**面试掌握在自己手上，尽量说自己知道的。不会的就说不会，不要猜。先思考，拖时间，不要秒答，想好再说，并且问清楚条件。**

Parker（分布式持久化KV存储系统）  
接入数据量：接入25个集群，总数据量：64.9 TB，占全年接入量的90%左右。  
SLA: 99.95%  
测试覆盖率: 单测覆盖率84%，自动化测试用例140个  
产品功能：Parker主要用于持久化的KV数据，一般数据量在TB级以上。主要用于代替类似场景中的Redis集群，大大节省机器成本，Parker只需要原来10%的机器规模来存储相同数据，同时整体性能只是略低于Redis。  
业务满意度：Parker在智能推荐、内容产品中心、Push等多个业务中得到应用。特别是在智能推荐-用户画像业务中替换了原有的Redis集群，同时读写延迟等指标达到业务要求，得到了业务团队认可。

1. 算法：维护第二大的值

遍历一遍，n，保留max和第二大的max

1. Select 优先级问题
2. 算法题：从后向前每k个节点做一个反转，若剩余不足k个，则不用反转(27)

先 k反转的实现，图模拟，然后再写整体的

1. 算法题：判断树是否是对称的

层次遍历 l.r,r.l

1. 栈实现O(1)获得最小的数getMin()

辅助栈、小的入

1. 智力题 有1000个一模一样的瓶子，其中有999瓶是普通的水，有1瓶是毒药。任何喝下毒药的生物都会在一星期之后死亡。现在，你只有10只小白鼠和一星期的时间，如何检验出那个瓶子里有毒药？

二进制，2\*\*10 = 1024 1死2死3死则是7号(111)有毒

1. 确认链表是否有环

双指针，快慢；

1. 口述快排，什么时候是n2（每次有一部分是空的）什么时候是nlogn（pivot对半开）
2. 二分查找
3. Rand3实现rand5,rand5实现rand3

Def rand3():大的实现小的 def rand5(): 小的实现大的

X = -1 x = sys.maxsize

While not 0<=x<3: while x>5:

X = rand5() 3(rand3 - 1) + rand3

Return x return x

11. 5个海盗100个金币，1号分配不死

45：0 100

345：100 0 0 (or 99 1 0)

2345：98 0 1 1(or 97 0 2 1)

1 2 3 4 5: 97 0 1( 98 0 1 0 1 or 97 0 1 0 2)

1. 硬中断和软中断(硬中断是外设引发的，软中断是执行中断指令产生的，用来处理硬中断中时间较长的工作)
2. 操作系统中Kernel Mode和User Mode的区别（CPU处于KM，任务可以执行特权级指令，对I/O设备有全部的访问权，能访问任何虚拟地址和控制虚拟内存硬件。CPU处于UM，硬件防止特权指令的执行，对内存和I/0空间的访问操作进行检查。如运行的代码不能通过os的某种门机制，就不能进入内核模式）
3. 海量数据，前K（分而治之/映射+hash统计+堆排序/快速/归并）；海量数据，不重复（2bitmap，00、01、10、11）or重复的数据(bitmap)；找中位数(分治：遍历，hash划分，统计个数直到k个；去到该机器中，排序找到n\*\*2/2-x)
4. Go语言的特点：1.底层使用goroutine作为并发实体，每个实体之间通过channel通讯来实现数据共享，将并发单元间的数据耦合拆解开。2.goroutine底层是使用协程(运行在用户态的用户线程)实现并发，用户空间、避免了内核态和用户态的相互切换，更小的栈空间能创建大量的实例3. 充分利用了多核的硬件资源，近似的把若干goroutine均分在物理线程上4.标准库完善5.代码简洁6.性能强劲的同时，开发效率又不差于Python等动态语言
5. TCP和UDP的区别(1.TCP面向连接udp无连接2.tcp提供可靠服务，udp不保证可靠3.tcp面向字节流，udp面向报文4.tcp有拥塞控制，udp无，UDP的性能收到网络层的影响很大，当网络层的状况比较好的时候，UDP的性能远远高于TCP，但是当网络层的情况不好时，UDP反而会使网络状况越来越差5.tcp只能点到点，udp支持一对一、一对多、多对一、多对多的交互通信6.tcp首部开销大20字节，udp8字节)
6. TCP三次握手和四次挥手过程(

<https://blog.csdn.net/qq_38950316/article/details/81087809>

1. 第一次握手，客户端发生SYN包(seq = x)到服务器，进入syn\_sent状态，等待服务器确认
2. 第二次握手，服务器收到syn包，必须确认客户的SYN(ack=x+1)，同时自己也发送SYN包，即SYN+ACK包，此时服务器进入SYN\_RECV状态
3. 第三次握手，客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=y+1)，此包发送完成，客户端和服务器进入established状态，完成三次握手

四次

1. 第一次挥手，客户端发送连接释放报文，停止数据发送，释放数据报文首部，FIN =1, seq = u（==前面已经传送过来的数据的最后一个字节的序号+1)，客户端进入，FIN-WAIT-1
2. 第二次挥手，服务器收到连接释放报文，发出确认报文，ACK = 1，ack=u+1，seq=v，服务器进入close-wait状态。tcp服务器通知高层应用，客户端向服务器端的方向释放，处于半关闭的状态。
3. 客户端在收到服务器端的确认请求后，客户端进入FIN-WAIT-2状态，等待服务器发送释放报文(FIN)
4. 第三次挥手，服务器将最后的数据发送完毕后，就向客户端发送连接释放报文，FIN=1，ACK=1, ack = u + 1，seq = w，服务器进入last-ack
5. 第四次挥手，客户端收到服务器的连接释放报文，必须发出确认，ACK = 1, ack = w+1，seq = u+1，客户端进入了TIME-WAIT状态，经过2msl后，客户端撤销响应的TCB后，进入closed状态
6. Server收到确认，立即进入closed状态。

)

1. http 1.0和1.1的区别（1.0 默认短，支持长，需要设置header connection: kepp\_Alive 1.1默认长）
2. 四次挥手的状态

Closed；收到ack；close-wait；server数据发送完，发送FIN；last-ack；收到ACK；closed

1. 为什么连接的时候是三次握手，关闭的时候却是四次握手？

当server收到client的SYN连接请求时，可以直接发送SYN+ACK报文。但是关闭的时候，server收到FIN，很可能不会立即关闭socket，所以只能先回复一个ACK报文。只有server端所有报文发送完了，才发送FIN报文。

21. 为什么不能用两次握手进行连接？

三次握手作用：1、双方彼此知道都已经准备好了2、就初始序列号协商。若两次，服务器发送的确认有可能丢失，服务器端认为已经成功，client将不知道S是否准备号，S的序列号是什么。且会死锁，client一直等待ACK。Server也超时重传一样的分组。

1. 如果已经建立了连接，但是客户端突然出现故障了怎么办？

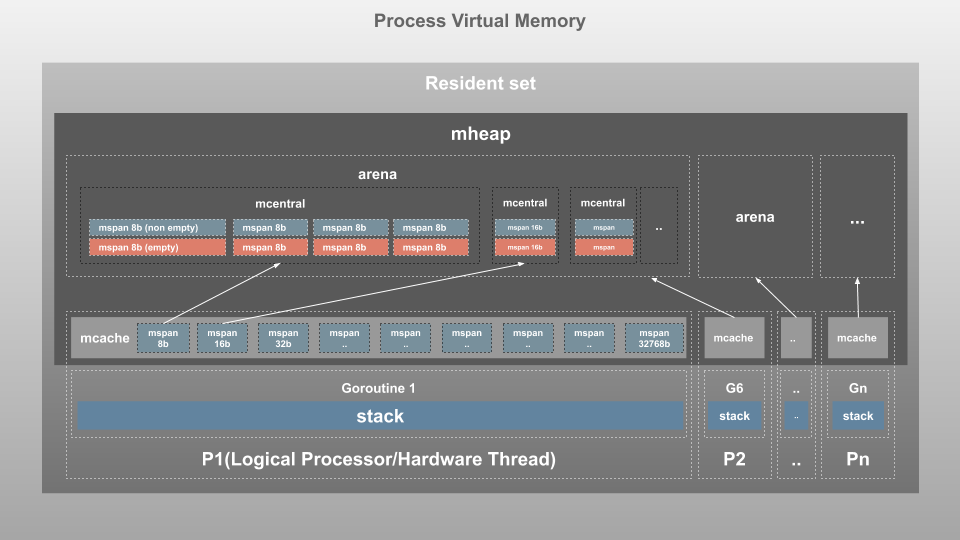
TCP有一个保活计数器，每次收到client都会复位，2h，2h没收到，发送探测报文段，每隔75s发送一次，一连发10个无反应，断开。

23. time\_wait大量出现的原因（socket忙于读写或者程序忘了没有close、爬虫服务器）？如何避免这种问题？（两种解决方案：基于socket调用close；基于TCP连接的参数设置，连接超时设置或者按照一定的频率监控连接是否活动，没有活动就关闭连接。1、让服务器能够快速回收和重用那些TIME\_WAIT的资源，内核参数优化/etc/sysctl.conf）

24. Go的内存分配器、垃圾回收机制、栈内存管理

Go静态语言，不需要vm。通过go的二进制文件中嵌入了go runtime，实现gc、调度、并发。内存空间由栈区、堆区组成，函参、返回值、局部变量分配到栈上，这部分由编译器管理。go的内存管理=需要内存时自动分配内存，不需要gc。这些工作由runtime完成。

Runtime将goroutines(G)调度到P（逻辑处理器），每台P有M（逻辑机器）。



残余内存为os分配给进程的虚拟内存，Mheap为最大的内存块，进行gc的地方，存动态数据(编译时无法计算出大小的data)

Mysql

聚集索引：逻辑和物理结构相同

非聚集：逻辑和物理结构不相同

Innodb engine 还行级锁（通过索引实现），支持事务，支持外键(CONSTRAINT `xiaodi\_ibfk\_1` FOREIGN KEY (`dage\_id`) REFERENCES `dage` (`id`),保证数据一致性，和一些级联操作)、（聚集索引、主键索引的叶子节点就是数据文件，辅助索引的叶子节点是主键）

Myisam非聚集索引

四种事务隔离级别：1. 读未提交(事务未提交时，另一个事务可) 2. 读已提交(只能读已提交数据，解决脏读)3. 可重复读(一个事务内多次读取，数据都是一致的。解决了脏读、不可重复读)4. 序列化(事务串行执行，解决脏读、不可重复读、幻读)

不良反应：脏读、不可重复读(事务所读取的数据值变了)、幻读(类似前者，但是是再次读取时插入数量变了)、丢失修改(两事务同时对数据库做相同的修改，读的值相同，操作相同，前面的事务的修改被覆盖)

Post 不幂等 升级用这个、有新的事务产生

Put 幂等 停止用这个

MongoDB是B树（n叉，多节点）

红黑树：自动平衡

180个接口

实例不可比较，实例是指针

类型的实例，实例值

Var a int

typeofA := reflect.TypeOf(a)

aInstance := reflect.New(typeofA)

Goroutine本质是协程，协程之间的调度由程序底层完成(线程和进程的调度是sys)，协程有自己的寄存器上下文和栈，切换时将两者保存到。

Go defer栈先进后出

Select底下有多个可执行的case，则随机执行一个。

无缓冲chanel：传值立马close

Select可以通过default实习优先级队列

Cpu - cache(sram) - 内存(dram) - 硬盘(ssd)

腾讯面试（**CSIG算法比较简单，主要还是八股和实习、项目**）

1. Linux指令，怎么查看内存线程使用情况

top -Hp pid

1. Mysql union和union all

Union 把两个查询结果去重合并 all则不去重（第一个 SELECT 语句中被使用的字段名称也被用于结果的字段名称、各 SELECT 语句字段名称可以不同，但字段属性必须一致）

1. Mysql groupby

GROUP BY 语句根据一个或多个列对结果集进行分组

1. 四次挥手，为什么等待2MSL(time-wait)?

4.1 为了保证 客户端最后一个ACK能送到服务器。ACK可能丢失，当处在last-ack的服务器没有收到对已发送的FIN+ACK报文段的确认，就会超时重传，A就能在2msl内收到这个重传的FIN+ACK的报文段。如果客户端在time-wait不等待，发完ack就立即释放，就收不到重传的fin-ack，也不会再一次发送ack。服务器就无法按正常步骤进入closed状态

4.2 客户端发完最后一次ack，等2msl，可以让本连接持续时间产生的所有报文段从网络中消失。下一个新连接中不会出现旧的连接请求报文段，迷路后重新出现。