后会开始重传 ACK，2 是客户端 ACK 超时时间和服务器 FIN 传输时间
  2. 要确保下次连接不再受到这次的垃圾报文，因为 TCP 中 2 后报文的序列号会改变。
- Keep_alive

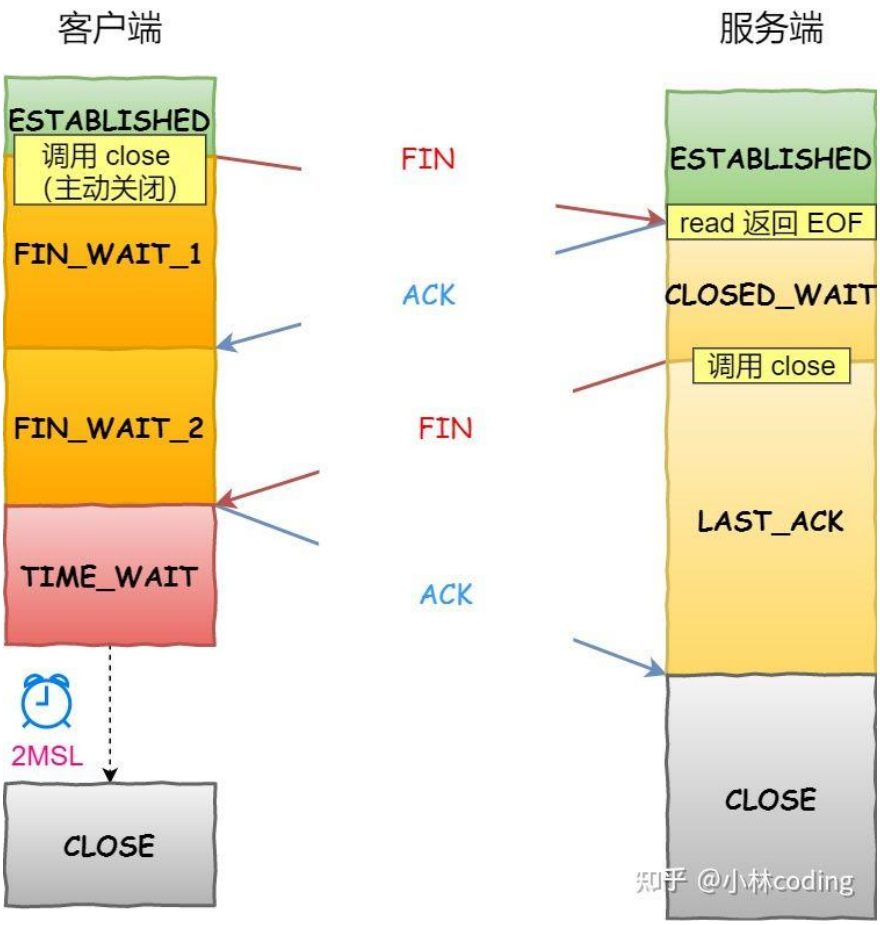
  长连接，即网页打开后，TCP 不会断开，短连接是每次有一个 HTTP 请求，就申请一个连接，每次申请资源都需要重新连接，但长连接也有时间限制，可以再服务器上设置，需要服务器和客户端都支持。
- TCP 和 UDP 的区别
  1. UDP 是不可靠的连接，常用于 DNS，视频，语音聊天等；TCP 是可靠的连接，需要通过三次握手建立连接
  2. TCP 是面向字节，UDP 是面向报文
  3. TCP 有拥塞控制和流量控制

4. UDP 更轻量
5. TCP 是按序到达的，UDP 不一定

- Socket

  客户端执行 socket()(创建一个 socket 描述符（socket descriptor）),connect()()
  服务器执行 socket(),bind()(分配特定地址和端口号给服务器),listen();





- 访问百度的过程
  1. DNS 域名解析
  2. 建立 TCP 三次握手连接
  3. 建立 HTTP 连接
  4. 服务器像浏览器发送 HTML 代码
  5. 浏览器解析代码并渲染给客户
- 内存泄漏

  由于程序的疏忽使分配的内存没有回收，不再能继续使用。

  堆内存泄漏

  系统资源泄露（Resource Leak）.主要指程序使用系统分配的资源比如 Bitmap,handle ,SOCKET 等没有使用相应的函数释放掉，导致系统资源的浪费，严重可导致系统效能降低，系统运行不稳定。
- 数据库索引

  索引是文件，包含对数据表内所有文件的引用指针，占用物理空间。

  索引是一种数据结构，有 B 树或者 B+树
- 为什么不用二叉树
  1. 二叉树需要旋转保持平衡，则需要把整个二叉树都放在内存里面
  2. 二叉树中逻辑很近的节点可能离得很远，无法利用局部性原理。
  3. 二叉树慢每次只能分为两个区间
- B 树

  因为节点是区间范围，所以比较大，直接申请页大小的空间，计算机内存分配是按页

对齐的，磁盘 IO 每次读取一页所以只需要一次 IO。

- B+树

  B+树可以很好的利用局部性原理，若我们访问节点 key 为 50，则 key 为 55、60、62 的节点将来也可能被访问，我们可以利用磁盘预读原理提前将这些数据读入内存，减少了磁盘 IO 的次数。

  B+树的叶子节点都是通过链表连接起来的，在磁盘中顺序存储，一次访问即会把所有节点都读入内存。

  B+树只有叶子节点有 data 域，则一页中可以存放的数据量更大，总的磁盘 IO 次数少。

- GET 和 POST 的区别

  GET 和 POST 是 HTTP 协议中发送请求的两种方式。

  1. GET 通过 cookie 或者 url 传递参数，POST 放在 body 里面
  2. GET 短，POST 长
  3. POST 更安全
  4. GET 多次请求同一资源会得到相同的结构，POST 不是

- 事务的特性

  1. 一致性 **consistency**：事务会从一个一致性状态转向另一个一致性状态。
  2. 原子性 **atomicity**：事务包含的所有操作会全部成功或者全部回滚。
  3. 隔离性 **isolation**：并发事务是隔离的
  4. 持久性 **durability**：一个事务一旦提交，那么对数据库的改变就是永久的，即便在遇到故障的情况下也不会丢失提交数据的操作。

- 什么是数据库事务

  事务是不可分割的数据库操作序列，是数据库并发操作的基本单位。

  数据库事务通过重做日志（redo log）和回滚日志（undo log）实现。

  重做日志将每个提交的事务的所有操作都记录，实现持久性和原子性

  回滚日志实现一致性，通过回滚执行一条相反的 undo log。

- 四种隔离级别

  1. 可串行化
  2. 可重复读
  3. 读已提交
  4. 读未提交

- MVCC

  MVCC 是数据库的多版本并发控制，通过保存数据在某个时间点的快照实现，每个事务根据开始时间不同，对同一张表看到的数据可能不同。

  InnoDB 每一行数据都有一个隐藏的回滚指针，用于指向该行修改前的最后一个历史版本，这个历史版本存放在 undo log 中。如果要执行更新操作，会将原记录放入 undo log 中，并通过隐藏的回滚指针指向 undo log 中的原记录。其它事务此时需要查询时，就是查询 undo log 中这行数据的最后一个历史版本。

  MVCC 最大的好处是读不加锁，读写不冲突，极大地增加了 MySQL 的并发性。通过 MVCC，保证了事务 ACID 中的 I（隔离性）特性。

- 乐观锁和悲观锁

  悲观锁：先把事务锁起来，再进行读写，直到提交事务。

  乐观锁：操作数据时不会对事务加锁，提交的时候通过 version 机制来验证是否冲突

  写冲突很多时不适合乐观锁

- 慢查询优化

1. 首先分析语句看是否 load 了多余的数据。
2. 查看语句的执行计划，看索引的使用情况，修改语句或者修改索引，提高索引命中率。
3. 分表

- 重载为什么可以仅靠参数列表的不同：

因为编译时会给他们不同的函数名。构造函数可以被重载，析构函数不可以。
构造函数可以重载，析构函数不可以。
隐藏的参数可以不同，函数体不同（普通成员函数）；重写或者覆盖仅仅函数体不同（虚函数）。

- 多态

编译时多态：运算符重载，函数重载
运行时多态：虚函数、继承
虚函数机制在构造函数中不工作，因为子类先调用父类构造函数，而此时虚函数表还没有被构建，调用虚函数可能会调用还未被构造的子类成员，
构造函数不能是虚函数：构造函数不可能被继承，写成虚函数那么父类无法分配空间。
静态函数不属于任何类对象，因此也不能是虚函数。
析构函数必须是虚函数，不然父类指针指向子类对象，析构时只调用父类的析构函数会造成内存泄漏。可以写成纯虚函数，这样类就是抽象类
虚函数在表现多态性时不能时内联函数。

- new delete

1. malloc 是堆，new 是自由存储区
2. malloc 返回 void*,new 返回对象类型指针
3. malloc 需要指定内存大小，new 自动分配
4. malloc 失败时返回 NULL，new 失败时返回异常
5. new 的实现是先底层调用 operate new（调用 malloc）分配内存再调用构造函数，再返回相应的指针，delete 先调用析构函数再底层调用 free 释放内存。

New 需要在堆中分配内存，时间很慢，还可能存在内存不顾的异常，placement new 可以在预先准备好的内存缓存区里构造对象，不需要查找内存，内存分配的时间是常数，不会出现内存不足的情况。

- C++ 标准库（STL）

1. Vector

Reverse()不能创建对象，只是分配空间，resize 创建对象，可以直接用[]访问。
相关问题：
扩容过程：(36 条消息) c++STL vector 扩容过程_低头看天，抬头走路的博客-CSDN 博客_c++ vector 扩容
Vector 迭代器失效：(36 条消息) C++ vector 迭代器失效问题_Gerald Kwok 的博客-CSDN 博客_c++ vector 迭代器失效
(36 条消息) C++中 vector 迭代器失效问题及其解决方法_慕白昂的博客-CSDN 博客_vector 迭代器失效问题
Emplace 和 insert、emplace_back 和 push_back:
C++ emplace_back - 简书 (jianshu.com)
Vector 是一个类，有三个指针 myfirst、mylast、myend 分别表示首地址，元素容量地址，容器容量地址。
扩容和失效：当添加元素发现容量不足时，就会自动扩容，扩容时会找一个更大

的连续内存，原来的内存就会失效，扩容一般为当前大小的两倍。

2. Deque:map 数组来管理分散的几片连续内存，支持[]，支持两端 pushpop
3. List:双向链表实现，不能[]
4. Set、map 用红黑树实现

哈希表（unordered_set，unordered_map）和红黑树：

哈希表无序，冲突的地方可以有一个单链表

红黑树是平衡二叉树，但不是高度平衡（AVL）,维护成本低

红黑树特点:

1. 根节点和叶子节点（都是空节点）都是黑色。
2. 每个红色节点的两个子节点都是黑色。
3. 从任一节点到其每个叶子的所有路径都包含相同数目的黑色节点。

每次插入和删除都通过变色和旋转来保证红黑树符合上面的规则.

- C++ 11 新特性

1. 右值引用、移动语义、完美转发
2. 智能指针

Shared_ptr:每多一个指针指向内存，计时器加一，做函数形参也加一，退出函数减一。计数器减到 0 就释放该指针。

Make_shared 的使用：make_shared 理解_CPriLuke 的博客-CSDN 博客_make_shared

Weak_ptr：从 shared_ptr 创建，获得一些状态信息

unique_ptr：表示指针不可以拷贝复制

和 new 的区别：

普通指针在作用域外被释放，但 new 不会主动释放空间，但指向 new 出的空间的指针已经被释放，会造成内存泄漏。作用域外 New 指针释放，但智能指针有别的方式可以再访问。

New 和普通指针的区别;

普通指针因为指向的内容在作用域结束后被销毁，所以不能作为返回值返回，new 出来的指针是可以的。

- Public：任意成员函数

Protected:本类或子类

Private:本类

- Static

静态变量：存放在静态（全局）存储区，但作用域不变，不能跨文件

静态成员变量：

仍存放在全局数据区（具体对象的内存在堆里面）被多个对象共享，在类中定义，在类体外初始化时分配内存。

静态成员函数：

（1）静态成员函数只能访问静态成员变量。只能去调用其他静态成员函数。

（2）相比普通成员函数，不管有没有创建对象，主函数都可以调用静态成员函数。

- Const:

被修饰的值是只读变量，必须在定义时给他赋初值。

常量指针（底层 const）：int const* p = &temp;const int *p = &temp;

不能通过指针改变指向的值，指针指向的地址可以改变。

指针常量（顶层 const）：int*const p = &temp;
可以通过指针改变指向的值，但指针的指向不能改变
两者结合使用也可。

- C++模板
  template <typename T>
  函数模板重载
- 深浅拷贝
  当类持有动态分配的内存、指向其他数据的指针等的时候，需要深拷贝，保证拷贝的类里的指针指向自己的空间。
  普通拷贝构造函数是浅拷贝，如果显示的定义拷贝构造函数，并且为它分配新内存空间，就是深拷贝。
  写时拷贝：深拷贝读取到这个空间时不会真的分配内存，真正需要拷贝时才开辟空间。
  采用引用计数，多一个指针指向空间就加一，当需要修改空间的值时，开辟自己的空间，原来的空间的引用计数减一，现在的加一。
- 空类在声明时不会产生任何构造函数，只会生成一个字节的占位符；
  但空类在定义对象时会生成六个默认的成员函数。
- 赋值函数与拷贝构造函数：
  赋值函数是当两个对象都已经存在，拷贝构造函数被拷贝的对象不存在；
  拷贝构造大多数是复制，赋值函数是引用。
  赋值函数要先把本来的内存释放掉，而拷贝构造是初始化对象内存。
- 内存：
  代码区：存放程序的代码，即 CPU 执行的机器指令，一般只读。
  静态区（全局区）：存放静态变量和全局变量。
  常量区：存放常量(程序在运行的期间不能够被改变的量，例如: 10，字符串常量”abcde”， 数组的名字等)
  堆区：由程序员调用 malloc()函数来主动申请的，需使用 free()函数来释放内存。
  栈区：存放函数内的局部变量，形参和函数返回值。
- 类型转换
  static_cast<newType>(data) ：
  （1） 近似类型转换，如 int 转 double,short 转 int,const 转非 const,向上转型等。
  （2） void*指针和具体类型指针之间的转换，例如 void *转 int *、char *转 void *等。
  const_cast<newType>(data) ：
  用来将 const/volatile 类型转换为非 const/volatile 类型。
  reinterpret_cast<newType>(data) ：
  两个具体类型指针之间的转换、int 和指针之间的转换。非常简单粗暴，但是风险很高。
  dynamic_cast<newType>(data) ：
  用于在类的继承层次之间进行类型转换，它既允许向上转型（Upcasting），就是把继承类指针转换为基类指针；也允许向下转型（Downcasting）。向上转型是无条件的，不会进行任何检测，所以都能成功；向下转型的前提必须是安全的，要借助 RTTI 进行检测，所有只有一部分能成功。（RTTI 机制不了解可以去查一下）

- DNS 解析过程

  1. 输入域名后，查询本地 DNS 服务器（一般有运营商提供）
  2. 本地服务器会先查缓存，如果缓存中有此条则返回，若没有则继续像 DNS 根服务器进行查询。
  3. 根服务器没有说明 IP 地址和域名的具体关系，而是告诉本地服务器去预服务器查，并给出域服务器地址。
  4. 本地服务器继续向域服务器发送请求，例如.com 域服务器，域服务器会告诉本地服务器域名的解析服务器的地址。
  5. 本地服务器再解析服务器找到 IP 地址，然后返回给电脑并且把这个关系放入缓存。

  DNS 的 TTL 参数（Time_To_Live）

  本地 DNS 服务器对于域名缓存的最长时间

- url（统一资源定位符）解析过程

  url 组成部分：

  1. 传输协议：

     HTTP 协议（超文本传输协议），基于请求/响应的无状态协议，支持图片视频等，由于 HTTP 自己本身是一个明文协议，每个 HTTP 请求和返回的数据在网络上都是明文传播，无论是 url、header 还是 body。 只要在网络节点捉包，就能获取完整的数据报文，要防止泄密的唯一手段就是使用 HTTPS（用 SSL 协议协商出的密钥加密明文 HTTP 数据）。

     HTTPS（HTTP+ssl）(加密传输)通过证书等来保障网站真实性，建立加密的信息通道，保证数据的完整性，例如用于支付网站。

     ftp(文件上传下载协议)一般用于服务器和客户端文件的直接传输。
  2. 服务器域名

     根据 DNS 解析出 IP
  3. 端口号

     用来区分一台服务器上的不同项目，用户必须用正确的端口才能访问到网页。
  4. 问号传参

  一、URL 解析：

  HTTP 基于 TCP/IP 协议，浏览器作为 HTTP 客户端通过 URL 向 HTTP 服务端即 WEB 服务器发送请求，服务器收到请求后，向客户端发送响应信息。

  一个 HTTP 请求报文由请求行，请求头部，空行。

  请求行分为三个部分：请求方法、请求地址 URL 和 HTTP 协议版本，它们之间用空格分割。

  如果出现中文要编码 jiema。

  二、缓存检查

  所谓的浏览器缓存就是浏览器将用户请求过的资源存储到本地电脑。当浏览器再次访问时就可以直接从本地进行加载，不需要去服务端进行请求

  三、DNS 解析

  四、建立 TCP 通道

  浏览器通过 DNS 解析获得 IP 地址后，就可以向服务器发送 TCP 连接请求，通过 TCP 三次握手建立连接，浏览器就可以把 HTTP 请求数据发送给服务器。

  五、服务器接收到请求后，会根据端口号找到资源文件读取文件内容返回给客户端，

返回的称为响应报文，包含 HTTP 响应码，响应头和响应主体。

六、四次挥手关闭 TCP 连接

七、客户端渲染

- Linux 命令

找文件：find 搜索路径 [选项] 搜索内容

-name: 按照文件名搜索；

-iname: 按照文件名搜索，不区分文件名大小；

-inum: 按照 inode 号搜索；

-size[+-]大小：按照指定大小搜索文件

-atime [+-]时间: 按照文件访问时间搜索

-mtime [+-]时间: 按照文改时间搜索

-ctime [+-]时间: 按照文件修改时间搜索

whereis

查看内存使用：

1. Free
2. Cat /proc/meminfo
3. Top 显示内存和 CPU 使用量

top -p pid

查看磁盘占用情况：

df 输出磁盘文件系统的使用情况。

du 命令 du -sh file 可以查看一个文件或目录的磁盘占用情况。-s 显示总用量，如果查看目录时不加-s 则显示目录下各个文件的情况。-h 以合适单位显示大小。

在日志中查找：

cat -n test.log |grep "debug" >debug.txt

进入 vim 编辑模式：vim filename

输入“/关键字”,按 enter 键查找

查找下一个，按“n”即可

找文件中的字符串：

```
grep 'word' file1 file2 file3
```

下载文件到本地(上传)：scp

查看当前进程状态：ps

根据进程名看进程号：

Awk 文本分析

- 海量数据问题

1、最简单的方法就是快排，取 topk

2、局部淘汰法。用一个容器保存前 k 个数，然后将剩余的所有数字——与容器内的最小数字相比，如果所有后续的元素都比容器内的 k 个数还小，那么容器内这 k 个数就是最大 k 个数。如果某一后续元素比容器内最小数字大，则删掉容器内最小元素，并将该元素插入容器，最后遍历完所有的数，得到的结果容器中保存的数即为最终结果了

3、分治法。将 1 亿个数据分成 100 份，每份 100 万个数据，找到每份数据中最大的 10000 个，最后在剩下的 100*10000 个数据里面找出最大的 10000 个。100 万个数据里面查找最大的 10000 个数据的方法如下：用快速排序的方法，将数据分为 2 堆，

如果大的那堆个数 N 大于 10000 个，继续对大堆快速排序一次分成 2 堆，如果大的那堆个数 N 大于 10000 个，继续对大堆快速排序一次分成 2 堆，如果大堆个数 N 小于 10000 个，就在小的那堆里面快速排序一次，找第 10000-n 大的数字；递归以上过程，就可以找到第 1w 大的数。参考上面的找出第 1w 大数字，就可以类似的方法找到前 10000 大数字了。此种方法需要每次的内存空间为 10^6*4=4MB，一共需要 101 次这样的比较。

4、采用最小堆。首先读入前 10000 个数来创建大小为 10000 的最小堆，建堆的时间复杂度为 O（mlogm）（m 为数组的大小即为 10000），然后遍历后续的数字，并于堆顶（最小）数字进行比较。如果比最小的数小，则继续读取后续数字；如果比堆顶数字大，则替换堆顶元素并重新调整堆为最小堆。整个过程直至 1 亿个数全部遍历完为止。然后按照中序遍历的方式输出当前堆中的所有 10000 个数字。该算法的时间复杂度为 O（nmlogm），空间复杂度是 10000（常数）。

- select、poll 和 epoll

单线程或者单进程来监听文件描述符（socket）的就绪状态，如果就绪则返回，否则会超时。

selete:轮询的方式监听。通过检查存放 fd 标志位的数据结构来实现，单个进程可监听的 fd 数量受限。时间复杂度 O(n),仅知道有 I/O 事件发生，不知道是哪几个流，无差别轮询所有流，找出能读出或写入数据的流，并对其进行操作，同时处理的流越多，轮询时间越长。

Poll：轮询。描述 fd 的方式不同。O(n),将用户传入的数组放入内核空间，然后查询每个 fd 对应的设备状态，但是因为它是链表存储的，所以大小不受限。

Epoll：事件触发，I/O 通知机制。时间复杂度 O(1)会把某个流发生了什么事件告诉我们，每个事件绑定上 fd。

EPollLT 和 EPollET:

- lambda 匿名函数

- volatile 关键字

  编译器不再优化，取值会从内存中固定的地址取。

- 必须用初始化列表的情况：
  1. 数据成员是对象，并且这个对象只有含参数的构造函数，没有无参数的构造函数；
  2. 对象引用或者 cosnt 修饰的数据成员
  3. 子类初始化父类的私有成员，需要在(并且也只能在)参数初始化列表中显示调用父类的构造函数

  类对象的构造顺序显示，进入构造函数体后，进行的是计算，是对成员变量的赋值操作，显然，赋值和初始化是不同的，这样就体现出了效率差异。

- 初始化 string 类

- MYSQL
  1. PHP mysqli_fetch_array() 函数从结果集中取得一行作为关联数组，或数字数组，或二者兼有返回根据从结果集取得的行生成的数组，如果没有更多行则返回 false。

  while($row = mysqli_fetch_array($retval, MYSQLI_ASSOC))
  2. Join 多表联合查询
  3. Alter
  4. 单列索引和复合索引的底层实现

5. VARCHAR 可变长，但会留下一个字节来存长度，CHAR 不可变长

- 虚拟内存

  虚拟的连续地址->实际上不连续的物理地址（包括内存和磁盘）

  请求分页存储管理。

  请求分段存储管理。

  请求段页式存储管理。

- 聚簇索引和非聚簇索引

  索引 B+的叶子节点存储了整行数据的是聚簇索引，即将数据与索引放在了一起；

  索引 B+树的叶子节点只存储了主键的值被成为非聚簇索引

- ACID

  原子性（Atomicity，或称不可分割性）：undo log

  一致性（Consistency）

  隔离性（Isolation）:写写（锁机制）、写读（MVCC）

  持久性（Durability）:redo log

- MySQL 优化

  1. 选取最适用的字段属性
  2. 使用 JOIN 来代替子查询
  3. 使用联合(UNION)来代替手动创建的临时表
  4. 使用事务，事物以 BEGIN 关键字开始，COMMIT 关键字结束

- 权限策略

  1.前端记录所有的权限。用户登录后，后端返回用户角色，前端根据角色自行分配页面。优点是前端完全控制，想怎么改就怎么改；缺点是当角色越来越多时，可能会给前端路由编写上带来一定的麻烦。

  2.前端仅记录页面，后端记录权限。用户登陆后，后端返回用户权限列表，前端根据该列表生成可访问页面。优点是前端完全基于后端数据，后期几乎没有维护成本；缺点是为了降低维护成本，必须拥有菜单配置页面及权限分配页面

- 常见设计模式

  1. 单例模式
  2. 工厂模式
  3. 策略模式

  OOA/OOD 设计模式

- Django 项目：

  Render、redirect、httpresponse:

  render：只会返回页面内容，但是未发送第二次请求

  redirect：发挥了第二次请求，url 更新

  httpresponse:是字符串，用于 Ajax 前端动态刷新

  response.get 和 response.post:

  POST 和 GET 是 HTTP 协议定义的与服务器交互的方法

  GET 提交，请求的数据会附在 URL 之后

  POST 提交：把提交的数据放置在是 HTTP 包的包体中。

  GET 用来获取资源，它只是获取、查询数据，不会修改服务器的数据。

  POST 则是可以向服务器发送修改请求，进行数据的修改的，但会缓存。

  AJAX 最大的优点是在不重新加载整个页面的情况下，可以与服务器交换数据并更新部分网页内容。

- CAS 车站项目

  CAS 是以原子操作为基础，采用事务->提交->提交失败->重试这样特定编程手法的机制，它使得正在访问共享资源的线程不依赖于任何其它线程的调度和执行，并且能够在有限的步骤内完成。

  但会有 ABA 问题（AtomicStampedReference/AtomicMarkableReference 解决），储存版本号。

  利用 CAS 操作（Compare & Set）实现无锁队列_syzcch 的博客-CSDN 博客_c++ cas 指令

- Synchronized、Lock、CAS 区别

  synchronized: 少量同步

  Lock: 大量同步

  Lock 可以提高多个线程进行读操作的效率。（可以通过 readwritelock 实现读写分离）在资源竞争不是很激烈的情况下，Synchronized 的性能要优于 ReetrantLock，但是在资源竞争很激烈的情况下，Synchronized 的性能会下降几十倍，但是 ReetrantLock 的性能能维持常态；

  ReentrantLock 提供了多样化的同步，比如有时间限制的同步，可以被 Interrupt 的同步（synchronized 的同步是不能 Interrupt 的）等。在资源竞争不激烈的情形下，性能稍微比 synchronized 差点点。但是当同步非常激烈的时候，synchronized 的性能一下子能下降好几十倍。而 ReentrantLock 确还能维持常态。

  底层：

  synchronized: 底层使用指令码方式来控制锁的，映射成字节码指令就是增加来两个指令：monitorenter 和 monitorexit。当线程执行遇到 monitorenter 指令时会尝试获取内置锁，如果获取锁则锁计数器+1，如果没有获取锁则阻塞；当遇到 monitorexit 指令时锁计数器-1，如果计数器为 0 则释放锁。

  Lock: 底层是 CAS 乐观锁，依赖 AbstractQueuedSynchronizer 类，把所有的请求线程构成一个 CLH 队列。而对该队列的操作均通过 Lock-Free（CAS）操作。

  CAS：无锁，当冲突太多时性能不好，依据是“对于绝大部分锁，在整个同步周期内都是不存在竞争的”。

- Openmp 和 mpi

  OpenMP 并不适合需要复杂的线程间同步和互斥的场合。OpenMP 的另一个缺点是不能在非共享内存系统(如计算机集群)上使用。在这样的系统上，MPI 使用较多。

  MPI 使用进程间通信的方式协调并行计算，能协调多台主机间的并行计算。

  Openmp 使用共享内存的方式协调并行计算，用于单主机多核/多 CPU 并行。

  给一个 括号序列，求最长合法子串长度

  (36 条消息) 字节跳动飞书后端 offer 一举拿下，后端面试的基础天花板_超级实习生的博客-CSDN 博客

- 解决哈希表冲突：

  (36 条消息) 哈希表及处理冲突的方法_启福铭远的博客-CSDN 博客

- Union、struct、class

  默认继承权限和成员访问权限 Class 默认 private,struct 默认 public

  Class 可以用来定义模板参数

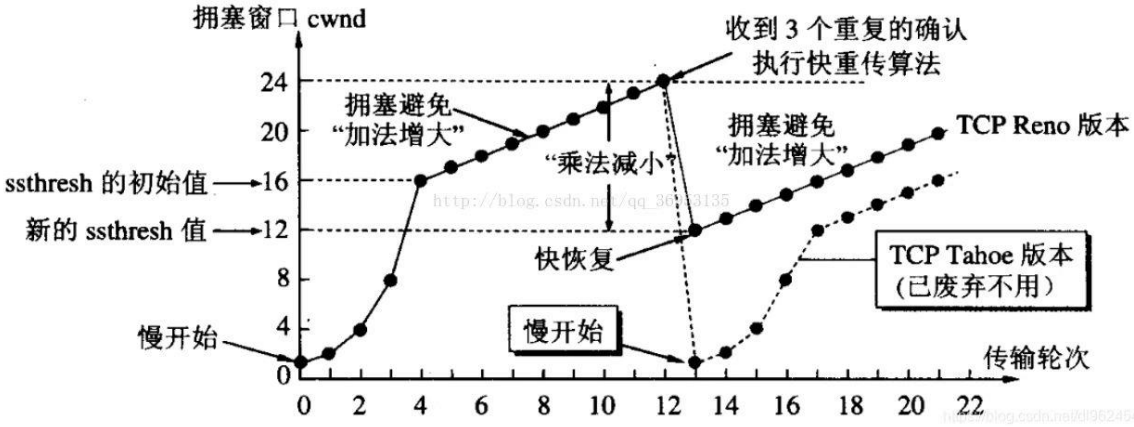
  Struct 和 Union 的详细区别 - jessonliu - 博客园 (cnblogs.com)

  C++ 中 struct/class 的内存布局 (huailiang.github.io)

- 左值：可以取地址并且有名字的东西就是左值。
  右值：不能取地址的没有名字的东西就是右值。
  左值：可以放到等号左边的东西叫左值。
  右值：不可以放到等号左边的东西就叫右值。
  移动语义：左值变右值从而省略拷贝构造函数的内存占用，变为移动构造。
  移动构造函数和移动赋值函数代替拷贝构造和拷贝复制。
- Memcpy
- Drop、delete、truncate
  Delete 可以用 where 语句
  Truncate 删除所有数据
  Drop 删除表结构
  Delete 可以用回滚找回数据
  Delete 是逐行找回数据，truncate 是删除原来的表再新建一个一样的表。
- 数据库分库分表
  垂直切分：垂直分库，根据表的耦合性分库，垂直分表，将内容过长的列分表，避免跨页、减少 I/O。
  水平切分:不能进行更细粒度的垂直切分，如行数过多，单库单写，最好要分库分表，因为单库分表还是会竞争同一个物理机的 I/O,CPU,内存。

- Epoll、I/O 多路复用
  I/O 多路复用：单线程或者单进程同时监测若干个文件描述符是否可以执行 I/O
- 同步异步、阻塞非阻塞
  请求方需要一直等待结果----阻塞
  请求方不用等待，会被通知------非阻塞
  请求方需要参与这段过程不能做其他事------同步
  请求方不需要参与这段过程---------异步
- Fork()
  子进程是不会执行前面父进程已经执行过的程序了得，因为 PCB 中记录了当前进程运行到哪里，而子进程又是完全拷贝过来的，所以 PCB 的程序计数器也是和父进程相同的，所以是从 fork()后面的程序继续执行。
  父进程中本来存在的变量的虚拟地址是相同，但因为进程间虚拟地址是独立的，所以物理地址不同
- 惊群问题
  多个进程被唤醒但只有一个进程去处理。
  用户态加锁，内核态新增参数/标志，保证 epoll 一个事件只有一个进程被唤醒，对于 select 和 poll，保证进程都绑定在一个套接字上，内核在收到新连接的时候，只会唤醒其中一个进行处理，还会进行负载均衡。
- 慢查询：
  看慢查询日志，找到满于多少秒的 sql 语句
  查看 explain,看索引是怎么设置的，看查询了多少行，索引设置是否有效。
  看有没有用 limit，用了性能也会差
  [如何查找 MySQL 中查询慢的 SQL 语句 - Agoly - 博客园 (cnblogs.com)]
- 红黑树的插入
- C++线程池

- 进程调度算法
  1. 先来先服务
  2. 短进程优先
  3. RR 时间片算法
  4. 最高响应比优先：响应比=1+（作业等待时间/作业处理时间）
  5. CFS 进程调度，根据任务权重计算运行时间，转换成 vruntime 然后存在红黑树里面。
- 虚函数类对象内存
  (36 条消息) C++中虚函数继承类的内存占用大小计算_Relly-Lee 的博客-CSDN 博客_类中虚函数占多少字节
- 虚函数表
  C++ 虚函数表剖析 - 知乎 (zhihu.com)
  构造子类对象时，先构造基类对象，vptr 指向基类的虚函数表，如果构造函数是虚函数，就要去找 vptr，那么此时虚函数表指针还没有被构造出来。
  析构函数是虚函数，如果不是，则基类指针只会静态绑定基类虚函数，从而在析构时只执行基类虚函数，如果是虚函数，那么基类指针指向派生类 vptr，从而执行派生类的析构函数，自动调用基类析构函数，这样整个派生类对象才能被完全释放。
- Unique_ptr 如何赋值，用 move，地址块的内存转移给另外的指针。
  C++ 11 创建和使用 unique_ptr - 滴水瓦 - 博客园 (cnblogs.com)
- 可变参函数模板、类模板
  递归的参数模板
- TCP 可靠性：
  (36 条消息) TCP 可靠性的保证机制总结_xuzhangze 的博客-CSDN 博客_tcp 保证可靠性的机制
  1. 校验和
  2. 序列号
     保证可靠性、按序到达、多次发送一次确认、去掉重复数据
  3. ACK 机制
     TCP 首部有一个 ACK，=1 时确认有效，确认字段值表示这之前的所有数据都已经按顺序到达，发送方收到后继续发送下一部分报文，如果没有收到则启动重传。
  4. 超时重传机制
     发送方发送的数据丢失，那么超过 RTO（略大于往返时间 RTT，RTT 也是不断计算更新的）没有收到 ACK 则重发；Linux 中 RTO 会以 500ms 的整数倍进行指数变化。
     接收方的 ACK 丢失，发送方仍然重传刚刚的数据，但接收方根据序列号判断重复后会丢掉并重发 ACK。
  5. 连接管理机制——三次握手四次挥手
  6. 流量控制
     TCP 头部有一个 16 位的窗口长度，有一个窗口扩大因子 M。接收方在收到发送方的数据后发送的 ACK 里面会在窗口长度存放缓冲区剩余大小，如果为 0，则发送方将停止发送，并不断发送窗口探测数据段，让接收方发送窗口大小给自己。
  7. 拥塞控制



快重传和快恢复：

(36 条消息) TCP 拥塞控制——快重传与快恢复_kongkongkkk 的博客-CSDN 博客

1. 接受方每收到一个失序的报文就立即发送重复确认
2. 发送方只要一连收到三个重复确认就立即重传对方没有收到的报文，不用等待重传计数器到时。
3. 有的快重传会把拥塞窗口设置为 ssthresh+3*mss,因为已经有三个重复 ACK 表示有三个报文在网络中发送而没有拥塞。

拥塞控制和滑动窗口（流量控制）结合：发送窗口的上限是接收方给出的接收窗口和自己的拥塞窗口中的较小值。

UDP 可以在应用层模仿 TCP 的可靠性传输。

- 为了实现进程间的隔离，为每个进程分配自己独立的地址空间，这就是虚拟内存，虚拟内存可以通过分页映射向物理内存映射，为了节省页的存储空间大小，可以采用多级页表

页面置换：当物理内存接近满时，需要将不常使用的页面置换到磁盘中去。

现在的 LINUX 使用的是两级的软件 LRU

- Innodb 和 myisam 的区别

InnoDB&MyISAM 区别 - 知乎 (zhihu.com)

1. InnoDB 的 B+树主键索引的叶子节点就是数据文件，普通索引叶子节点是主键的值；MyISAM 的 B+树主键索引和普通索引的叶子节点都是数据文件的地址指针；；
2. InnoDB 支持外键和事务，Myisam 不支持。
3. InnoDB 支持行级锁，MYisam 表级锁。

- SSL：在 HTTP 和 TCP 层之间，通过颁发证书实现加密。用户加密用公用密钥，解密用私人密钥。用户与 IIS 服务器建立连接后，服务器会把数字证书与公用密钥发送给用户，用户端生成会话密钥，并用公共密钥对会话密钥进行加密，然后传递给服务器，服务器端用私人密钥进行解密，这样，用户端和服务器端就建立了一条安全通道，只有 SSL 允许的用户才能与 IIS 服务器进行通信。
- SQL 注入：安全漏洞，在用户输入的字符串中添加 sql 语句。

解决:过滤输入内容、参数化查询

- CPU 满的排查：
- 数据库建索引：多个单列索引和联合索引的区别详解_深寒色的猫、的博客-CSDN 博客_复合索引和单索引的区别

联合索引优于多个单列索引，因为多个单列索引需要多个 B+树，开销大。

联合索引优先用第一个字段排序，如果没有第一个字段就无法使用联合索引。
不能用性别等重复率高的，首先一般不用这个当聚集索引，其次，因为是非聚集索引，会需要对筛选出的数据再进行回表，开销太大。

- 单例模式：

类只能有唯一的实例，且这个实例可以被全局访问（static 特性），且不允许用户自己创建实例（构造函数 private）。（一处写多处读）
是线程安全的，不能赋值和拷贝，用户通过 static 成员函数来访问。
工厂模式：C++ 工厂模式 分析和总结-阿里云开发者社区 (aliyun.com)
简单工厂模式：根据传入类的参数，动态决定创建哪一类实例（通过虚父类和子类来实现）。缺点是每次新增一种类就要改变工厂函数，违反开放封闭。
工厂方法模式：
每个产品类对应一个工厂类，每个工厂类实例化一个产品对象。
抽象工厂模式：每个工厂可以生产同一个产品的很多型号，产品类和工厂类都是虚基类。

- TCP 粘包问题：面试题：聊聊 TCP 的粘包、拆包以及解决方案 - 知乎 (zhihu.com)

  如果一次请求发送的数据量比较小，没达到缓冲区大小，TCP 则会将多个请求合并为同一个请求进行发送，这就形成了粘包问题。

  如果一次请求发送的数据量比较大，超过了缓冲区大小，TCP 就会将其拆分为多次发送，这就是拆包。

- 

-
- 时间复杂度问题
- 主从复制

手把手教你搭建 MySQL 主从复制经典架构（一主一从、主主、一主多从、多主一从） - 云+社区 - 腾讯云 (tencent.com)
建立一个和主数据库相同的从数据库，实现数据的多处自动备份，实现数据库的扩展。
优点：
1. 数据热备，主数据库服务器故障后，可以之间在从库上继续
2. 多库提高 I/O 性能
3. 读写分离，是数据库实现更大的并发。
实现原理：
主库 db 的更新事件被写入到主库的 bin-log；从库读取主库的 bin-log，并将 bin-

log 的内容写到到自己的 relad log；从库读取 relady log，解析成 sql 语句，将更新的内容写到的从库的 db。

- Coredump: gdb 查看 coredump 文件 - 简书 (jianshu.com)
  Coredump 后，gdb test core
  Gdb 调试正在运行的程序：使用 attach 进程号 gdb 调试当前运行的程序_YB_Promise 的博客-CSDN 博客_gdb 运行程序
- 进程通信方式的应用场景
  进程通信以及常见的应用场景 - 作业部落 Cmd Markdown 编辑阅读器 (zybuluo.com)
- 数据库的可重复读是如何实现的:undolog
- C++并发：
  多线程：thread t();t.join();
- Python 和 C++的区别：
  1. Python 是动态语言，在运行时确定数据类型并存储数据类型；
     C++是静态语言，在运行前编译时确定数据类型。
  2. C++是编译型，使用专门的编译器编译为该平台硬件的机器码并包装成可执行程序，运行时直接执行（效率高）；
     Python 是解释型，在运行时编译（可移植性好）；
- 函数调用的栈操作：
  函数调用过程中栈到底是怎么压入和弹出的？ - 知乎 (zhihu.com)
  ESP、EBP、EIP
- 覆盖索引：MySQL 的覆盖索引与回表 - 知乎 (zhihu.com)
- 死锁：循环等待、互斥、保持与请求、不剥夺
  N 线程访问 N 个资源避免死锁的方法：指定获取锁的顺序，并强制线程按照指定的顺序获取锁。所有的线程按照指定顺序加锁和释放锁。
- 查看内存泄漏：memwatch、mtrace 和 muntrace
- Linux 线程和进程的区别
- 时间复杂度和空间复杂度：

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

快排空间复杂度 O(logn)因为递归会消耗一部分空间，每次评平分数组的最好情况是这样。

Hash 和红黑树的时间复杂度

红黑树增删改查：O（logn）

红黑树构建一棵树 O(nlogn)

- Mysql 关联表
- 手写 String

  (36 条消息) c++手写 string 类_CrazyFox%的博客-CSDN 博客
- 手写 LRU
- 快排堆排
- Const 修饰一个值能用指针修改吗：

  const 变量通过指针修改 详解_「已注销」的博客-CSDN 博客
- 菱形继承

  冗余和二义性问题
- 搜索 topk 的 QQ 号

  (36 条消息) 怎么在海量数据中找出重复次数最多的一个__技术分享__的博客-CSDN 博客_海量数据中找出重复最多的一个

  1. 按照 hash%x 把所有数据放在 x 个小文件里面，用 hashmap 统计每个小文件中出现的词和频次，使用最小堆取出频次最大的 100 个，然后进行归并。
- Limit 分页查询：limit （pagenum-1）*pagesize,pagesize;

  优化：
- (37 条消息) 多个单列索引和联合索引的区别详解_深寒色的猫、的博客-CSDN 博客_复合索引和单索引的区别

  联合索引最左前缀：联合索引可以看作任何复合索引的左前缀的索引，是因为

底层 B+树是优先从左到右进行排序的。