
自我介绍：

面试官您好，我叫陈俊嘉，就读于中山大学计科专业，岗位意向是后台开发，过去的两个月里在今日头条的存储架构部门担任实习生一职，在这期间的主要工作是开发快速启动redis集群以及代理的工具，并且在工具开发完成之后，利用这个工具复现线上的1个Bug，那自己在工作之余也会深入了解redis，例如其丰富的数据类型以及持久化、复制功能机制。对于后台开发这个岗位，我比较熟悉网络编程这一模块，例如一些IO复用组件、TCP、UDP套接字编程，我相信自己能够胜任这个岗位，谢谢。

不要急，先想清楚面试官想问什么，再回答！

微信问题准备:

- 昨天与1面面试官聊到了微信支付打黑的这个项目，感觉有点类似于我看到的一个支付宝宣传片，大家无法转账大量的金额到骗子的手里，想具体问问是怎么入手去做这一件事的？
- 针对不同的群体（例如骗子是一类群体，赌博庄家是一类群体），咱们这个维度的选择是否不同，比如说骗子、庄家都可以通过电话短信来往、可以通过资金来往，但是从人这个维度的话（如果没有先验条件的存在，从人的角度我感觉会比较难，像1面面试官说的从同个人但是换了另一个商家其实是已经有先验条件的存在了，那如果一个人突然改变自身习惯开始过上庄家赌博生活，此时如果从人的维度考虑，那又是如何的呢？），听起来的确是比较有意思的，不知道您能不能具体讲一下。
    - 实则是一个基于社交网络的用户分类问题，黑用户与普通用户的二分类问题。
    - 资金来往
    - 电话短信来往
    - 用户好友数，好友关联网络（相关系数）
    - 用户之间交流（文字、图片）、例如资金交流的钱数，一些例如频繁出现转账、转钱这方面
    - 人？？
- 想具体问一下精准打击这个概念，因为觉得这个概念真的比较重要。
- 想问一下如果在您这个团队担任实习生一职，您认为这个实习生应该具备什么样的素质以及技术栈。
- 对于我来说的话，您觉得我还需要加强哪些方面，或者说补充哪些方面的知识点。（可以不问）

今日头条实习项目中最大的挑战\困难（难回答）

- 讲配置文件的问题吧，最后结论是当遇到困难的时候，最好是和有经验，例如导师或者是同事讨论，再查找一些资料进行借鉴，定好方案再动手。

科研项目中最大的挑战\困难

- 讲五天写一篇论文出来的过程，过程包括了着手建立一个较为完整的模型，对比过往的方法，在某些地方取得了较好的性能提升，某些方便确实存在不足，但是时间非常紧，只能立马把方案固定下来，然后做实验、画图、写论文，一气呵成。有缓有急的科研工作是非常正常的，在非常急切地需要赶进度的时候，需要更加关注时间截点的时候，难免对整体模型会有所疏忽，但是当交稿后，可以更进一步研究，与人合作是非常关键的一个步骤，大家齐心协力完成，效率更高，更快。

腾讯面经

- **new和malloc的区别，placement new, operator new？**
    - new是一种操作符(对象内默认的操作符)，malloc是一个函数

    - new申请内存返回的类型是对象类型的指针（与对象匹配），而malloc只是单纯地被通知去申请多大的内存块它的返回是无类型指针、需要强制类型转换
    - new和delete（操作符可重载）会调用对应的构造函数以及析构函数完成对象的创建与析构，malloc和free根据指针申请与释放内存块
    - new具体步骤：
        - 调用operator new分配足够的内存空间
        - 调用对象的构造函数初始化
        - 构造完毕，返回该对象的类型指针
    - malloc具体步骤：申请给定参数的内存块

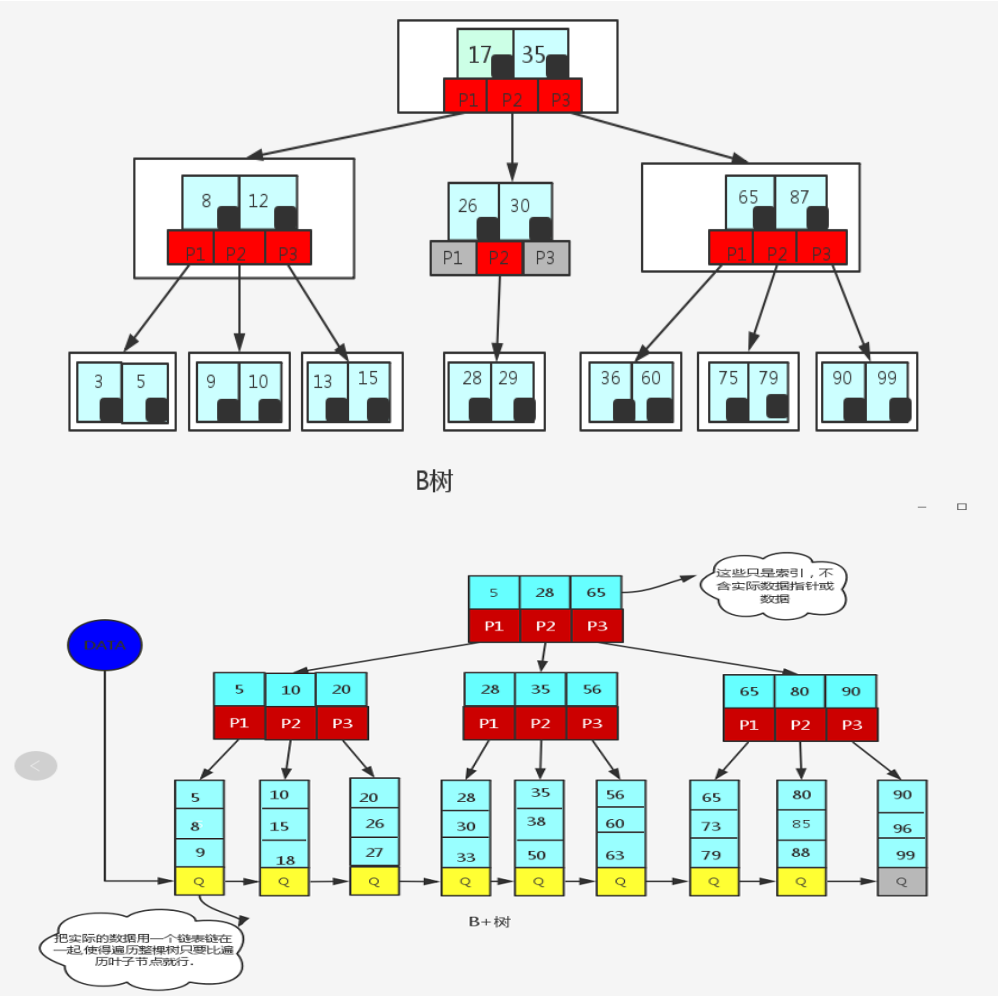
- **c++中class和struct的主要区别？**

    - 默认继承权限：class是private，struct是public
    - 默认成员函数访问权限：class是private，struct是public

- 四次挥手过程，为什么不能直接三次挥手？类似于三次握手过程？

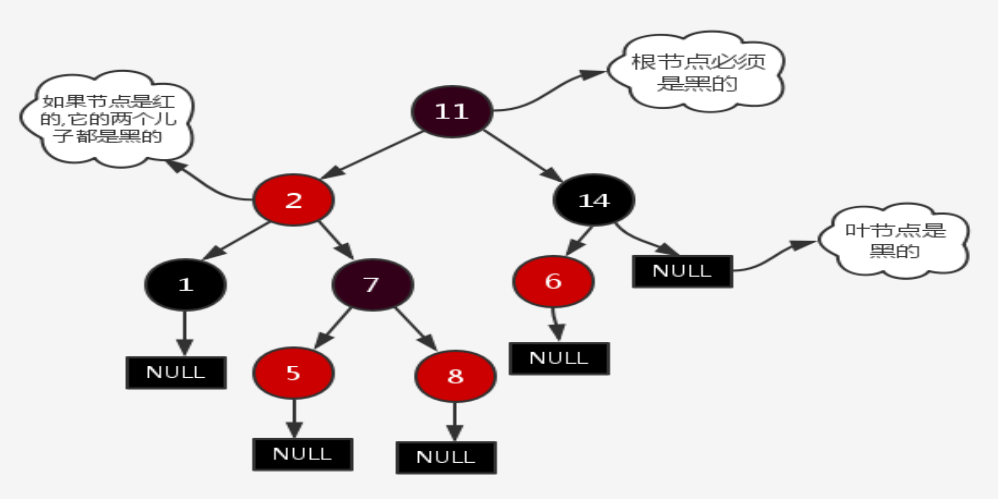
    - 在主动方接收到被动方的ACK时只是告诉对方，我们不发数据了，但是还可以接收数据，在接收ack与发送fin之间可能还会接收到数据，因此不能够直接选择发送 FIN。

- 红黑树、B+树、B树的区别



    - B+树与B树区别：
        - B+树是B树的升级版，B树的每个节点都保存指向数据的指针，而B+树只会在叶子节点上保存指向数据的指针，因此B+树叶子节点是包含了所有的索引的，而B树在每一个节点都有索引及指向数据的指针。

        - B+树对磁盘友好：
            - 内部节点不含指向数据的指针，节省了一定的磁盘占用，使得内部索引尽可能在一个块内存储。
            - 查询效率稳定，所有索引最终指向叶子节点，查询必须经过根到叶子的路径。
            - B+树的遍历可以利用叶子节点间链表串联的方式遍历，而B树就必须利用中序遍历了（往返顺序查找等）。



    - 红黑树性质：
        - 节点是黑色或者红色
        - 根节点是黑色的
        - 所有叶子节点是黑色的（叶子节点是null节点）
        - 从根节点到叶子节点的路径上黑色节点数量必须是相同的
        - 若父节点是红色的，则其两个子节点必须是黑色的
    - 红黑树弱平衡保证了插入删除查询最坏操作都是O(logN):
        - 因为一条路径上（根到叶子）的黑色节点数量必须相同，那么最短的路径则有可能是全部为黑色，最长的则可能是红色连续交替，这些性质保证了其最长路径不会超过最短路径的2倍，保证了logn的分叉。

- **cout和printf区别**

    - cout定义在iostream中的一种对象类型，printf是定义在stdio.h头文件中的一种函数
    - cout更安全，内含大量的运算符重载，可以输出各种基本数据类型，printf需要严格定义的数据类型才可以对应输出

- **sizeof和strlen的区别**

    - sizeof类似于new一样也是操作符，strlen是函数
    - sizeof的操作对象可以是变量、函数、表达式、数据类型、对象等等，而strlen的传入参数是char*指针，它主要计算字符串长度，遇到结束标识符时返回其长度。
    - sizeof编译已经确定好要返回的大小，strlen是在运行时计算字符串长度。

- **在浏览器地址栏输入一个URL后回车，背后会进行哪些技术步骤？**

    - 浏览器先获取到url （Uniform Resource Locator），统一资源定位符：协议（http、https）：//域名（或是IP地址）/路径/文件。
    - 若是获取到的是域名而不是Ip地址（域名越后越高级），则需要查询对应的IP地址，具体的查询过程：
        - 本地hosts文件查询

        - 本地DNS服务器查询（查询本地的DNS服务器是用UDP协议，快，丢失重新请求即可）
        - 向更高级的DNS服务器查询（服务器之间的查询利用的是TCP协议，保证可靠性，请求了一次不需要再请求，减少树顶的服务器压力）
    - 这时候浏览器已经将域名解析成了具体的IP地址了，那么可以由顶往下层层传递，分别根据应用层的HTTP协议，传输层TCP协议，网络层IP、ARP协议等组织请求报文，通过数据链路以及物理层传递到相对应的服务器上，服务器响应发回相对应的报文，浏览器得到数据（通常是HTML）渲染到屏幕上展示给用户。

- 进程间通信方式：

    - 无名管道：有血缘关系的进程中使用，通常是父子关系。
    - 有名管道：任意进程中都可以用有名管道通信。
    - 消息队列：消息队列的话可以被认为是一个消息链表。某个进程放置消息，其余进程可以根据需求去读取消息。管道得先有读取者，写入才有意义，消息队列不需要这样。
    - 信号量、互斥量、读写锁等同步手段：应该是与线程类似的锁机制。
    - 共享内存（回答的时候可以深入这一块讲）：是IPC（interprocess communication, 进程间通信）形式中最快的，不需要执行内核的系统调用来传递数据。假设在文件复制过程中，利用管道、FIFO、消息列队进行进程间通信时，我们总共是需要4个系统调用，包含了文件到某个进程、进程到管道、FIFO、消息列队，管道、FIFO、消息列队到另外一个进程，另外一个进程到文件，四次数据复制过程，比较耗时，共享内存区只需在进程中控制文件读取到共享内存区，再由另外一个进程将共享内存区的数据复制到输出文件即可，总共2次系统调用。
    - 套接字：TCP与UDP这些。

- 线程间通信方式：

    - 线程间通信主要是为了线程同步，没有数据交换的通信机制。线程并发执行，能够共享进程中共享内存区，例如堆数据、全局变量等
    - 线程安全机制：
        - 锁机制。线程访问资源前必须获得其锁，若是锁已经被占有则需要等待，线程访问结束后释放锁。
        - 信号量。锁机制一种，分二元与多元信号量，多元信号量n，每个线程访问时将信号量-1，当n等于0时则等待，二元信号量则是n=1;
        - 互斥量。比信号量更严格，某个线程获取信号量后必须由它自身释放，其余线程释放无效。
        - 临界区。比互斥量更严格，临界区内的资源只允许本进程去试图访问，其余进程不允许。
        - 读写锁。是为了在读的情况下允许高并发访问，写的情况下执行锁机制，锁状态包含自由、共享、独占，要想独占得等到锁自由才可以。

- 死锁（互相等待对方的资源）的四个必要条件

    - 互斥：一个资源每次只能被一个进程所持有
    - 占有且等待：进程因请求资源阻塞时不会释放自己已持有资源
    - 不可强行占有：不可强行剥夺进程占有资源，除非进程已停止释放了资源
    - 循环等待：若干多个进程形成头尾相接的等待资源关系
        - 银行家算法能够有效地避免死锁：
            - 银行家算法是操作系统通过维护4个表，判断能够给予进程资源，让进程运行的依据。
            - allocation：进程已持有资源
            - max：进程需要的最大资源

        - available：目前操作系统所剩余资源（可被获得资源）
        - need：max-allocation，进程还需要多少资源才能继续运行
        - 实际上就是维护两个表need, available，need小于available则赋予其运行权利。

- 进程、线程的状态

    - 线程：就绪（可以立马运行，但没有CPU时间片）、等待（等待资源释放）、运行
    - 进程：
        - R（可执行状态）
        - S（可中断睡眠状态，等待唤醒）
        - D（不可中断睡眠状态），例如执行vfork系统调用后，父进程将进入TASK_UNINTERRUPTIBLE状态，直到子进程调用exit或exec
        - T（暂停或者跟踪状态）
        - Z（僵尸进程、僵死状态，留下空壳（包含退出信息）给父进程查阅
        - X（退出状态、进程即将销毁）

- 数据库事务的属性（**ACID**）

    - Atomicity原子性：要么执行要么不执行
    - Consistent一致性：数据库状态转换必须保持一致。300+300->400+200
    - Isolation隔离性：事务之间不会相互影响。
    - Durability持久性：事务提交成功，则永久保存在数据库内。

- 如何为**UDP**增加可靠性

    - 请求-应答式应用程序使用UDP的话，那么必须在客户程序中增加以下两个特性：
        - 序列号：客户能验证服务端的应答是否匹配自己发出的相应请求。
        - 超时和重传：用于处理丢失的数据。
            - 因为RTT(Round-Trip Time，同个数据报来回总时间)是随着网络条件的变化而变化的，我们的超时和重传算法必须是采用一个实测的RTT作为基准去设定的。Jacobson算法能够给出了这个算法的细节。
            - Jacobson无法解决重传二义性问题：
                - 请求丢失
                - 应答丢失
                - RTO(retransmission timeout, 重传超时)太小
            - 如何解决重传二义性？我们可以对每个请求在加入序列号的同时，加入当前的客户端的时间戳，服务器只需要在做出应答的同时将客户端的时间戳也同样加入到应答中去，客户就能够有效判断应答时对应于哪个请求。

- **TCP**与**UDP**的区别

    - TCP是面向连接的字节流传输协议，UDP是无连接的简单传输协议。
    - TCP是通过对端确认、重传机制、滑动窗口等确保通信是可靠地，单纯的UDP无法保证通信是可靠的。
    - TCP保证了数据可靠传输，即使它无法保证数据报是顺序到达的，但是接收方能够对数据报进行排序重组。UDP不保证数据报会到达对端，也不保证顺序到达。
    - TCP一般是一对一。UDP一般可以一对一，一对多，多对多。
    - TCP一般不会有记录边界的值，而UDP每个数据报都有一个长度值。

- **static** 静态函数、静态全局变量、静态局部变量、静态成员变量、静态成员函数

    - 当static修饰一个函数时（**静态函数**，面向过程）
        - 函数只可以被本文件的其余函数调用，不可以跨文件调用
        - 其他文件有相同函数名时，不会发生冲突
    - 当static修饰一个全局变量时（**静态全部变量**，面向过程）
        - 静态全局变量只对本文件上有效，其余文件即使用了extern也无法引用静态全局变量
        - 与其余文件有着相同变量的全局变量不会发生冲突
    - 当static修饰一个局部变量时（**静态局部变量**，面向过程）
        - 静态局部变量与静态全局变量是存储在进程的数据区(已初始化的变量存储在.data段、未初始化的变量存储在.bss段)
        - 静态局部变量只在作用域内有效，出了作用域是无效的，但其始终驻留在数据区直到程序结束
        - 初始化只在第一次有效，其余无效
    - 当static修饰一个成员变量时（**静态成员变量**，面向对象）
        - 这个变量是所有对象所共有的。普通变量每个对象实例都有自己的一份拷贝，而静态成员变量不是的，所有对象实例共享一份。
        - 静态成员变量同样存储在数据区，内存在初始化时才分配，静态成员变量必须在初始化，而且只能在类体外进行初始化。
        - 静态成员变量不需要通过某个实例访问，直接访问这个类::变量即可
    - 当static修饰一个成员函数时（**静态成员函数**，面向对象）
        - 静态成员函数属于类的本身，不属于任何一个实例对象，不具备this指针。**因此不能够访问非静态成员函数与非静态变量，只能访问静态成员函数与静态变量**
        - 静态成员函数与静态成员变量一样可以通过类::函数，类::变量直接访问。

- 谈一谈**volatile**关键字

    - 阻止编译器为了提高速度将一个变量缓存到寄存器内而不写回，读取某个值到寄存器操作后必须写回变量里
    - 阻止编译器调整操作volatile变量的指令顺序，即使能够阻止编译器优化，也无法阻止CPU动态调度
        - 这里可以延伸去讲singleton（如何创建单例模式的过程）
            - new的第二和第三个过程因为CPU的动态调度被交换：
                - 分配内存
                - 调用构造函数
                - 返回内存地址地址给用户

```
#define barrier() ___asm___ volatile ("lwsync")
volatile T* pInst = 0;
T* GetInstance(){
        if (pInst == NULL) {
                lock();
                if (pInst == NULL) {
                        T* temp = new T;
                        barrier(); // 建立水坝，cpu动态调度无法穿越这个水坝进行乱序执行
                        pInst = temp;
                }
                unlock();
        }
```

```
        return pInst;
}
```