1. 自我介绍，1，2，3段
   面试官你好，我是伍振仁，我本科就读于华北科技学院采矿工程，15年毕业后进入一家有色金属家乡国企工作，由于不太喜欢那边的工作节奏和对计算机的浓厚兴趣，在2018年辞职，2019年参加考研录取为中国科学技术大学的软件工程研究生，目前研二在读。
   进入学校后结实了很多志同道合的同学朋友，学习了很多计算机基础课程，比如数据库、算法数据结构、操作系统，也结识了很多志同道合的同学朋友，和研究生同学一起做了一个主打搜索功能的金融理财App的Springboot项目，学习了很多开发的技能，同时也加深了对计算机基础的理解，去年十一月作为后台开发实习生进入腾讯，主要负责数据链路构建和维护，以及旧链路到新链路到迁移。**本人喜欢有技术氛围的团队，也很喜欢计算机的这一类的工作。**现在想找一份暑期实习的后台开发岗位，谢谢面试官。
2. 请你介绍一下项目，以及实习负责的内容
   金融理财搜索App主要负责后台的后台管理系统，例如索引配置、搜索框和热搜推荐、前台的搜索功能

   实习在一个搜索部门，组里是做知识图谱应用，本人负责的主要是应用数据构建，主要是工作上维护链路、线上问题追查和需求的开发，以及旧链路到新链路的迁移。
3. redis了解持久化、原理、集群
4. mysql索引、crud操作数据的本质、sql调优、分库分表（用户 商品 订单例子）、日志、事务、乐观锁悲观锁
   1. 事务，开始begin，结束，commit。单条sql也是一个事务。。
      四个隔离级别：脏读，读未提交，就像腾讯文档，一起修改。不可重复读，读已提交，一个事务可能会改多次，另一个事务多次读取得到的值前后不一致。幻读，可重复读，对于已有的值可以保证一致性，但是新写的值则没有办法控制，比如ID 1-10，写入了一个，select出来发现数据不一致。这个问题可以用mvcc解决。串行化，最大程度保证一致性，但是这个不支持并发。
   2. MVCC：
      1. 当前读和快照读：**当前读**，基于悲观锁，
         就是当前读取的数据一定是最新版本的，加了读锁 select for update,共享锁，其他事务只可以读，不可以写。排他锁，其他事务不能读和写。**悲观锁**包括共享锁和悲观锁。JDK是synchronized，
         乐观锁适合读多写少的场景，比如CAS，版本链控制，CAS的ABA问题，因为CAS基于值的比较来判断数据是否正确，但是如果这个值从1改成2，2再改成1，那也可以认为正确，不排除会引发其他问题。
         MVCC就是基于乐观锁，使用版本链来控制数据的正确一致性。Java的就是原子类，
      2. 快照读，基于快照的版本来实现版本控制。
   3. 二阶段提交，分布式数据库，prepare阶段，参与者本地写redolog和unlog日志，把commit请求发给协调者，commit阶段，协调者根据投票情况把ack发给参与者。
      三阶段提交，多了一个pre commit阶段，引入了超时机制，在某个参与者超时之后会发个信息给他，他可以再次发送请求。
   4. Innodb和Mysiam
      1. Innodb支持事务，MySiam不支持事务。

2. 锁粒度，Innodb行锁，Mysiam支持表锁。锁粒度太细加锁释放锁的代价太高
3. Innodb是聚簇索引，以id主键作为逻辑地址连续存储在磁盘中，Mysiam不是聚簇索引。聚簇索引就是非主键索引的叶子是主键，再通过主键索引，得到的叶子结点才是数据。非聚簇索引直接叶子结点就是数据。
4. 都支持b+索引。索引存在磁盘中，使用的时候调入内存。
5. Innodb获取行数select count（因为事务，行更新太频繁，代价太高），Myisam是有这类元数据，保存了行数之类的元信息
6. Innodb必须要有主键，聚簇索引。myisam则不需要。

5. SQL优化：
    1. 表中建索引，where和group by经常用到的字段，覆盖索引
    2. 尽量避免使用select *，where中有索引中不存在的字段的时候，索引就会失效
    3. 使用短索引

       对串列进行索引，如果可能应该指定一个前缀长度。例如，如果有一个CHAR(255)的列，如果在前10 个或20 个字符内，多数值是惟一的，那么就不要对整个列进行索引
    4. 索引失效的原因：
        1. Like模糊查询不要用左通配符，因为B+树索引的机制
        2. 大于 小于或者不等于的情况
        3. 对索引进行加减运算也会失效，运算或者函数处理过的数据都会失效
        4. 复合索引要(a,b,c) 只where了c，没有前导a、b
    5. Explain两个很重要的参数：
        1. type，有All、range、index、即扫描方式，system（内存中） > const > eq_ref > ref > range > index > ALL
        2. Extra，NULL、Using index、Using where、Using index condition（命中索引、Using filesort（文件排序、Using temporary（临时内存
    6. **什么时候不要使用索引?**
        1. 经常增删改的列不要建立索引
        2. 有大量重复的列不建立索引；（不好找）
        3. 表记录太少不要建立索引。直接读到内存中就可以。
    7. 1. 索引需要占用***磁盘空间***，因此在创建索引时要考虑到磁盘空间是否足够
        2. 创建索引时需要**对表加锁**，因此实际操作中需要在业务空闲期间进行

5. Linux，至少准备15个指令，常用的不常用但实用的
   ll 、 grep、 ps -ef、tail、ls、diff -r 、man、find、whoami、sudo、which、echo、source
6. elastic search文档型数据库的索引存储形式、数据类型keyword、text，分词，以及日志场景
    1. 其实 es 第一是准实时的，数据写入 1 秒后可以搜索到。有 5 秒的数据，停留在 buffer、translog os cache、segment file os cache 中，而不在磁盘上，此时如果宕机，会导致 5 秒的数据丢失。
        1. 数据先写入内存 buffer，然后每隔 1s，将数据 refresh 到 os cache，到了 os cache 数据就能被搜索到（所以我们才说 es 从写入到能被搜索到，中间有 1s 的延迟）
        2. 每隔 5s，将数据写入 translog 文件（这样如果机器宕机，内存数据全没，最多会有 5s 的数据丢失），translog 大到一定程度，或者默认每隔 30mins，会触发 commit 操作，将缓冲区的数据都 flush 到 segment file 磁盘文件中。

3. 数据写入 segment file 之后，同时就建立好了倒排索引
2.

7. 数据结构，冒泡、快排、
8. 操作系统，进程线程、异步、内存管理、IO中断
9. 计算机网络，三次握手四次挥手、七层五层模型、TCP层的拥塞控制
10. 计算机系统，存储器、流水线
11. 消息队列，说一说Kafka
12. Java，JVM、GC、锁升级、HashMap、集合类家族，并发编程、线程池，泛型编程
    1. 集合类，
        1. Map的遍历方式
            1. for(Map.Entry<k,v> entry: map.entrySet()){}
            2. for(K key: map.keySet())
            3. Iterator<Map.Entry<k,v>> iterator = map.entrySet().iterator();
            4. map.forEach((k, v) -> System.out.println("key: " + k + " value:" + v)); lambda表达式。
        2. 优先级队列和比较器，Comparable，Comparator
            1. Queue pq = new Priority<>(new Comparor{

                @Override

                public int compare(Node a, Node b){

                if(a.v b.v1)

                else a v2 b v2

                }

                })

                或者
                Queue pq = new Priority<>(){}
                Node需要重写compareTo方法。

                Arrays.sort(num, new Comparator) **只适用于in t[]数组等基本类型，**
            2. 对于非基本类，可以用Collections.sort(nodes, new Comparator() {
                List<?T>都可以
            3. o1 - o2 升序，大于0才需要交换。
        3. 给定的默认容量为 16，负载因子为 0.75。Map 在使用过程中不断的往里面存放数据，当数量达到了 `16 * 0.75 = 12` 就需要将当前 16 的容量进行扩容，而扩容这个过程涉及到 rehash、复制数据等操作，所以非常消耗性能。

            因此通常建议能提前预估 HashMap 的大小最好，尽量的减少扩容带来的性能损耗。
        4. 从这两个核心方法（get/put）可以看出 1.8 中对大链表做了优化，修改为红黑树之后查询效率直接提高到了 `O(logn)`。
            但是 HashMap 原有的问题也都存在，比如在并发场景下使用时容易出现死循环。
        5. EntrySet 进行遍历。

可以把 key value 同时取出，第二种还得需要通过 key 取一次 value，效率较低。

6. JDK 推出了专项专用的 ConcurrentHashMap ，该类位于 `java.util.concurrent` 包下，专门用于解决并发问题。ConcurrentHashMap 同样也分为 1.7 、1.8 版，两者在实现上略有不同。
    1. 如图所示，是由分段锁 Segment 数组（一锁多个Entry）、HashEntry 组成，和 HashMap 一样，仍然是数组加链表。
    2. 1.8抛弃了原有的 Segment 分段锁，而采用了 `CAS + synchronized` 来保证并发安全性。也将 1.7 中存放数据的 HashEntry 改为 Node，但作用都是相同的。

       其中的 `val next` 都用了 volatile 修饰，保证了可见性。
7.
    1. ReentrantLock和synchronized的区别。**锁的细粒度和灵活度**：很明显ReenTrantLock优于Synchronized在
    2. Synchronized优化以前，synchronized的性能是比ReenTrantLock差很多的，但是自从Synchronized引入了偏向锁，轻量级锁（自旋锁）后，两者的性能就差不多了，在两种方法都可用的情况下，官方甚至建议使用synchronized，其实synchronized的优化我感觉就借鉴了ReenTrantLock中的CAS技术。都是试图在用户态就把加锁问题解决，避免进入内核态的线程阻塞。
    3. Synchronized经过编译后，会在同步块前后分别形成monitorenter和monitorexit两个字节码指令，在执行monitorenter指令时，首先要尝试获取对象锁，如果对象没有别锁定，或者当前已经拥有这个对象锁，把锁的计数器加1，相应的在执行monitorexit指令时，会将计数器减1，当计数器为0时，锁就被释放了。如果获取锁失败，那当前线程就要阻塞，直到对象锁被另一个线程释放为止。
    4. ReentrantLock**等待可中断**，持有锁的线程长期不释放的时候，正在等待的线程可以选择放弃等待，这相当于Synchronized来说可以避免出现死锁的情况。通过lock.lockInterruptibly()来实现这个机制。
       Synchronized不可中断
    5. 多个线程等待同一个锁时，必须按照申请锁的时间顺序获得锁，Synchronized锁非公平锁，ReentrantLock默认的构造函数是创建的非公平锁，可以通过参数true设为公平锁，但公平锁表现的性能不是很好。
    6. 总结：reentrantLock实现更加灵活、粒度更小，以前效率比synchronized高，CAS实现，尽量在用户态解决冲突的问题。可以实现公平锁，即原生的队列实现AQS，而非公平锁可能会有老线程饥饿的情况。AQS是一个线程等待队列，还有一个状态同步器依赖volatile关键字，AbstractQueuedSynchronizer。

2. JVM
    1. 类加载
    2. 内存模型，运行时数据区。
        1. 1.7方法区属于JVM，1.8方法区在元空间，就是直接内存，一般IO的buffer也在直接内存里面。属于OS的空间而不是虚拟机。
    3. GC垃圾回收器
        1. 垃圾回收算法
            1. 标记清除法，标记即清除，清除速度快，但是空间不连续，容易产生内存碎片
            2. 标记整理法，区别在于整理，解决了内存碎片的问题。涉及到对象引用的改变，速度比较慢
            3. 复制算法。两块区域，存活对象移动到第二块区域，然后清除之后再交换回

来，优点不会产生碎片，缺点 占用双倍空间。
4. JVM是分代回收，因为新生代新垃圾比较频繁，为了内存碎片与性能考虑，使用复制算法。老年代因为存活的对象多，垃圾比较少，所以使用标记清除即可。新生代存活次数超过16晋升老年代。

2. 四种引用
    1. 强引用，不会被垃圾回收器回收，会报Out Of Memory错误。new的实例就是强引用
    2. 软引用，GC内存满时会回收，soft Reference，一般缓存可以用软引用
    3. 弱引用，比较大的文件，需要读一次删一次。weak reference，只要GC就会被回收，threadLocal因此造成内存泄露。
    4. 虚引用，随时都有可能被回收。
3. 垃圾回收器
    1. 串行，单线程，适合单人电脑。--XX -serial 开启串行
    2. 吞吐量优先，多线程，堆内存较大，多核cpu支持。**单位时间内**STW卡顿时间最短，吞吐量最大。
    3. 响应时间优先，尽量单次STW卡顿时间最短，最快响应。
4. JVM调优
    1. 一般调整堆栈的大小比例，
    2. 调整老年代和新生代的比例，
3.

13. 开放式问题，高可用、高性能、分布式、rpc、
14. 原来的项目，一个请求的过程，spring视角、计算机网络视角
    spring：首先要组装一个请求体，路由到指定的机器的服务，根据url路径，通过一个dispatch定位到指定的controller的接口，然后调用相应的service服务，执行一定的逻辑的时候，根据需要再到持久层进行增删改查操作。执行完了之后再返回一个信息组装成一个返回体给前端展示给用户。
    计算机网络视角，一次URL：应用层准备好应用数据，与服务器开启一次会话，这个期间要把应用数据经过加密等操作，包装成tcp支持的单元，就是加请求头组装成一个一个的TCP报文与服务器建立连接，传输层建立连接需要经历三次握手，根据序号和ack确认机制来保证数据传输的安全，同时在网络情况不好的时候也会加一些方法来进行拥塞控制，在与服务器交互的过程中，一个个的报文分成了更小的单元，就是网络层的概念，ip数据报是比较小的单位，1500字节一个，根据ip地址和相应的路由协议来找到合适的网关到达目的的网络。到了目的网络的时候，还需要找到对应的主机，根据ARP协议找到主机的mac地址，链路之间传输会分解成一个个的数据帧，再细说就是在网线里面分频或者分道编码解码之类的到目标机器里面。
15. 为什么不继续在腾讯实习
    首先是部门超编了，入职的时候就明确告诉我没有转正名额了，其实部门人都很好
16. 自己业务的思考，数据链路：构建->追查case->写一些需求->改造读写方式->迁移。整个召回排序的架构，帅哥文档
    看以后分在什么岗位，是偏业务的还是偏基础架构的，
17. 对面试官有什么问题，
    业务大概是什么、我进去的技术栈是什么、进去之后表现好有转正机会吗