**jdk jre jvm 三者的区别**

**JVM** ：英文名称（Java Virtual Machine），就是我们耳熟能详的 Java 虚拟机。它只认识 xxx.class 这种类型的文件，它能够将 class 文件中的字节码指令进行识别并调用操作系统向上的 API 完成动作。所以说，jvm 是 Java 能够跨平台的核心，具体的下文会详细说明。

**JRE** ：英文名称（Java Runtime Environment），我们叫它：Java 运行时环境。它主要包含两个部分，jvm 的标准实现和 Java 的一些基本类库。它相对于 jvm 来说，多出来的是一部分的 Java 类库。

**JDK** ：英文名称（Java Development Kit），Java 开发工具包。jdk 是整个 Java 开发的核心，它集成了 jre 和一些好用的小工具。例如：javac.exe，java.exe，jar.exe 等。

显然，这三者的关系是：一层层的嵌套关系。**JDK>JRE>JVM**。

**HDFS的副本数为什么为3，为什么不是2或者是4？**

HDFS的副本数默认是3，并不是必须是3.

修改hdfs的副本数，首先想到的是修改hdfs-site.xml中的dfs.replication参数

HDFS采用一种称为机架感知的策略来改进数据的可靠性、可用性和网络带宽的利用率。

在大多数情况下，HDFS的副本系数是3，HDFS的存放策略是一个副本存放在本地机架节点上，另一个副本存放在同一机架的另一个节点上，第三个副本存放在在不同机架的节点上。这种策略减少了机架间的数据传输，提高了写操作的效率。机架错误的概率远比节点错误的概率小，所以这种策略不会对数据的可靠性和可用性造成影响。与此同时，因为数据只存在两个机架上，这种策略减少了读数据时需要的网络传输带宽。

在这种策略下，副本并不是均匀地分布在机架上。这种策略在不损坏可靠性和读取性能的情况下，改善了写的性能。

**怎么查看端口号 怎么查看进程**

查看端口号netstat  -anp  |grep   端口号

ps -aux | grep \*\*\* 查询\*\*\*进程的详细信息

ps -aux 查询内存中进程信息

**查看java进程**

ps -ef | grep java

jps

MR 的工作原理

略，请查看之前资料。

**说下隐语义模型的应用案例**

属于推荐系统算法的一种，应用案例需要结合自己项目来进行展开。（如果自己项目写了的话。）

LFM（latent factor model）隐语义模型的核心思想是通过隐含特征(latent factor)联系用户兴趣和物品，采取基于用户行为统计的自动聚类。

        隐含语义分析技术的分类来自对用户行为的统计，代表了用户对物品分类的看法。隐含语义分析技术和ItemCF在物品分类方面的思想类似，如果两个物品被很多用户同时喜欢，那么这两个物品就很有可能属于同一个类。

        隐含语义分析技术允许我们指定最终有多少个分类，这个数字越大，分类的粒度就会越细，反正分类粒度就越粗。

        隐含语义分析技术会计算出物品属于每个类的权重，因此每个物品都不是硬性地被分到某一个类中。

        隐含语义分析技术给出的每个分类都不是同一个维度的，它是基于用户的共同兴趣计算出来的，如果用户的共同兴趣是某一个维度，那么LFM给出的类也是相同的维度。

        隐含语义分析技术可以通过统计用户行为决定物品在每个类中的权重，如果喜欢某个类的用户都会喜欢某个物品，那么这个物品在这个类中的权重就可能比较高。

**用java开发过吗？**

一般建议回答开发过，比如我们在开发Hive UDF函数时，写HDFS API，HBase API时，都是使用Java进行开发的。之前也使用Java写过简单的类，接口等代码。

**多线程是怎么运作的。**

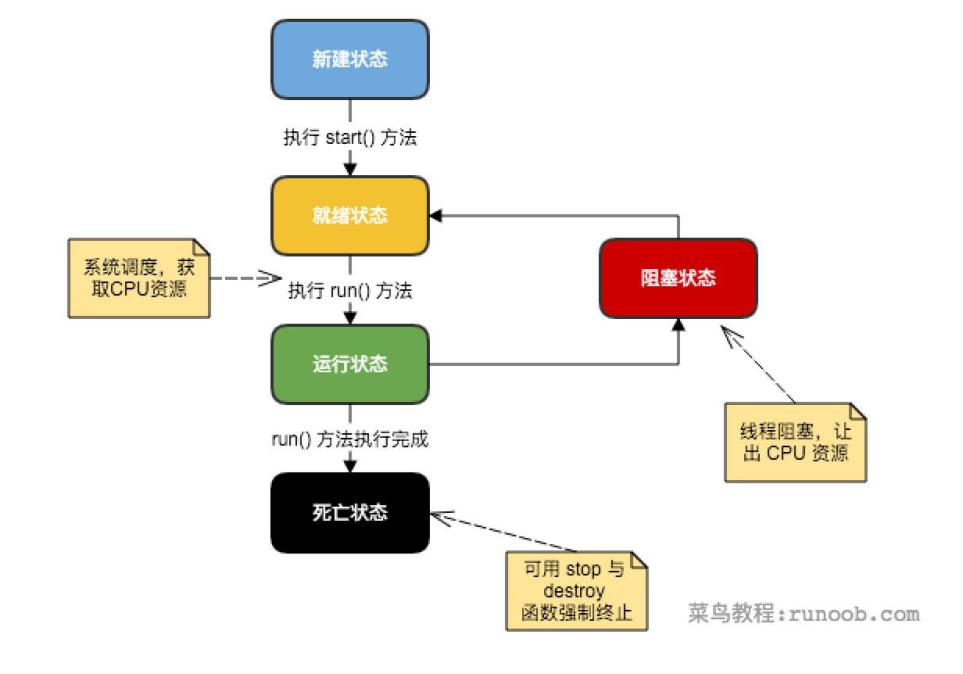
多线程的优势有：

    （1）进程之间不能共享数据，线程可以；

    （2）系统创建进程需要为该进程重新分配系统资源，故创建线程代价比较小；

    （3）Java语言内置了多线程功能支持，简化了java多线程编程。

线程的生命周期：



**1、新建状态**

       用new关键字和Thread类或其子类建立一个线程对象后，该线程对象就处于新生状态。处于新生状态的线程有自己的内存空间，通过调用start方法进入就绪状态（runnable）。

注意：不能对已经启动的线程再次调用start()方法，否则会出现[Java](http://lib.csdn.net/base/17" \o "Java EE知识库" \t "_blank).lang.IllegalThreadStateException异常。

**2、就绪状态**

       处于就绪状态的线程已经具备了运行条件，但还没有分配到CPU，处于线程就绪队列（尽管是采用队列形式，事实上，把它称为可运行池而不是可运行队列。因为cpu的调度不一定是按照先进先出的顺序来调度的），等待系统为其分配CPU。等待状态并不是执行状态，当系统选定一个等待执行的Thread对象后，它就会从等待执行状态进入执行状态，系统挑选的动作称之为“cpu调度”。一旦获得CPU，线程就进入运行状态并自动调用自己的run方法。

提示：如果希望子线程调用start()方法后立即执行，可以使用Thread.sleep()方式使主线程睡眠一伙儿，转去执行子线程。

**3、运行状态**

      处于运行状态的线程最为复杂，它可以变为阻塞状态、就绪状态和死亡状态。

处于就绪状态的线程，如果获得了cpu的调度，就会从就绪状态变为运行状态，执行run()方法中的任务。如果该线程失去了cpu资源，就会又从运行状态变为就绪状态。重新等待系统分配资源。也可以对在运行状态的线程调用yield()方法，它就会让出cpu资源，再次变为就绪状态。

当发生如下情况是，线程会从运行状态变为阻塞状态：

     ①、线程调用sleep方法主动放弃所占用的系统资源

     ②、线程调用一个阻塞式IO方法，在该方法返回之前，该线程被阻塞

     ③、线程试图获得一个同步监视器，但更改同步监视器正被其他线程所持有

     ④、线程在等待某个通知（notify）

     ⑤、程序调用了线程的suspend方法将线程挂起。不过该方法容易导致死锁，所以程序应该尽量避免使用该方法。

当线程的run()方法执行完，或者被强制性地终止，例如出现异常，或者调用了stop()、desyory()方法等等，就会从运行状态转变为死亡状态。

**4、阻塞状态**

      处于运行状态的线程在某些情况下，如执行了sleep（睡眠）方法，或等待I/O设备等资源，将让出CPU并暂时停止自己的运行，进入阻塞状态。

在阻塞状态的线程不能进入就绪队列。只有当引起阻塞的原因消除时，如睡眠时间已到，或等待的I/O设备空闲下来，线程便转入就绪状态，重新到就绪队列中排队等待，被系统选中后从原来停止的位置开始继续运行。有三种方法可以暂停Threads执行：

**5、死亡状态**

      当线程的run()方法执行完，或者被强制性地终止，就认为它死去。这个线程对象也许是活的，但是，它已经不是一个单独执行的线程。线程一旦死亡，就不能复生。 如果在一个死去的线程上调用start()方法，会抛出java.lang.IllegalThreadStateException异常。

线程的创建和启动方式：

1）继承Thread类创建线程类

通过继承Thread类创建线程类的具体步骤和具体代码如下：

   • 定义一个继承Thread类的子类，并重写该类的run()方法；

   • 创建Thread子类的实例，即创建了线程对象；

   • 调用该线程对象的start()方法启动线程。

2）实现Runnable接口创建线程类

通过实现Runnable接口创建线程类的具体步骤和具体代码如下：

   • 定义Runnable接口的实现类，并重写该接口的run()方法；

   • 创建Runnable实现类的实例，并以此实例作为Thread的target对象，即该Thread对象才是真正的线程对象。

（3）通过Callable和Future创建线程

通过Callable和Future创建线程的具体步骤和具体代码如下：

   • 创建Callable接口的实现类，并实现call()方法，该call()方法将作为线程执行体，并且有返回值。  
   • 创建Callable实现类的实例，使用FutureTask类来包装Callable对象，该FutureTask对象封装了该Callable对象的call()方法的返回值。  
   • 使用FutureTask对象作为Thread对象的target创建并启动新线程。  
   • 调用FutureTask对象的get()方法来获得子线程执行结束后的返回值其中

线程管理

**线程睡眠——sleep**

**线程让步——yield**

**线程合并——join**

**设置线程的优先级**

**后台（守护）线程**

线程同步

 java允许多线程并发控制，当多个线程同时操作一个可共享的资源变量时（如数据的增删改查），将会导致数据不准确，相互之间产生冲突，因此加入同步锁以避免在该线程没有完成操作之前，被其他线程的调用，从而保证了该变量的唯一性和准确性。

1、同步方法

      即有synchronized关键字修饰的方法。由于java的每个对象都有一个内置锁，当用此关键字修饰方法时，内置锁会保护整个方法。在调用该方法前，需要获得内置锁，否则就处于阻塞状态。

public synchronized void save(){}

 注： synchronized关键字也可以修饰静态方法，此时如果调用该静态方法，将会锁住整个类

 2、同步代码块

     即有synchronized关键字修饰的语句块。被该关键字修饰的语句块会自动被加上内置锁，从而实现同步。

**3、使用特殊域变量(volatile)实现线程同步**

**4、使用重入锁（Lock）实现线程同步**

多线程时java最喜欢问的问题之一。详细看链接

<https://www.cnblogs.com/snow-flower/p/6114765.html>

sql的一些问题：group by 时候前面字段比较多？

kafka是怎么工作的？

**zookeeper有了解吗？**

zookeeper是一个分布式服务框架，主要是用来解决分布式应用中经常遇到的一些数据管理问题，如：统一命名服务、状态同步服务、**集群管理**、分布式应用配置项的管理等。有较好的容错机制，我们在实现Hadoop集群高可用，HBase集群元数据管理方面都用到了zookeeper。

Zookeeper的角色一般包含leader, Follower. 我们开发环境中的zookeeper节点是3台，一个是leader，剩下的是follower。（可能面试官会问zookeeper选举机制，回答不知道就行了，说清楚的话比较复杂）

**平时开发用什么工具？**

IDEA(Java开发工具)，Xshell（连接服务器，注意是服务器，不是虚拟机），Git（版本控制工具，说会的话，一些命令要知道，另外，私服是收费的）

**linux  netcat是怎么玩的？**

Netcat是一款非常出名的网络工具，简称“NC”,有渗透测试中的“瑞士军刀”之称。它可以用作端口监听、端口扫描、远程文件传输、还可以实现远程shell等功能。简单用过一些，比如nc -lk 9999之类的。

你们搭建kafka是怎么测试连接的？

**kafka的acks机制？怎么保证kafka最大吞吐量？**

acks参数指定了在集群中有多少个分区副本收到消息，kafka producer才会认为消息是被写入成功。这个参数对消息丢失的可能性有很大的影响

如果acks=0，生产者在成功写入消息之前是不会等待任何的来自服务器的响应。也就是说如果当中出现了错误，导致broker没有收到消息，那么生产者是无从得知的，消息也就丢失了。不过，因为生产者不需要等待服务器的响应，从而可以以网络可以支持的最大的速度来发送消息，使得系统能够达到很高的吞吐量。

如果acks=1，只要集群的leader节点收到消息，生产者就会收到来自服务器成功的响应。若果消息不能够被leader节点接收（比如说leader节点崩溃，而新的leader尚未选出来），这时候生产者会收到一个错误响应，但是为了避免数据的丢失，生产者会重发消息。不过，如果一个没有收到消息的节点成为新leader，消息还是会丢失的。这个时候的吞吐量取决于使用的同步还是异步发送。如果让发送客户端（生产者）等待服务器的响应（通过调用Futrue对象的get()方法），显然会增加延迟（一次发送和响应的会话延迟）。如果客户端（生产者）使用回调，延迟问题就可以得到解决了，不过吞吐量还是会受到发送中消息数量的限制（比如生产者在收到服务器响应之前可以发送多少个消息）

如果acks=all / -1，只有在集群所有的跟随副本都接收到消息后，生产者才会受到一个来自服务器的成功响应。这种模式是最安全的，它可以保证集群中不止一个服务器接收到消息，就算有服务器崩溃了，这个集群还是能够正常运行。不过它比acks=1的延迟性更高，因为生产者要等待的所有参与复制消息的节点接收到消息。

我们为了保证kafka的吞吐量，以及消息的不丢失，ack设的为1.

**Kafka 高吞吐率的实现**

**顺序读写**

kafka的消息是不断追加到文件中的，这个特性使kafka可以充分利用磁盘的顺序读写性能顺序读写不需要硬盘磁头的寻道时间，只需很少的扇区旋转时间，所以速度远快于随机读写

**零拷贝**

**文件分段**  
      kafka的队列topic被分为了多个区partition，每个partition又分为多个段segment，所以一个队列中的消息实际上是保存在N多个片段文件中通过分段的方式，每次文件操作都是对一个小文件的操作，非常轻便，同时也增加了并行处理能力

**批量发送**

      Kafka允许进行批量发送消息，先将消息缓存在内存中，然后一次请求批量发送出去比如可以指定缓存的消息达到某个量的时候就发出去，或者缓存了固定的时间后就发送出去如100条消息就发送，或者每5秒发送一次这种策略将大大减少服务端的I/O次数

**数据压缩**

      Kafka还支持对消息集合进行压缩，Producer可以通过GZIP或Snappy格式对消息集合进行压缩压缩的好处就是减少传输的数据量，减轻对网络传输的压力

**Consumer的负载均衡**

当一个group中,有consumer加入或者离开时,会触发partitions均衡.均衡的最终目的,是提升topic的并发消费能力

**shell 的if判断的条件，什么时候是true？**

不懂这句话的意思。。。，是下面这个意思吗？

**字符串判断**

str1 = str2　　　　　　当两个串有相同内容、长度时为真  
str1 != str2　　　　　 当串str1和str2不等时为真  
-n str1　　　　　　　 当串的长度大于0时为真(串非空)  
-z str1　　　　　　　 当串的长度为0时为真(空串)  
str1　　　　　　　　   当串str1为非空时为真

Kafka的Topic特点

**Flume的扇入扇出**

在flume中有时候需要将一个源（source）将数据发送到多个地方（sink），在flume中该术语叫做扇出（fan out）

多个source配一个channel和一个sinks，这叫扇入；

**ELK的几个查询关键词**

"query": 在请求消息体中的query允许我们用Query DSL的方式查询。

"term": 查询时判断某个document是否包含某个具体的值，不会对被查询的值进行分词查询

"match" 将被查询值进行分词，然后用评分机制(TF/IDF)进行打分

"match\_phrase": 查询指定段落

"Bool": 结合其他真值查询，通常和must should mustnot(与或非)一起组合出复杂的查询

"range": 查询时指定某个字段在某个特定的范围

"from": 以一定的偏移量来查看我们检索的结果，缺省从检索的第一条数据开始显示

"size": 指定检索结果中输出的数据条数，缺省为10条

"sort": 允许我们将检索的结果以指定的字段进行排序显示

"aggs": 基于搜索查询，可以嵌套聚合来组合复杂的需求

**为什么HBase查询比较快**

主要原因是由其架构和底层的数据结构决定的，即由LSM-Tree(Log-Structured Merge-Tree) + HTable(region分区) + Cache决定

客户端可以直接定位到要查数据所在的HRegion server服务器，然后直接在服务器的一个region上查找要匹配的数据，并且这些数据部分是经过cache缓存的

HBase会将数据保存到内存中，在内存中的数据是有序的，如果内存空间满了，会刷写到HFile中，而在HFile中保存的内容也是有序的。当数据写入HFile后，内存中的数据会被丢弃。HFile文件为磁盘顺序读取做了优化

HBase的写入速度快是因为它其实并不是真的立即写入文件中，而是先写入内存，随后异步刷入HFile。所以在客户端看来，写入速度很快。另外，写入时候将随机写入转换成顺序写，数据写入速度也很稳定。读取速度快是因为它使用了LSM树型结构，而不是B或B+树。磁盘的顺序读取速度很快，但是相比而言，寻找磁道的速度就要慢很多。HBase的存储结构导致它需要磁盘寻道时间在可预测范围内，并且读取与所要查询的rowkey连续的任意数量的记录都不会引发额外的寻道开销。比如有5个存储文件，那么最多需要5次磁盘寻道就可以。而关系型数据库，即使有索引，也无法确定磁盘寻道次数。而且，HBase读取首先会在缓存（BlockCache）中查找，它采用了LRU（最近最少使用算法），如果缓存中没找到，会从内存中的MemStore中查找，只有这两个地方都找不到时，才会加载HFile中的内容

**工作中用什么将Kafka中的数据导入HBase**

一般使用Java API进行，也可以使用Scala，使用Spark Streaming，将Kafka数据导入HBase

**Linux查看一个文件的大小**

du -h

**Linux如何查看一个文件后500行数据**

tail -n 500 aaa.txt

Spark算子的种类，分别说几个

略

**你在项目中主要负责的部分**

参考简历上个人职责。

**使用hive怎么过滤空值**

（1）不同数据类型对空值的存储规则

int与string类型数据存储，null默认存储为 \N；

string类型的数据如果为""，存储则是""；

另外往int类型的字段插入数据“”时，结果还是\N。

（2）不同数据类型，空值的查询

对于int可以使用is null来判断空；

而对于string类型，条件is null 查出来的是\N的数据；而条件 =’’，查询出来的是""的数据。

结论：判断空时要根据实际的存储来进行判断。在开发过程中如果需要对空进行判断，一定得知道存储的是哪种数据。

有个处理空的小技巧，Hive给出一种并非完美的解决方法——自定义底层用什么字符来表示NULL

**3 索引类型有哪些？**

普通索引：仅加速查询

唯一索引：加速查询 + 列值唯一（可以有null）

主键索引：加速查询 + 列值唯一（不可以有null）+ 表中只有一个

组合索引：多列值组成一个索引，专门用于组合搜索，其效率大于索引合并

全文索引：对文本的内容进行分词，进行搜索

**4 数据库如果考虑时区，用什么类型？**

TIMESTAMP是标准的unix timestamp，它存储的是1970-1-1到现在经过的秒数，4字节存储

DATETIME的范围更大，好像可以从0000-00-00 00:00:00到9999-12-31 23:59:59，8字节存储

select

date\_format(from\_utc\_timestamp(to\_utc\_timestamp("中国北京时间",'GMT+8'),"要转化地方的GMT"),'yyyy-MM-dd HH:mm:ss') as local\_time

from ccsd.tb\_name

**5 组合索引使用like还有效吗?**

like在某些情况下是不会使用索引的，但是有些情况是走索引的：例如：

like '%keyword%'或者like '%keyword'是不会走索引的，但是当like 'keyword%'也不一定就会走索引，这样看具体情况，如果select中查询了其他字段【就是这个字段没有在这个like所引用的组合索引或者索引字段中，除了id字段之外】这样的情况就不会走索引，如果想要走自己想要的索引，可以添加上force index(ziduan1\_index)，这样就可以了

使用like走索引的情况:查询的字段都得是组合索引中的才行

**6 组合索引怎么使用才会生效**

建立的组合索引顺序：（a,b,c,d,e,f）

1、如果组合索引中的索引都在，则索引全部生效

2、如果组合索引中最左边的索引a不存在，那么索引都不生效

3、如果组合索引中最左边的索引a存在但没有使用全部的索引。首先按照建立索引的顺序排好，从a开始往后判断是否具有连续性（依据建立索引的顺序），如果直至最后都具有连续性，那么这些索引生效。如果在下一个节点处的索引连续性断掉，那么当前节点的索引和左边的索引生效。当前节点右边的索引不生效

**7 事务的特性？**

1：原子性：事务包含的所有操作要么全部成功，要么全部失败回滚;成功必须要完全应用到数据库，失败则不能对数据库产生影响；

2：一致性：事务执行前和执行后必须处于一致性状态，

例：用户A和用户B的前加起来一共是5000; 无论AB用户之间是如何相互转换的，事务结束后两个用户的钱加起来还是5000,这就是事务的一致性。

3：隔离性：当多个用户并发访问数据库时，数据库为每一个用户开启的事务，不被其他事务的操作所干扰，多个并发事务之间要相互隔离；

4：持久性：一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便在数据库系统遇到故障的情况下也不会丢失事物的操作。

**8 数据库隔离级别**

01：Read uncommitted(读未提交)：最低级别，任何情况都会发生。  
02：Read Committed(读已提交)：可避免脏读的发生。  
03：Repeatable read(可重复读)：可避免脏读、不可重复读的发生。  
04：Serializable(串行化)：避免脏读、不可重复读，幻读的发生。

**9 linux系统常用的操作命令**

增加awk和sed的复习。

**10 自己在linux上装过哪些软件？**

Hadoop、Hive、zookeeper、HBase、Java、Spark都可以说。尽量说自己熟悉的。

**11 对堆和栈的理解？**

JVM虚拟机的问题。详情查看JVM虚拟机相关内容。

**12 java数据结构**

1数组  
2链表  
3栈和队列  
4二叉树  
5堆和堆栈  
6散列表  
7红黑树 <https://www.jianshu.com/p/8e54797ec3e0>

还有一个比较简单的问题，是java数据类型，要知道并答出8中基本数据类型，以及其他复杂数据类型

13 双向 链表如果从中取出一个数据，如何使其还是一个双向链表？

14 算法了解吗?知道哪些？说下原理

15 离线大数据和实时大数据的区别?你如何技术选型？

16 离线数据流向过程

17 kafka熟吗？

18 非结构化数据库有哪些?用过哪些?

HBase、MongoDB、Redis

19 kafka有哪些节点类型

Broker，producer，consumer

20 如果3台kafka的集群宕机了一台，怎么办？宕了两台呢?

21 kafka和传统消息队列有何区别？

22 集群是动态增加的吗？

23 触发器的作用

24 工作中SQL的优化

25 项目耗时多久?从什么状态开始的？什么状态结束的？你负责哪些？

1，你写的所有项目，说一遍

2，MR的执行流程

3，flume扇入扇出

**4，kafka 的topic分区设计和consumer有什么关系**

每个group中可以有多个consumer，每个consumer属于一个consumer group；  
通常情况下，一个group中会包含多个consumer，这样不仅可以提高topic中消息的并发消费能力，而且还能提高”故障容错”性，如果group中的某个consumer失效那么其消费的partitions将会有其他consumer自动接管。  
 对于Topic中的一条特定的消息，只会被订阅此Topic的每个group中的其中一个consumer消费，此消息不会发送给一个group的多个consumer；  
那么一个group中所有的consumer将会交错的消费整个Topic，每个group中consumer消息消费互相独立，我们可以认为一个group是一个”订阅”者。  
 在kafka中,一个partition中的消息只会被group中的一个consumer消费(同一时刻)；  
一个Topic中的每个partions，只会被一个”订阅者”中的一个consumer消费，不过一个consumer可以同时消费多个partitions中的消息。  
 kafka的设计原理决定,对于一个topic，同一个group中不能有多于partitions个数的consumer同时消费，否则将意味着某些consumer将无法得到消息。  
kafka只能保证一个partition中的消息被某个consumer消费时是顺序的；事实上，从Topic角度来说,当有多个partitions时,消息仍不是全局有序的。

5，tableau怎么用的。

6，为什么用hbase存储

**7，hive的优化**

<https://www.cnblogs.com/swordfall/p/11037539.html>

**8，hive数据倾斜原因和处理**

<https://blog.csdn.net/XiaodunLP/article/details/87866350>

<https://blog.csdn.net/jin6872115/article/details/79878391>

**spark 数据倾斜**

<https://blog.csdn.net/Fenggms/article/details/83020683>

9，flink了解么

了解一些，Flink核心是一个流式的数据流执行引擎，类似于sparkStreaming,常被说为是第三代流数据处理引擎。但是在工作中没有用过，有自己学习一些。主要有两大类API, 对Streaming数据类应用，提供DataStream API；对批处理类应用，提供DataSet API（支持Java/Scala）

10，数据仓库用过哪些模型，说几个，例如雪花和星型

11，用过哪些算法

12，java线程怎么保证同步

参考上文多线程内容

13，线程锁的实现方式有几种

**14，java  GC回收垃圾说一下**

<https://blog.csdn.net/laomo_bible/article/details/83112622>

JDK8 默认GC类型： 默认使用的是ParallelGC，对应收集器为：

新生代使用的是：Parallel Scavenge收集器  
老年代使用的是：Serial Old收集器，

JDK8新增G1收集器

java -XX:+PrintCommandLineFlags -version

[-XX：UseParallelGC](http://codewenda.com/xx%EF%BC%9Auseparallelgc%E5%92%8C-xx%E4%B9%8B%E9%97%B4%E7%9A%84%E5%8C%BA%E5%88%AB%EF%BC%9A-useparnewgc/)

**重点回答内容：JVM组成（堆内存，栈），GC类型，收集器类型，GC过程，GC算法等（详细参考链接）**

15，jvm底层怎么玩的

16，手写一个java 的设计模式

23种设计模式，常见的如单例模式，工厂模式，装饰模式，

总体来说设计模式分为三大类：

1. 创建型

共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。

2. 结构型

共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。

3. 行为型

共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

17，python熟么，写过项目么

Web项目写Django项目，爬虫项目Scrapy项目

Spark Join的优化经验

Spark和flink的区别

**Kafka和传统的MQ之间的区别**

1.架构模型方面

RabbitMQ遵循AMQP协议，RabbitMQ的brokerExchange,Binding,queue组成，其中exchange和binding组成了消息的路由键；客户端Producer通过连接channel和server进行通信，Consumer从queue获取消息进行消费（长连接，queue有消息会推送到consumer端，consumer循环从输入流读取数据）。rabbitMQ以broker为中心；有消息的确认机制。

kafka遵从一般的MQ结构，producer，broker，consumer，以consumer为中心，消息的消费信息保存的客户端consumer上，consumer根据消费的点，从broker上批量pull数据；无消息确认机制。

2.吞吐量

kafka具有高的吞吐量，内部采用消息的批量处理，zero-copy机制，数据的存储和获取是本地磁盘顺序批量操作，具有O(1)的复杂度，消息处理的效率很高。

rabbitMQ在吞吐量方面稍逊于kafka，他们的出发点不一样，rabbitMQ支持对消息的可靠的传递，支持事务，不支持批量的操作；基于存储的可靠性的要求存储可以采用内存或者硬盘。

3.可用性

rabbitMQ支持miror的queue，主queue失效，miror queue接管。

kafka的broker支持主备模式。

4.集群负载均衡

kafka采用zookeeper对集群中的broker、consumer进行管理，可以注册topic到zookeeper上；通过zookeeper的协调机制，producer保存对应topic的broker信息，可以随机或者轮询发送到broker上；并且producer可以基于语义指定分片，消息发送到broker的某分片上。

Kafka的应用和理解

Flume的扇出应用的好处

Elasticsearch 的主要应用以及理解

系统调度周期多长

一般一天或一周较多，视自己项目业务而定。

调度设计的理解实现

产品经理的职责以及与你的联系

产品经理提需求等，

分类聚类算法的理解，以及在项目中的实现

d

项目中出现的挑战问题

项目步骤的解释

**Hive执行计划**

<https://www.cppentry.com/bencandy.php?fid=117&id=201834>

hive用的多么 hive动态分区和分桶

略。

**sqoop工具可以条件导出吗**

可以条件导出

**sqoop代码怎么写**

sqoop import --connect jdbc:mysql://192.168.1.1:3306/events --username root --passwd 123456 --table  student  --hive-import --hive-table student -m 1

**数仓分层 数仓和普通数据库区别**

传统的分层就是分为3层 即ODS，DW，DM层，在我们项目里面，是将DW层细分为了DWD和DWS层，DWD主要用来存事实表

源数据层：原始数据。来源包含业务库、埋点日志、其他数据源

ODS层：Operate data store，操作数据存储，是最接近数据源中数据的一层，数据源中的数据，经过抽取、洗净、传输，也就说传说中的ETL之后，装入本层。本层的数据，总体上大多是按照源头业务系统的分类方式而分类的。但是，这一层面的数据却不等同于原始数据。在源数据装入这一层时，要进行诸如去噪(例如有一条数据中人的年龄是 300 岁，这种属于异常数据，就需要提前做一些处理)、去重(例如在个人资料表中，同一 ID 却有两条重复数据，在接入的时候需要做一步去重)、字段命名规范等一系列操作。

DW层：Data warehouse，数据仓库层。在这里，从ODS层中获得的数据按照主题建立各种数据模型。例如以研究人的旅游消费为主题的数据集中，便可以结合航空公司的登机出行信息，以及银联系统的刷卡记录，进行结合分析，产生数据集。在这里，我们需要了解四个概念：维（dimension）、事实（Fact）、指标（Index）和粒度（ Granularity）

（DW可以分为DWD和DWS层）

DWD：data warehouse detail 细节数据层。是业务层与数据仓库的隔离层。这一层主要解决一些数据质量问题和数据的完整度问题。比如用户的资料信息来自于很多不同表，而且经常出现延迟丢数据等问题，为了方便各个使用方更好的使用数据，我们可以在这一层做一个屏蔽。 dwd 主要是对 ods 层做一些数据清洗和规范化的操作。

DWS层：data warehouse service 服务数据层， 轻度汇总层，从ODS层中对用户的行为做一个初步的汇总，抽象出来一些通用的维度：时间、ip、id，并根据这些维度做一些统计值，比如用户每个时间段在不同登录ip购买的商品数等。这里做一层轻度的汇总会让计算更加的高效。整合汇总成分析某一个主题域的服务数据，一般是宽表。DWS层主要是用来存储对数据进行小量聚合的，DWS层可以解决企业中百分之70的问题，剩下的解决不了的那就是再进行一次高度聚合，将结果放在DM层中。

dws 和 dwd 有没有依赖关系主要看有没有这种需求。站在一个理想的角度来讲，如果 ods 层的数据就非常规整，基本能满足我们绝大部分的需求，这当然是好的，这时候 dwd 层其实也没太大必要。 但是现实中接触的情况是 ods 层的数据很难保证质量，毕竟数据的来源多种多样，推送方也会有自己的推送逻辑，在这种情况下，我们就需要通过额外的一层 dwd 来屏蔽一些底层的差异。

DM层（**data market**）：又称数据集市或宽表。按照业务划分，如流量、订单、用户等，生成字段比较多的宽表，用于提供后续的业务查询，OLAP分析，数据分发等。数据生成方式：由轻度汇总层和明细层数据计算生成。更深层的数据处理。比如这个项目当中的DWS层是统计过去30天内的，但是在DM层中会有统计过去90天的指标，可以理解为DM层是DWS层的又一次聚合。

APP层：应用层是根据业务需要，由前面三层数据统计而出的结果，可以直接提供查询展现，或导入至Mysql中使用。数据生成方式：由明细层、轻度汇总层，数据集市层生成，一般要求数据主要来源于集市层。该层主要是提供数据产品和数据分析使用的数据，一般会存放在es、mysql等系统中供线上系统使用。比如我们经常说的报表数据，或者说那种大宽表，一般就放在这里。

**数据仓库构建步骤：**

第一步：选择业务流程

维度方法的基础是首先确认哪些业务处理流程是数据库需要覆盖的,因此建模的第一个步骤是描述需要建模的业务流程,描述业务流程,可以简单的使用文本记录下俩或者使用MPMN(业务流程建模标注)的方法,也可以使用UML等.

第二步：确认粒度

粒度用于确定事实表中表示的是什么,在选择事实表存储最细粒度的事务记录，每小时更新增量，凌晨2两点更新昨天全量

第三步：确认维度

维度是指分析的各个角度，如版本、渠道、功能

维度的层次(Hierarchy)和级别(Level)

维度表随时间变化，缓慢维度变化：修正、迁移、新增

第四步：确认事实

将原始表与维度表进行关联，生成事实表

还需要建立一些相关的中间汇总表，以方便查询

第五步：建立物理模型

**oozie怎么用的？**

可以从以下几个方面回答：

1. 怎么开发workflow，调度哪些内容，比如hive,shell等
2. 怎么设置任务调度，定时时间，方式，
3. 结合项目说明

**项目里具体hive做的哪一块**

ETL中数据清洗，数据处理，数据分析。结合项目业务来讲。

华为不是用的oozie 用的**别的调度工具**

其他调度工具还有Azkaban、Airflow等，可以了解一下。

**hive的自定义函数实现了什么接口**

1）自定义UDF需要继承：org.apache.hadoop.hive.ql.UDF

2).需要evaluate函数，evaluate函数支持重载。

还有UDAF、UDTF

**hive举几个会发生mr过程的语句例子**

count，sum ,min ,avg ,max 函数，各种join操作

**hive外部表应用场景**

加载映射原始数据到Hive表

**scala的数据类型有哪些**

Scala 的数据类型和Java是类似的，所有Java的基本类型在scala包中都有对应的类，将Scala代码编译为Java字节码时，Scala编译器将尽可能使用Java的基本类型，从而提供基本类型的性能优势，8种基本数据类型，位于scala包中：Byte,Short,Int,Long,Float,Double,Char,Boolean

Any :Any是所有其他类的超类

AnyRef:Scala里所有引用类(reference class)的基类

Unit：只有一个实例值 ()，方法返回Unit 相当于返回了Java中的void

Null:每个引用类的子类

Nothing:在Scala的类层级的最低端；它是任何其他类型的子类型，可以赋值给任何其他类型，用于异常，表明不正常的返回

String类型

**scala的偏函数能和我说说吗？**

在Scala中，偏函数是**具有类型PartialFunction[-T,+V]的一种函数**。T是其接受的函数类型，V是其返回的结果类型。**偏函数最大的特点就是它只接受和处理其参数定义域的一个子集，而对于这个子集之外的参数则抛出运行时异常**。这与Case语句的特性非常契合，因为我们在使用case语句是，常常是匹配一组具体的模式，最后用“\_”来代表剩余的模式。如果一一组case语句没有涵盖所有的情况，那么这组case语句就可以被看做是一个偏函数。

<https://blog.csdn.net/bluishglc/article/details/50995939>

**谈谈scala的柯里化**

柯里化（Currying）是把接受多个参数的函数变换成接受一个单一参数(最初函数的第一个参数)的函数，并且返回接受余下的参数且返回结果的新函数的技术。

例如：

def add(x:Int,y:Int)=x+y

柯里化后：

def add(x:Int)(y:Int)=x+y

**scala的apply和unapply方法是什么作用**

apply方法

通常，在一个类的半生对象中定义apply方法，在生成这个类的对象时，就省去了new关键字。

unapply方法

可以认为unapply方法是apply方法的反向操作，apply方法接受构造参数变成对象，而unapply方法接受一个对象，从中提取值。

appay和unapply方法会被隐式调用。

**scala里面定义一个集合有哪些方法**

常用的集合有三种：List，Set，Map

List 继承至 Seq，集合中的元素是可以重复的 val l = List(1, 1, 2, 2, 3, 5)

Set 中的元素是不可重复的 val s = Set(1, 1, 2, 2, 3, 5)

Map 中的元素呈现 key -> value 的形式，其中第一个参数是映射的键，第二个参数是映射的值。且Map中每一个元素的键是唯一的。

val m = Map(1 -> 1, 1 -> 3, 2 -> 3, 3 -> 4, 4 -> 4, 5 -> 7)

以上内容， 还可以通过其他转换得到

**scala的元组定义之后可变吗**

映射是K/V对偶的集合，对偶是元组的最简单形式，元组可以装着多个不同类型的值

元组在Scala语言中是一个非常有用的容器对象。与列表一样,元组也是不可变的;但是与列表不同,元组可以包含不同类型的元素

**java注解底层是怎么实现的**

注解的底层也是使用反射实现的

**你写过springboot吗，他里面有哪些注解**

使用注解的优势：

     1.采用纯java代码，不在需要配置繁杂的xml文件

     2.在配置中也可享受面向对象带来的好处

     3.类型安全对重构可以提供良好的支持

     4.减少复杂配置文件的同时亦能享受到springIoC容器提供的功能

@SpringBootApplication：申明让spring boot自动给程序进行必要的配置，这个配置等同于：@Configuration ，@EnableAutoConfiguration 和 @ComponentScan 三个配置。

@ResponseBody：表示该方法的返回结果直接写入HTTP response body中，一般在异步获取数据时使用，用于构建RESTful的api。在使用@RequestMapping后，返回值通常解析为跳转路径，加上@esponsebody后返回结果不会被解析为跳转路径，而是直接写入HTTP response body中。比如异步获取json数据，加上@Responsebody后，会直接返回json数据。该注解一般会配合@RequestMapping一起使用。

@Controller：用于定义控制器类，在spring项目中由控制器负责将用户发来的URL请求转发到对应的服务接口（service层），一般这个注解在类中，通常方法需要配合注解@RequestMapping。

@RestController：用于标注控制层组件(如struts中的action)，@ResponseBody和@Controller的合集。

@RequestMapping：提供路由信息，负责URL到Controller中的具体函数的映射。

<https://blog.csdn.net/weixin_40753536/article/details/81285046>

**java日志框架用过哪个**

**Log4j**

**Slf4j**

Commons Logging

**java顶级父类有哪些方法**

[Java顶级父类-Object](http://www.baidu.com/link?url=Q0esTBr5tp3_SrYS34s5rZyt33ngT_e1ntGyLneKeFVtVEtT3IFpYUiKVX8vgtYFCmJXBg-UzTv9iBJXU7xaV_" \t "_blank)

protected Object clone() 创建并返回此对象的一个副本。   
boolean equals(Object obj) 指示某个其他对象是否与此对象“相等”。   
protected void finalize() 当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。   
Class<? extendsObject> getClass() 返回一个对象的运行时类。   
int hashCode() 返回该对象的哈希码值。   
void notify() 唤醒在此对象监视器上等待的单个线程。   
void notifyAll() 唤醒在此对象监视器上等待的所有线程。   
String toString() 返回该对象的字符串表示。   
void wait() 导致当前的线程等待，直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法。   
void wait(long timeout) 导致当前的线程等待，直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者超过指定的时间量。   
void wait(long timeout, int nanos) 导致当前的线程等待，直到其他线程调用此对象的 notify()你们公司的java书写规范

我们公司参考阿里巴巴开发规范进行代码书写。

**MySQL的执行计划**

Explain

**hashmap和treemap的区别知道吗**

HashMap 非线程安全 TreeMap 非线程安全

HashMap 是基于哈希表的 Map 接口的实现，TreeMap:基于红黑树实现

HashMap的结果是没有排序的，而TreeMap输出的结果是排好序的

HashMap通常比TreeMap快一点（树和哈希表的数据结构使然），建议多使用HashMap,在需要排序的Map时候才用TreeMap

**有哪些方法可以遍历一个hashmap**

大约有5种方式

//1.使用增强for 和 keySet()方法  
  
    //先使用Map接口中的keySet()方法  获取返回值set<k>对象，或者叫数组  
    Set<Integer> s=h.keySet();  
    //此时获得key值  
    //遍历使用增强for  
    for(Integer i:s)  
    {          
        //i为key  
        //使用Map接口中的get(key) 方法，返回值为对应的value  
        system.out.println(h.get(i));  
    }  
  
//2.使用增强for和Iterator，keyset遍历          
    //先使用Map接口中的keySet()方法  获取返回值set<k>对象，或者叫数组  
    Set<Integer> s=h.keySet();  
    //使用Iterator接口中的iterator()方法，返回值为Iterator<E>对象,set继承了Collection  
    //所以拥有iterator()方法，可以直接使用  
    Iterator i=s.iterator();  
    //使用增强for遍历  
    while(i.hasnext())  
    {  
        system.out.println(s.get(i.next()));  
    }  
  
//3.使用entrySet()和增强for  
        Set<Map.Entry<Integer, String>> s3=h.entrySet();  
        for(Map.Entry<Integer, String> m:s3)  
        {  
            System.out.println(m.getValue()+m.getKey());  
        }  
  
//4.使用entrySet()和Iterator  
        Set<Map.Entry<Integer, String>> s4=h.entrySet();  
        Iterator<Map.Entry<Integer, String>> i2=s4.iterator();  
        while(i2.hasNext())  
        {  
            Map.Entry<Integer, String> a=i2.next();  
            System.out.println(a.getValue()+a.getKey());  
        }  
  
//5.只能遍历到内容的方法  
        //Collection<String> c2=new ArrayList<String>();  
        Collection<String> c=h.values();  
        //ArrayList<String> a=(ArrayList<String>)c;  
          
        for(String s:c)  
        {  
            System.out.println(s);  
        }\*/

**list和arraylist的关系**

List是一个接口，而ArrayList是List接口的一个实现类。 ArrayList类继承并实现了List接口。 因此，List接口不能被构造，也就是我们说的不能创建实例对象，但是我们可以像下面那样为List接口创建一个指向自己的对象引用，而ArrayList实现类的实例对象就在这充当了这个指向List接口的对象引用。

List a=new ArrayList();  
则a拥有List的所有属性和方法，不会拥有其实现类ArrayList的独有的属性和方法。 如果List与ArrayList中有相同的属性(如int i),有相同的方法(如void f()), 则a.i是调用了List中的i ,a.f()是调用了ArrayList中的f();

**hashmap是怎么get和put的，就是它怎么是实现的**

get方法调用

1.当调用get方法时会调用hash函数，这个hash函数会将key的hashCode值返回，返回的hashCode与Entry数组长度-1进行逻辑与运算得到一个index值，用这个index值来确定数据存储在Entry数组当中的位置

2.通过循环来遍历索引位置对应的链表，初始值为数据存储在Entry数组当中的位置，循环条件为Entry对象不为null，改变循环条件为Entry对象的下一个节点

3.如果hash函数得到的hash值与Entry对象当中key的hash值相等，并且Entry对象当中的key值与get方法传进来的key值equals相同则返回该Entry对象的value值，否则返回null

put方法调用

1.调用hash函数得到key的HashCode值

2.通过HashCode值与数组长度-1逻辑与运算得到一个index值

3.遍历索引位置对应的链表，如果Entry对象的hash值与hash函数得到的hash值相等，并且该Entry对象的key值与put方法传过来的key值相等则，将该Entry对象的value值赋给一个变量，将该Entry对象的value值重新设置为put方法传过来的value值。将旧的value返回。

4.添加Entry对象到相应的索引位置

<https://blog.csdn.net/linsongbin1/article/details/54667453>