## 题目1：给定a，b两个文件，各存放50亿个url，每个url长度为1-255字节，内存限制是4g，让你找出a，b文件共同的url，说明思路和执行方法。

1. 问题分析：

考官主要想针对学员的海量文件处理的思路进行考核。对一些业务场景提出疑问，考验学生的处理问题的能力。学员应考虑到“分而治之”思想来处理文件。

1. 核心答案讲解：

可以估计每个文件安的大小为5G×64=320G，远远大于内存限制的4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

遍历文件a，对每个url求取hash(url)%1000，然后根据所取得的值将url分别存储到1000个小文件（记为a0,a1,...,a999）中。这样每个小文件的大约为300M。

遍历文件b，采取和a相同的方式将url分别存储到1000小文件（记为b0,b1,...,b999）。这样处理后，所有可能相同的url都在对应的小文件（a0vsb0,a1vsb1,...,a999vsb999）中，不对应的小文件不可能有相同的url。然后我们只要求出1000对小文件中相同的url即可。

　　求每对小文件中相同的url时，可以把其中一个小文件的url存储到hash\_set中。然后遍历另一个小文件的每个url，看其是否在刚才构建的hash\_set中，如果是，那么就是共同的url，存到文件里面就可以了。

1. 问题扩展

如果允许有一定的错误率，可以使用Bloom filter，4G内存大概可以表示340亿bit。将其中一个文件中的url使用Bloom filter映射为这340亿bit，然后挨个读取另外一个文件的url，检查是否与Bloom filter，如果是，那么该url应该是共同的url（注意会有一定的错误率）。

1. 结合项目中使用

在大数据离线统计中，MR的读写过程的“分而治之”的思想的理解。

在项目中读取HDFS上文件进行统计的时候的方法，可以使用的方法。

## 题目2：二叉树前序、中序、后续遍历方式（递归以及非递归）。

1. 问题分析：’

考官主要考察的是学员对于基础的树结构的掌握程度。其中树里面最基础的二叉的一个遍历的一个考核，学员需要能掌握手写其代码，理解其遍历的思想，以及二叉树的实现。

1. 核心答案讲解：

//代码在二叉树遍历代码

递归：

1、先序遍历（先遍历根节点，在遍历左节点，最后遍历右节点）

2、中序遍历（先遍历左节点，在遍历中节点，最后遍历右节点）

3、后序遍历（先遍历左节点，在遍历右节点，最后遍历根节点）

非递归 （非递归遍历二叉树必须考虑到栈的应用）

1. 先序遍历

a）访问之节点，并把结点node入栈。当前结点置为左孩子；

b）推断结点node是否为空，若为空。则取出栈顶结点并出栈，将右孩子置为当前结点；否则反复a）步直到当前结点为空或者栈为空（能够发现栈中的结点就是为了访问右孩子才存储的）

1. 中序排列

同样的道理。仅仅只是訪问的顺序移到出栈时

1. 后续排列

对于任一结点P，将其入栈，然后沿其左子树一直往下搜索。直到搜索到没有左孩子的结点，此时该结点出如今栈顶，可是此时不能将其出栈并訪问，因此其右孩子还为被訪问。

所以接下来依照同样的规则对其右子树进行同样的处理，当訪问完其右孩子时。该结点又出如今栈顶，此时能够将其出栈并訪问。这样就保证了正确的訪问顺序。能够看出，在这个过程中，每一个结点都两次出如今栈顶，仅仅有在第二次出如今栈顶时，才干訪问它。因此须要多设置一个变量标识该结点是否是第一次出如今栈顶。

1. 问题扩展

1、其中广度优先遍历和深度优先遍历（前、中、后序遍历）。

2、可以向学员扩展一下什么是树？树的特点有哪些？

（英文Tree）：它是一种抽象数据类型（ADT）或是实作这种抽象数据类型的数据结构，用来模拟具有树状结构性质的数据集合。它是由n（n>=1）个有限节点组成一个具有层次关系的集合。把他叫做“树”是因为它看起来像一颗倒挂的树，也就是树根朝上，而树叶向下的，它具有以下的特点：

  1、每个节点有零个或多个子节点：

  2、没有父节点的节点称为根节点：

  3、每一个非根节点有且只有一个父节点

  4、除了根节点外，每个子节点可以分为多个不相交的子树：

3、树的种类有哪些？

无序树、有序树（二叉树、完全二叉树、满二叉树、平衡二叉树、排序二叉树等）

1. 结合项目中使用

先序遍历：在第一次遍历到节点时就执行操作，一般只是想遍历执行操作（或输出结果）可选用先序遍历；

中序遍历：对于二分搜索树，中序遍历的操作顺序（或输出结果顺序）是符合从小到大（或从大到小）顺序的，故要遍历输出排序好的结果需要使用中序遍历

后序遍历：后续遍历的特点是执行操作时，肯定已经遍历过该节点的左右子节点，故适用于要进行破坏性操作的情况，比如删除所有节点

## 题目3：求数组所有可能的子数组？

1. 问题分析：

考官主要相对数据结构与算法基础的考核，如数组数据结构以及需要遍历数组内所有元素这类需求的常见处理方法和快速反应能力。这道题的解题思路可以参考二叉树先序遍历

1. 核心答案讲解：

处理这里需求常见的是递归法

递归法 ：假设集合为集合A，从集合A的每个元素自身分析，它只有两种状态，或是某个子集的元素，或是不属于任何子集，所以求子集的过程就可以看成对每个元素进行“取舍”的过程。上图中，根结点是初始状态，叶子结点是终结状态，该状态下的8个叶子结点就表示集合A的8个子集。第i层(i=1,2,3…n)表示已对前面i-1层做了取舍，所以这里可以用递归了。整个过程其实就是对二叉树的先序遍历。

1. 问题扩展

递归法是比较常见的处理此类问题方法，然而还有另一种按位对应法可以解决此类问题

按位对应法 ：

集合A={a,b,c},对于任意一个元素，在每个子集中，要么存在，要么不存在。

映射为子集： (a,b,c)

(1,1,1)->(a,b,c)

(1,1,0)->(a,b)

(1,0,1)->(a,c)

(1,0,0)->(a)

(0,1,1)->(b,c)

(0,1,0)->(b)

(0,0,1)->(c)

(0,0,0)->@(@表示空集)

与计算机中数据存储方式相似，故可以通过一个整型数与集合映射00…00 ~ 11…11（1表示有，0表示无，反之亦可），通过该整型数逐次增可遍历获取所有的数，即获取集合的相应子集。

1. 结合项目中使用

在程序中展现所有组合的可能性时，可以使用该算法。

## 题目4：给定一个数和一个有序数组，求有序数组的两个数的和满足这个数？（已知这两数存在）

1. 问题分析：考官主要相对数据结构与算法基础的考核，看到这道题目的时候就可以想到这道题肯定是跟遍历有关系的。
2. 核心答案讲解：

此题可以借鉴快速排序的思路，从数组的两边同时遍历数组

有一个 一个有序的整形数组，给定一个数，在数组中找出两个数的和等于这个数，并打印出来

第一种方法：将有序数组的最小值与最大值进行相加后，与给定数进行比较，如果想等，则在数组中找出两个数之和等与给定数；如果小于给定数，则从小的位置向后移，在进行比较；如果大于给定数，则从大的位置向前移，在进行比较。

1. 问题扩展

如何求取这种算法的时间复杂度呢？

因为该程序无论如何，只会遍历一遍集合，所以该程序的时间复杂度为O（n）

1. 结合项目中使用

在程序中查找特定组合时，可以参考使用这种算法

## 题目5：string、stringbulider、stringbuffer的区别

1. 问题分析：

考官考核的主要是面试人员对 这三个类 的作用 和 特点上的理解是否透彻

1. 核心答案讲解：

1.String 是字符串常量而StringBulider和StringBuffer 的字符串变量

顾名思义String对象一旦创建就不能更改，而另外两个可以更  改。注意关于String创建完对象进行赋值，之后还是可以再次赋值，String  s=a;  s=a+"b";看似是更改了对象的值其实在此处是又创建了一个新的对象值为ab,所以原有的对象并没有进行更改，而StringBuffer和StringBulider是直接在原有的对象的基础上进行更改的。

2.运行速度：StringBulider>StringBuffer>String

    正是因为String在对值进行更改时是重新创建一个新的对象，原有的对象被当作垃圾进行回收，所以是最慢的。

3.线程安全：StringBulider是线程不安全的，StringBuffer是线程安全的。

如果一个StringBuffer对象在字符串缓冲区被多个线程使用时，StringBuffer中很多方法可以带有synchronized关键字，所以可以保证线程是安全的，但StringBuilder的方法则没有该关键字，所以不能保证线程安全，有可能会出现一些错误的操作。所以如果要进行的操作是多线程的，那么就要使用StringBuffer，但是在单线程的情况下，还是建议使用速度比较快的StringBuilder。

1. 问题扩展

String：适用于少量的字符串操作的情况

StringBuilder：适用于单线程下在字符缓冲区进行大量操作的情况

StringBuffer：适用多线程下在字符缓冲区进行大量操作的情况

1. 结合项目中使用

1. String：它被用于裁剪，拼接。(当然如果拼接过多的话还是建议用StringBuffer,或者StringBuild)搜索字符串，比较字符串，截取字符串，转换大小写等。在项目中不经常发生变化的业务场景中，优先使用String

2. StringBuffer：用于拼接，替换，删除。在项目多线程环境下运行，如:XML解析，HTTP参数解析与封装等。

3. StringBuilder:它同StringBuffer使用方式一样，不过在项目中使用的地方建议是单线程的环境下，如：SQL拼接，JSON封装等

## 题6: ArrayList、LinkedList、Vector区别

(1)问题分析：

考官主要考核面试人员对Java基础 集合的了解程度,以此判断面试人员Java基础是否扎实

(2)核心答案讲解：

ArrayList

ArrayList是最常用的List实现类，内部是通过数组实现的，它允许对元素进行快速随机访问。数组的缺点是每个元素之间不能有间隔，当数组大小不满足时需要增加存储能力，就要讲已经有数组的数据复制到新的存储空间中。当从ArrayList的中间位置插入或者删除元素时，需要对数组进行复制、移动、代价比较高。因此，它适合随机查找和遍历，不适合插入和删除

Vector

Vector与ArrayList一样，也是通过数组实现的，不同的是它支持线程的同步，即某一时刻只有一个线程能够写Vector，避免多线程同时写而引起的不一致性，但实现同步需要很高的花费，因此，访问它比访问ArrayList慢。

LinkedList

LinkedList是用链表结构存储数据的，很适合数据的动态插入和删除，随机访问和遍历速度比较慢。另外，他还提供了List接口中没有定义的方法，专门用于操作表头和表尾元素，可以当作堆栈、队列和双向队列使用。

(3)问题扩展

1) ArrayList在内存不够时默认是扩展50% + 1个，Vector是默认扩展1倍。

2) Vector提供indexOf(obj, start)接口，ArrayList没有。

3) Vector属于线程安全级别的，但是大多数情况下不使用Vector，因为线程安全需要更大的系统开销。

(4)结合项目中使用

项目中的使用场景

1) 对于需要快速插入，删除元素，应该使用LinkedList。

2) 对于需要快速随机访问元素，应该使用ArrayList。

3) 对于“单线程环境” 或者 “多线程环境，但List仅仅只会被单个线程操作”，此时应该使用非同步的类(如ArrayList)。

对于“多线程环境，且List可能同时被多个线程操作”，此时，应该使用同步的类Vector。

## 题7: HashMap 时间复杂度？

(1)问题分析：

考官主要考核 面试人员 Java基础数据类型的相关算法的掌握程度

(2)核心答案讲解：

hashmap时间复杂度, 理想情况下HashMap的时间复杂度为O（1）

因为对于内存来说，访问任何地址的时间是一样的，即时间极短，相当于可以同时访问到所有地址。而在时间复杂度为O（1）时，需要很大的内存空间，所以必须要对内存和时间进行取舍。

(3)问题扩展:

如何在空间 和 时间上做出最有选择

hashmap在初始化的时候有俩值，初始大小和负载因子，默认是16和0.75

大小不用解释，负载因子就是一个小数，扩容的时候用的，在put的时候，hashmap就检查一下自己的大小，是不是大于16x0.75（大小x负载因子），如果是，就扩容，扩容到2^n.

hash值要进行取余运算，而数学证明，hash%(2^n)=hash^(2^n-1)，为了效率，就采用了2倍的扩容。ps：^为异或，2^n为2的n次方

(4)结合项目中使用:

非并发场景使用HashMap，并发场景可以使用Hashtable，但是推荐使用ConcurrentHashMap（锁粒度更低、效率更高）。

另外使用在使用HashMap时要注意null值的判断，

Hashtable也要注意防止put null key和 null value。

HashMap提供对key的Set进行遍历，因此它是fail-fast的，但HashTable提供对key的Enumeration进行遍历，它不支持fail-fast

## 题8: 求一个数组的第二大值？

(1)问题分析：

这道题很明显就是考察面试者最基础的算法能力，思考的话就往排序的方面想就好了，如果能想到多种方法解决问题的话，会显示出自己的基础深厚，让面试官对你印象更好

(2)核心答案讲解：

用两个变量max,max2,其中max储存最大值,max2储存第二大值

初始化的时候,将数组中的第一个元素中较大的存进max中,较小的存进max2中

然后从第三个元素(下标为2)的元素开始,如果遇到的数比max大,就让max2=max;max等于遇到的数

一直循环,直到数组尾部

最后输出max2

(3)问题扩展:

排序算法是非常基础却又是各家面试经常会遇到的问题，面试前应该对9大排序算法应该有比较初级的了解，至少能够手写三种算法知道六种算法的名字

常见问题

你了解的排序算法大概有几种，如何实现 ？

冒泡排序，插入排序，选择排序，归并排序，快速排序，希尔排序，堆排序等

1. 项目应用:

排序算法在项目中使用比较广泛，比如做排行榜，无序数据按照某种规格进行排序等，使用好可以大幅度减少任务执行时间

## 题目9：有一个1G大小的一个文件，里面每一行是一个词，词的大小不超过16字节，内存限制大小是1M，要求返回频数最高的100个词。

1. 问题分析：

考官主要想针对学员的真实业务逻辑进行考核。对一些业务场景提出疑问，考验学生的处理问题的能力。分而治之 + hash统计 + 堆/快速排序。

1. 核心答案讲解：

顺序读文件中，对于每个词x，取hash(x)%5000，然后按照该值存到5000个小文件（记为x0,x1,...x4999）中。这样每个文件大概是200k左右。  
如果其中的有的文件超过了1M大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过1M。 对每个小文件，统计每个文件中出现的词以及相应的频率（可以采用trie树/hash\_map等），并取出出现频率最大的100个词（可以用含100个结点的最小堆），并把100个词及相应的频率存入文件，这样又得到了5000个文件。下一步就是把这5000个文件进行归并（类似与归并排序）的过程了。

1. 问题扩展

此题涉及到分而治之的思想以及归并排序等基础的算法思想，结合着归并排序的分而治之的思想，将学员的基础算法能力加强。

1. 结合项目中使用

在大数据离线统计中，MR的读写过程的“分而治之”的思想的理解。

在项目中读取HDFS上文件进行统计的时候的方法，可以使用的方法。

## 题目10：现有海量日志数据保存在一个超级大的文件中，该文件无法直接读入内存，要求从中提取某天出访问百度次数最多的那个IP。

1. 问题分析：

考官主要想针对学员的真实业务逻辑进行考核。对一些业务场景提出疑问，考验学生的处理问题的能力。分而治之 + hash统计 。

1. 核心答案讲解：

首先是这一天，并且是访问百度的日志中的IP取出来，逐个写入到一个大文件中。注意到IP是32位的，最多有个2^32个IP。同样可以采用映射的方法，比如模1000，把整个大文件映射为1000个小文件，再找出每个小文中出现频率最大的IP（可以采用hash\_map进行频率统计，然后再找出频率最大的几个）及相应的频率。然后再在这1000个最大的IP中，找出那个频率最大的IP，即为所求。

1. 问题扩展

算法思想：分而治之+Hash

1.IP地址最多有2^32=4G种取值情况，所以不能完全加载到内存中处理；

2.可以考虑采用“分而治之”的思想，按照IP地址的Hash(IP)%1024值，把海量IP日志分别存储到1024个小文件中。这样，每个小文件最多包含4MB个IP地址；

3.对于每一个小文件，可以构建一个IP为key，出现次数为value的Hash map，同时记录当前出现次数最多的那个IP地址；

4.可以得到1024个小文件中的出现次数最多的IP，再依据常规的排序算法得到总体上出现次数最多的IP；。

（4） 结合项目中使用

在大数据离线统计中，MR的读写过程的“分而治之”的思想的理解。

在项目中读取HDFS上文件进行统计的时候的方法，可以使用的方法。

## 题目11：手写TopN算法，使用spark，scala实现。

1. 问题分析：

面试官主要考察的是学员对于大数据真实项目业务中一个基础的需求实现的思路。以及该学员是否有一定的代码书写能力。

1. 核心答案讲解：

在系统中，我们经常会遇到这样的需求：将大量（比如几十万、甚至上百万）的对象进行排序，然后 只需要取出最Top的前N名作为排行榜的数据，这即是一个TopN算法。常见的解决方案有三种：

（1）直接使用List的Sort方法进行处理。

（2）使用排序二叉树进行排序，然后取出前N名。

（3）使用最大堆排序，然后取出前N名。

第一种方案的性能是最差的，后两种方案性能会好一些，但是还是不能满足我们的需求。最主要的原 因在于使用二叉树和最大堆排序时，都是对所有的对象进行排序，而不是将代价花费在我们需要的少数的 TopN上。

思路：

使用一个长度为N的数组，来存放最Top的N个对象，越Top的对象其在数组中的Index就 越小。这样，每次加入一个对象时，就与Index最大的那个对象比较，如果比其更Top，则交换两个对象的 位置。如果被交换的对象是数组中的最后一个对象（Index最大），则该对象会被抛弃。如此，可以保证 容器中始终保持的都是最Top的N个对象。

1. 问题扩展

在实现TopN算法的时候，我们会使用到好几种方式来实现，比如使用排序二叉树、最大堆排序等这些算法来实现，而这些排序也是其中面试官在面试的时候的一个考点，所以可以将这些排序都领着学员过一下，以达到复习，或者是学习的目的。

1. 结合项目中使用：

在大数据离线统计中，会不时的有需求根据业务求出本月销售的前TOPN的数据。有的会需要使用SQL实现，我们也可以根据这个思路，使用代码来实现。比如说在电商领域，求上个月卖的最好的前10个商品，或者是每个品类下卖的最好的前10个商品。

。

## 题目12：如何使用redis，或者zookeeper实现分布式锁

1. 问题分析：’

考官主要考察的是学员对于JAVA基础中分布式锁的一个理解，学员需要讲解出来 锁的概念以及锁的作用，还有锁的实现方式有哪些。

1. 核心答案讲解：

在项目中，部署了多个tomcat应用，在执行定时任务时就会遇到同一任务可能执行多次的情况，我们可以借助分布式锁，保证在同一时间只有一个tomcat应用执行了定时任务。

分布式锁的实现方式有四种：

1. 使用redis的setnx()和expire()

setnx(key,value) 如果key不存在，设置为当前key的值为value;如果key存在，直接返回。expire()来设置超时时间

1. 使用redis的getset()

此方法使redisTemplate.boundValueOps(key).getAndSet(value)的方法，如果返回空，表示获取了分布式锁；如果返回不为空，表示分布式锁已经被其他程序占用。

1. 使用zookeeper的创建节点node。

使用zookeeper创建节点node，如果创建节点成功，表示获取了此分布式锁；如果创建节点失败，表示此分布式锁已经被其他程序占用(多个程序同时创建一个节点node，只有一个能够创建成功)

1. 使用zookeeper的创建临时序列节点。

使用zookeeper创建临时序列节点来实现分布式锁，适用于顺序执行的程序，大体思路就是创建临时序列节点，找出最小的序列节点，获取分布式锁，程 序执行完成之后此序列节点消失，通过watch来监控节点的变化，从剩下的节点的找到最小的序列节点，获取分布式锁，执行相应处理，依次类推

1. 问题扩展

在构建分布式锁的时候，我们使用到了redis这个数据库来构建，而redis同样是我们面试中的一个重点的数据库知识，可以讲解一下redis的一个基本的数据格式，还有哨兵模式等一些知识点。

在使用redis构建分布式锁的时候，我们使用到了aop切面编程的思想。可以普及一下面向切面编程。

1. 结合项目中使用

在项目中，部署了多个tomcat应用，在执行定时任务时就会遇到同一任务可能执行多次的情况，我们可以借助分布式锁，保证在同一时间只有一个tomcat应用执行了定时任务。

## 题目13：数组和链表的区别，能否用伪代码实现链表。

1. 问题分析：

面试官主要考察的是学员对于基础的数据结构中的数组以及链表的一个区别，考察学员对于基础的掌握程度，以及手写伪代码是为了考察学生是是否真正的掌握 。

1. 核心答案讲解：

数组是将元素在内存中连续存放，由于每个元素占用内存相同，可以通过下标迅速访问数组中任何元素。但是如果要在数组中增加一个元素，需要移动大量元素，在内存中空出一个元素的空间，然后将要增加的元素放在其中。同样的道理，如果想删除一个元素，同样需要移动大量元素去填掉被移动的元素。如果应用需要快速访问数据，很少或不插入和删除元素，就应该用数组。

链表恰好相反，链表中的元素在内存中不是顺序存储的，而是通过存在元素中的指针联系到一起。比如：上一个元素有个指针指到下一个元素，以此类推，直到最后一个元素。如果要访问链表中一个元素，需要从第一个元素开始，一直找到需要的元素位置。但是增加和删除一个元素对于链表数据结构就非常简单了，只要修改元素中的指针就可以了。如果应用需要经常插入和删除元素你就需要用链表数据结构了。

1. 问题扩展

在这个问题中，我们介绍完区别以后，可以对于数组和链表的特性进行一个介绍，比如对于数组和列表之间的扩容的区别进行简单的介绍。不要仅仅局限于回答一个结构上的

(1) 从逻辑结构角度来看

a： 数组必须事先定义固定的长度（元素个数），不能适应数据动态地增减的情况。当数据增加时，可能超出原先定义的元素个数；当数据减少时，造成内存浪费。

b：链表动态地进行存储分配，可以适应数据动态地增减的情况，且可以方便地插入、删除数据项。（数组中插入、删除数据项时，需要移动其它数据项）

　　(2)从内存存储角度来看

　　 a：(静态)数组从栈中分配空间, 对于程序员方便快速,但自由度小。

b：链表从堆中分配空间, 自由度大但申请管理比较麻烦.区别。结合应用回答区别。

1. 结合项目中使用：

在编写我们的业务代码的时候，时长会用到数组链表这样的数据结构，如果运用的熟练，可以对自己的代码的效率进行调优。

## 题目14：有了解过哪些机器学习的算法？

1. 问题分析：

考官主要考察的是学员是否对于人工智能方面的感兴趣，是否有空闲时间了解过机器学习算法这方面的知识，有则更好。

1. 核心答案讲解：

学员需要根据自己的实际情况去回答，学员也可以自己课下空余的时间去掌握一两常用的机器学习算法，以此来增加自己的筹码：

PCA思想：

PCA顾名思义，就是找出数据里最主要的方面，用数据里最主要的方面来代替原始数据。具体的，假如我们的数据集是n维的，共有m个数据(x(1),x(2),...,x(m))。我们希望将这m个数据的维度从n维降到n'维，希望这m个n'维的数据集尽可能的代表原始数据集。我们知道数据从n维降到n'维肯定会有损失，但是我们希望损失尽可能的小。

PCA的算法步骤：

设有m条n维数据。

1）将原始数据按列组成n行m列矩阵X

2）将X的每一行（代表一个属性字段）进行零均值化，即减去这一行的均值

3）求出协方差矩阵

4）求出协方差矩阵的特征值及对应的特征向量

5）将特征向量按对应特征值大小从上到下按行排列成矩阵，取前k行组成矩阵P

6）即为降维到k维后的数据

1. 问题扩展

在算法的推导过程中，会大量的使用到数学的推导公式，而数学思想对于程序员来说是必不可缺的一种思想，可以在讲解的时候，概括性的给学员讲一下数学的相关知识。

1. 结合项目中使用

此问题在项目中是没有什么体现的，面试官就是想考察一下面试者本身具有的学习能力，以此来评估该面试者的可培养价值，以及潜力。但是机器学习算法在推荐系统项目中、以后数据挖掘工作中会起到至关重要的作用。

## 题目15：协同过滤算法的底层实现是什么？

1. 问题分析：

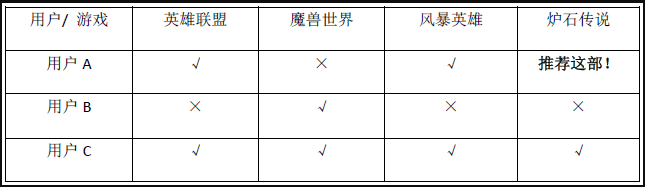
面试官主要是对于写了推荐系统项目，并且在推荐系统中使用到基于物品的协同过滤算法和基于用户的协同过滤算法的学员进行考核，通过对项目中算法的原理，以及细节来考证项目的可行性。

1. 核心答案讲解：

基于用户的CF基于用户的协同过滤，

通过用户对不同内容（物品）的行为，来评测用户之间的相似性，找到“邻居”，基于这种相似性做出推荐。这种推荐的本质是，给相似的用户推荐其他用户喜欢的内容，这就是我们经常看到的：和你类似的人还喜欢如下内容。下面这个列子可以说明：、

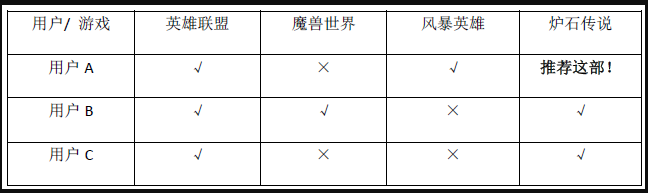
需要给用户A推荐游戏，根据用户B和用户C对游戏的偏好行为，给A推荐游戏，从下表可以知道，基于对游戏的偏好来讲，用户A跟用户C的相似度比用户跟用户B的相似度要大，所以，系统会给用户A推荐炉石传说。当然，举的这个例子十分简单，实际上，还需要考虑的是每个用户物品的偏好程度，虽然用户A和用户C都玩过英雄联盟，但是用户A和用户C对英雄联盟的偏好程度可能不一样，在真正的计算过程中，需要对这种偏好的程度设定一个参数，参数的大小表明用户对物品的偏好程度的大小。根据设置或调整参数的大小，得出最后的值给用户推荐商品，这样的推荐计算结果会更加严谨。



基于物品的CF的原理和基于用户的CF类似，

只是在计算邻居时采用物品本身，而不是从用户的角度，即基于用户对物品的偏好找到相似的物品，然后根据用户的历史偏好，推荐相似的物品给用户。从计算的角度来看，就是将所有用户对某个物品的偏好作为一个向量来计算物品之间的相似度，得到物品的相似物品后，根据用户历史的偏好预测当前用户还没有表示偏好的物品，计算得到一个排序的物品列表作为推荐。就是我们常见的：购买该商品的用户还购买了如下商品，等等就是文章开头前的啤酒和纸尿裤的故事，

因为超市的人员发现很多男人买纸尿裤的时候会买啤酒，根据这一用户行为，纸尿裤和啤酒的相似度较高，那么在用户购买纸尿裤的时候推荐啤酒，增加啤酒的销量。也用相同的例子来说明：可以从下表看出，用户B和用户C有一个共同的特征，即选择了英雄联盟也会选炉石传说，说明这两个游戏之间相似度会比较高，那么会当用户A选择了英雄联盟，系统会把炉石传说也推荐给他。但是同时要注意的一点，这种情况也是属于比较理想化的一种，物品和物品之间的相似度可能不一样，也需要调整参数，这里不进行深入的探讨了。



1. 问题扩展

该算法的在实现的过程中，在计算两者的相似度的时候，使用了数学中的：“欧几里德距离”，“皮尔逊相关系数”等数学相关的知识。同时也可以进行两者之间的使用的一个对比回答，回答面试官为什么我们选择这种，而不选择另外的一种。在非社交网络的网站中，内容内在的联系是很重要的推荐原则，它比基于相似用户的推荐原则更加有效。比如在购书网站上，当你看一本书的时候，推荐引擎会给你推荐相关的书籍，这个推荐的重要性远远超过了网站首页对该用户的综合推荐。可以看到，在这种情况下，Item CF 的推荐成为了引导用户浏览的重要手段。同时 Item CF 便于为推荐做出解释，在一个非社交网络的网站中，给某个用户推荐一本书，同时给出的解释是某某和你有相似兴趣的人也看了这本书，这很难让用户信服，因为用户可能根本不认识那个人；但如果解释说是因为这本书和你以前看的某本书相似，用户可能就觉得合理而采纳了此推荐。

相反的，在现今很流行的社交网络站点中，User CF 是一个更不错的选择，User CF 加上社会网络信息，可以增加用户对推荐解释的信服程度。

1. 结合项目中使用：

使用在大数据的推荐中的项目，项目的核心算法。

## 题目16：快速排序、归并排序、冒泡排序、选择排序（复杂度分别是多少）。

1. 问题分析：

面试官主要考察的是学员对于基础算法的掌握程度，其中冒泡排序和选择排序都属于简单的排序过程，而快速排序和归并排序是属于高级排序，而面试官问时间复杂度，这就需要学员对算法到掌握的程度，能够计算算法的时间复杂度。

1. 核心答案讲解：

时间复杂度我们使用大O表示法进行表示，

(1)冒泡排序的时间复杂度：

1.比较相邻的元素，如果第一个比第二个大（升序），就交换他们两个

2.对每一对相邻的元素做同样的工作，从开始到结尾的最后一对

这步做完后，最后的元素会是最大的数

3.针对所有的元素重复以上的步骤，除了最后一个

4.持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较

5.稳定性：数值相同的元素在排序中不交换位置为稳定反之为不稳定

6.最优复杂度：O(n) 最坏复杂度：O(n^2) 稳定性：稳定

(2)快速排序的时间复杂度：

通过一趟排序将要排序的数据分割成独立的两部分其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序。整个排序过程可以递归进行，以此达到整个数据变成有序序列

过程描述：

1、定义一个标杆，将列表的第一个元素存入标杆，列表的第一个位置相当于空出来了

2、定义两个指针low,high，分别记录列表头和列表尾的位置

3、判断low是不是小于high，列表尾high位置的元素是不是大于等于标杆，如果是high向左移动一位

重复执行这步判断，当条件不成立时，让low位置存放当前high位置的元素

4、判断low是不是小于high，列表头low位置的元素是不是小于标杆，如果是low向右移动一位

重复执行这步判断，当条件不成立时，让high位置存放当前low位置的元素

5、让4、5两步交替执行，直到low >= high时退出循环。此时low = high

6、此时找到的low的位置就是标杆的位置，将记录的标杆元素，放置在这个位置。

7、递归执行上述步骤直到low >= high时退出。

8、最优时间复杂度计算：n拆分多少次能只剩一个元素，也就是n除以多少次2能等于1

即n/2^x = 1 --> 2^x = n --> x = logn (log以2为底，n的对数)

每次让列表中的值沿着初始值分为左右两半的过程的时间复杂度为n

所以最优时间复杂度为nlogn

9、最优复杂度：O(nlogn) 最坏复杂度：O(n^2) 稳定性：不稳定

（3）快速排序的时间复杂度

1、归并排序的思想是先递归分解数组，再合并数组

2、将数组分解最小之后，然后合并两个有序数组。

3、基本思路就是比较两个数组最前面的数，谁小就取谁，取了后相应的指针就往后移一位。

4、然后再比较，直至一个数组为空，最后把两一个数组的剩余部分复制过来即可。

1. 最优复杂度：O(nlogn) 最坏复杂度：O(nlogn) 稳定性：稳定

（4）选择排序时间复杂度：

1、始终从未排序的序列中找到最小的放到最前面

2、将第一个元素先作为最小值，用第一个元素和后面的元素依次比较

3、首次碰见比第一个元素更小的元素就记录这个“较小元素”的位置

4、继续比较如果碰见比“较小元素”还更小的元素，就将记录的“较小元素”的位置信息，替换成“更小元素”的位置信息

5、直到比较完整个序列，最小的元素位置信息就被记录下来了

6、交换第一个元素和最小元素的位置，序列的头部就变成了最小的元素

7、再将第二个元素先作为序列剩余元素的最小元素，和剩下的元素重复上述步骤进行比较

8、将第二小的元素找到，并和第二个元素进行交换

9、多次重复上述步骤n-1次，即可得到升序序列

10、最优复杂度：O(n^2) 最坏复杂度：O(n^2) 稳定性：不稳定

1. 问题扩展

在讲解这些算法的时候，我们在上一步都讲解了算法的执行原理，那么在其中在对时间复杂度计数的时候，我们使用到了大O计数法，课扩展大O时间计算法，以及时间复杂度是什么。

1. 结合项目中使用：

在任何项目用代码实现的时候，都会涉及到数据结构，有数据结构就会涉及到效率的问题，我们使用 大O表示法来表示效率。

## 题目17：spark-submit几种提交模式的 区别是什么？

1. 问题分析：

面试官主要考察的是学员对于spark知识点中的spark-submit进行考核，第一考核学员是否真正在线上提交过spark程序，第二考核学员对spark提交模式具体的了解多少。学员介绍的时候介绍有哪几种模式，然后分别介绍一下该模式的流程是什么，最后讲一下区别。

1. 核心答案讲解：

在 spark的提交模式中，有三种提交方式：分别是基于spark集群的standalone模式，基于YARN集群的yarn-client和yarn-cluster三种模式，三者各有好坏处：

Standalone模式运行过程：

1、我们提交一个任务，任务就叫Application

2、初始化程序的入口SparkContext，

　　2.1 初始化DAG Scheduler

　　2.2 初始化Task Scheduler

3、Task Scheduler向master去进行注册并申请资源（CPU Core和Memory）

4、Master根据SparkContext的资源申请要求和Worker心跳周期内报告的信息决定在哪个Worker上分配资源，然后在该Worker上获取资源，然后启动StandaloneExecutorBackend；顺便初始化好了一个线程池

5、StandaloneExecutorBackend向Driver(SparkContext)注册,这样Driver就知道哪些Executor为他进行服务了。 到这个时候其实我们的初始化过程基本完成了，我们开始执行transformation的代码，但是代码并不会真正的运行，直到我们遇到一个action操作。生产一个job任务，进行stage的划分

6、SparkContext将Applicaiton代码发送给StandaloneExecutorBackend；并且SparkContext解析Applicaiton代码，构建DAG图，并提交给DAG Scheduler分解成Stage（当碰到Action操作 时，就会催生Job；每个Job中含有1个或多个Stage，Stage一般在获取外部数据和shuffle之前产生）。

7、将Stage（或者称为TaskSet）提交给Task Scheduler。Task Scheduler负责将Task分配到相应的Worker，最后提交给StandaloneExecutorBackend执行；

8、对task进行序列化，并根据task的分配算法，分配task

9、对接收过来的task进行反序列化，把task封装成一个线程

10、开始执行Task，并向SparkContext报告，直至Task完成。

11、资源注销

Yarn-client模式运行过程：

1.Spark Yarn Client向YARN的ResourceManager申请启动Application Master。同时在SparkContent初始化中将创建DAGScheduler和TASKScheduler等，由于我们选择的是Yarn-Client模式，程序会选择YarnClientClusterScheduler和YarnClientSchedulerBackend；

2.ResourceManager收到请求后，在集群中选择一个NodeManager，为该应用程序分配第一个Container，要求它在这个Container中启动应用程序的ApplicationMaster，与YARN-Cluster区别的是在该ApplicationMaster不运行SparkContext，只与SparkContext进行联系进行资源的分派；

3.Client中的SparkContext初始化完毕后，与ApplicationMaster建立通讯，向ResourceManager注册，根据任务信息向ResourceManager申请资源（Container）；

4.一旦ApplicationMaster申请到资源（也就是Container）后，便与对应的NodeManager通信，要求它在获得的Container中启动启动CoarseGrainedExecutorBackend，CoarseGrainedExecutorBackend启动后会向Client中的SparkContext注册并申请Task；

5.Client中的SparkContext分配Task给CoarseGrainedExecutorBackend执行，CoarseGrainedExecutorBackend运行Task并向Driver汇报运行的状态和进度，以让Client随时掌握各个任务的运行状态，从而可以在任务失败时重新启动任务；

1. 应用程序运行完成后，Client的SparkContext向ResourceManager申请注销并关闭自己。

Yarn-cluster模式运行过程：

1. Spark Yarn Client向YARN中提交应用程序，包括ApplicationMaster程序、启动ApplicationMaster的命令、需要在Executor中运行的程序等；

2. ResourceManager收到请求后，在集群中选择一个NodeManager，为该应用程序分配第一个Container，要求它在这个Container中启动应用程序的ApplicationMaster，其中ApplicationMaster进行SparkContext等的初始化；

3. ApplicationMaster向ResourceManager注册，这样用户可以直接通过ResourceManage查看应用程序的运行状态，然后它将采用轮询的方式通过RPC协议为各个任务申请资源，并监控它们的运行状态直到运行结束；

4. 一旦ApplicationMaster申请到资源（也就是Container）后，便与对应的NodeManager通信，要求它在获得的Container中启动启动CoarseGrainedExecutorBackend，CoarseGrainedExecutorBackend启动后会向ApplicationMaster中的SparkContext注册并申请Task。这一点和Standalone模式一样，只不过SparkContext在Spark Application中初始化时，使用CoarseGrainedSchedulerBackend配合YarnClusterScheduler进行任务的调度，其中YarnClusterScheduler只是对TaskSchedulerImpl的一个简单包装，增加了对Executor的等待逻辑等；

5. ApplicationMaster中的SparkContext分配Task给CoarseGrainedExecutorBackend执行，CoarseGrainedExecutorBackend运行Task并向ApplicationMaster汇报运行的状态和进度，以让ApplicationMaster随时掌握各个任务的运行状态，从而可以在任务失败时重新启动任务；

1. 应用程序运行完成后，ApplicationMaster向ResourceManager申请注销并关闭自己。

Yarn-client和yarn-cluster的区别是什么？

理解YARN-Client和YARN-Cluster深层次的区别之前先清楚一个概念：Application Master。在YARN中，每个Application实例都有一个ApplicationMaster进程，它是Application启动的第一个容器。它负责和ResourceManager打交道并请求资源，获取资源之后告诉NodeManager为其启动Container。从深层次的含义讲YARN-Cluster和YARN-Client模式的区别其实就是ApplicationMaster进程的区别。

1、YARN-Cluster模式下，Driver运行在AM(Application Master)中，它负责向YARN申请资源，并监督作业的运行状况。当用户提交了作业之后，就可以关掉Client，作业会继续在YARN上运行，因而YARN-Cluster模式不适合运行交互类型的作业；

2、YARN-Client模式下，Application Master仅仅向YARN请求Executor，Client会和请求的Container通信来调度他们工作，也就是说Client不能离开。

1. 问题扩展

在提交程序jar包的时候，我们的使用到了yarn的一个资源调度的申请过程。

1. 结合项目中使用：

在spark项目中，我们写完spark的代码之后，就需要选择在什么样的模式下进行提交运行该jar包。

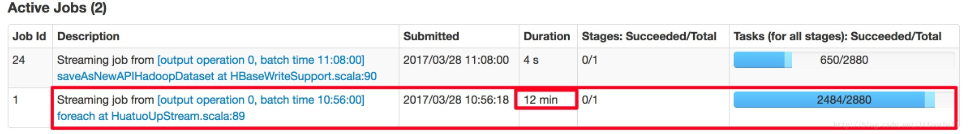
## 题目18：spark streming在实时处理时会发生什么故障，如何停止，解决。

1. 问题分析：

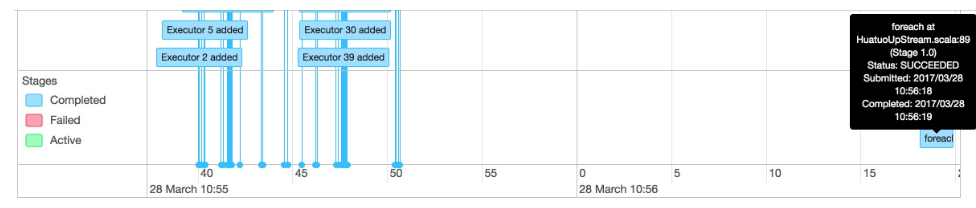
面试官主要考察的是面试者对于大数据真实项目中对spark集群的了解。以及当集群出现问题的时候的解决思路与方案。

1. 核心答案讲解：

在我们使用sparkStreaming执行的时候，出现的问题，偶尔会出现Stager停止不动的现象。



但是我们点击任务进去的时候！发现任务状态SUCCESSED的



然后我们去查看日志，发现日志中出现ERROR和WARN报出：

ERROR LiveListenerBus: Dropping SparkListenerEvent because no remaining room in event queue. This likely means one of the SparkListeners is too slow and cannot keep up with the rate at which tasks are being started by the scheduler.

WARN LiveListenerBus: Dropped 1 SparkListenerEvents since Thu Jan 01 08:00:00 CST 1970

分析问题的原因：

当消息队列中的消息数超过其spark.scheduler.listenerbus.eventqueue.size设置的数量(如果没有设置,默认为10000)时，会将最新的消息移除，这些消息本来是通知任务运行状态的，由于你移除了，状态无法得到更新，所以会出现上面描述的现象。

解决方案：

在spark-submit中添加参数。

--conf spark.scheduler.listenerbus.eventqueue.size=100000

问题解决。

1. 问题扩展

在spark中会出现大量的参数配置，一些参数配置不当，会造成程序效率低下。比如设置spark程序的缓存等

1. 结合项目中使用：

在大数据实时阶段，在我们集群中提交我们的编写的spark程序的时候，会出现错误。

。

## 题目19：spark工作机制。

1. 问题分析：

面试官主要考察的是学员对于大数据spark阶段的原理的掌握，主要考核学员对Spark的整体架构的实现。

1. 核心答案讲解：

在spark的核心架构机制中！有很多中机制，如Spark的容错机制、Shuffle机制等，都是属于Spark的核心机制。

容错机制：

DD之间的算子操作会形成DAG图，RDD之间的依赖关系会形成Lineage。

要理解Lineage机制首先要明确两种依赖的概念：

Shuffle Dependencies（宽依赖）

父分区可以被多个子分区所用 ，即多对多的关系

Narrow Dependencies（窄依赖）

父分区最多被一个子分区所用 ，即一对一或者多对一的关系

当出现某个节点计算错误的时候，会顺着RDD的操作顺序往回走。一旦是Narrow Dependencies错误，重新计算父RDD分区即可，因为其不依赖其他节点 ，而如果Shuffle Dependencies错误，重算代价较高，因为一旦重新计算其依赖的父RDD分区，会造成冗余计算

这时候就需要人为的添加检查点来提高容错机制的执行效率

什么情况下需要加CheckPoint

DAG中的Lineage过长，如果重算开销太大，故在特定几个Shuffle Dependencies上做CheckPoint是有价值的。

Checkpoint会产生磁盘开销，因为其就是将数据持久化到磁盘中，所以做检查点的RDD最好是已经在内存中缓存了。

1. 问题扩展

在整个工作机制中，不仅仅有容错机制的存在，还有Shuffle机制、i/o机制、调度机制等。

1. 结合项目中使用：

在我们去执行我们的spark程序的时候，当有成千的计算需要去执行的时候，适当的使用checkpoint机制，会很大程度提升我们的效率。

。

## 题目20：Kafka和sparkStreaming的整合，手动提交的offset调用了什么方法？

1. 问题分析：

面试官主要考察的是学员对于SparkStreaming+kafka的两个架构结合上面的知识点，考察到整合时候的具体步骤等，还有在具体使用该整合架构 的时候使用的API的考察。

1. 核心答案讲解：

在kafka于sparkStreaming整合的时候，其实是有两种提交方式，一种是自动提交offset一种就是手动提交offset。

在设置手动提交offset的时候，我们只需要在编写我们的程序的时候，

设置：properties.put("enable.auto.commit","false");程序将会手动提交我么你的offset。这样最大程度上保证我们的数据安全。

1. 问题扩展

不仅仅有手动提交，还有一种提交方式，自动提交。

enable.auto.commit 设置为false，如果是true，则这个消费者的偏移量会在后台自动提交，这样设置目的是为了后面自己提交offset，因为如果虽然获取到了消息，但是后面的转化操作并将结果写到如hive中并没有完成程序就挂了的话，这样是不能将这次的offset提交的，这样就可以等程序重启之后接着上次失败的地方继续消费

1. 结合项目中使用：

在流处理中，kafka与SparkStreaming进行结合，然后实时的读取数据，这样时候我们大多数使用的是手动提交模式。

。

## 题目41：SparkStreaming如何保证数据的防丢失

1. 问题分析：考官主要想考察SparkStreaming在实际开发环境中的运行机制以及和Kafka的数据对接了解程度做考察.
2. 核心答案讲解：

在Spark Streaming的生产实践中，要做到数据零丢失，需要满足以下几个先决条件:

1.输入的数据源是可靠的/数据接收器是可靠的

2.应用程序的metadata被application的driver持久化了(checkpointed )

3.启用了WAL特性（Write ahead log）

下面分别对这3个条件进行介绍:

1. 对于一些输入数据源（比如Kafka），Spark Streaming可以对已经接收的数据进行确认。输入的数据首先被接收器（receivers ）所接收，然后存储到Spark中（默认情况下，数据保存到2个执行器中以便进行容错）。数据一旦存储到Spark中，接收器可以对它进行确认（比如，如果消费Kafka里面的数据时可以更新Zookeeper里面的偏移量）。这种机制保证了在接收器突然挂掉的情况下也不会丢失数据：因为数据虽然被接收，但是没有被持久化的情况下是不会发送确认消息的。所以在接收器恢复的时候，数据可以被原端重新发送。
2. 可靠的数据源和接收器可以让我们从接收器挂掉的情况下恢复（或者是接收器运行的Exectuor和服务器挂掉都可以）。但是更棘手的问题是，如果Driver挂掉如何恢复？对此引入了很多技术来让Driver从失败中恢复。其中一个就是对应用程序的元数据进行Checkpoint。利用这个特性，Driver可以将应用程序的重要元数据持久化到可靠的存储中，比如HDFS；然后Driver可以利用这些持久化的数据进行恢复。
3. 但如果Exectuor已经接收并缓存了数据,这个时候挂掉了,这个时候数据是在Exectuor中,数据还是会丢失,启用了WAL机制，所以已经接收的数据被接收器写入到容错存储中，比如HDFS。由于采用了WAl机制，Driver可以从失败的点重新读取数据，即使Exectuor中内存的数据已经丢失了。在这个简单的方法下，Spark Streaming提供了一种即使是Driver挂掉也可以避免数据丢失的机制。
4. 问题扩展

如SparkStreaming整合Kafka有2种方式:Receiver 和 Direct.

Direct相对于Receiver的有点:

1.简化并行读取

2.高性能高可靠

3.一次且仅一次的事务机制

1. 结合项目中使用

相对来说Direct 适用于更多的业务场景以及有更好的可护展性。至于如何选择Receiver和Direct两种方式，除了业务场景外也跟团队相关，如果是应用初期，为了快速迭代应用，可以考虑采用Receiver方式；如果要深入使用的话则建议采用Direct方式

## 题目42：spark的checkpoint机制

1. 问题分析：SparkStreaming应用程序必须7\*24小时运行，因此必须能够适应与应用程序逻辑无关的故障（例如，系统故障，JVM崩溃等）。 为了实现这一点，Spark Streaming需要将足够的信息checkpoint到容错存储系统，以便它可以从故障中恢复
2. 核心答案讲解：

checkpoint的意思就是建立检查点,类似于快照,例如在spark计算里面 计算流程DAG特别长,服务器需要将整个DAG计算完成得出结果,但是如果在这很长的计算流程中突然中间算出的数据丢失了,spark又会根据RDD的依赖关系从头到尾计算一遍,这样子就很费性能,当然我们可以将中间的计算结果通过cache或者persist放到内存或者磁盘中,但是这样也不能保证数据完全不会丢失,存储的这个内存出问题了或者磁盘坏了,也会导致spark从头再根据RDD计算一遍,所以就有了checkpoint,其中checkpoint的作用就是将DAG中比较重要的中间数据做一个检查点将结果存储到一个高可用的地方(通常这个地方就是HDFS里面)

checkpoint有两种类型的数据：

1. 元数据checkpoint

在类似 HDFS 的容错存储上，保存 Streaming 计算信息。这种检查

点用来恢复运行 Streaming 应用程序失败的 Driver 进程。

1. 数据checkpoint

在进行跨越多个批次合并数据的有状态操作时尤其重要。通过周期检查

将转换 RDD 的中间状态进行可靠存储，借以切断无限增加的依赖。使用有状态的转换，如果 updateStateByKey 或者 reduceByKeyAndWindow 在应用程序中使用，那么需要提供检查点路径，对 RDD 进行周期性检查。

1. 问题扩展

checkpoint和持久化的区别：

1.持久化只是将数据保存在BlockManager中，而RDD的lineage是不变的。但是checkpoint执行完后，RDD已经没有之前所谓的依赖RDD了，而只有一个强行为其设置的checkpointRDD，RDD的lineage改变了。

2.持久化的数据丢失可能性更大，磁盘、内存都可能会存在数据丢失的情况。但是checkpoint的数据通常是存储在如HDFS等容错、高可用的文件系统，数据丢失可能性较小。

1. 结合项目中使用

我们每天都有来自各大搜索引擎搜索的数据，我们现在需要分析的是哪些搜索词带来的订单比较多，日志实时到达 Kafka 集群后，我们再通过 Spark Streaming 实时地从 Kafka 拉数据，然后解析日志、根据一定的逻辑过滤一些爬虫日志、订单搜索词关联等业务处理。为了能够在 Spark Streaming 程序挂掉后又能从断点处恢复，我们每隔 2 秒进行一次 Checkpoint，这些 Checkpoint 文件是存储在 HDFS 上的。在 Checkpoint 目录中主要存储两种数据：MetaDate checkpointing 和 Data checkpointing。

## 题目43：HBase的gc调优，为什么

1. 问题分析：在实际生产环境中,对于高并发Hbase写入项目，如果不进行Hbase参数调优。当数据到了一定的量的时候会出现各种错误导致写入失败，最普遍的有，连接超时、regionServer自动下线，本题考官主要想考察对HBase底层优化的了解程度,以及实际工作接触HBase时的使用程度.
2. 核心答案讲解：

大多数Hadoop客户以至少24G内存运行Hadoop,HBase使用的内存不断增长，但JDK可用的垃圾收集算法仍然相同。这导致了HBase的许多用户的一个主要问题：随着Java使用堆大小继续增长，垃圾回收导致的垃圾回收时停止程序所占用的时间变得越来越长。在垃圾回收导致的“stop-the-world”期间，任何到HBase客户端请求都不会被处理，造成用户可见的延迟，甚至超时。如果因为暂停导致请求超过一分钟响应，HBase本身也可能会停止.

HBase依赖Apache Zookeeper的管理群集成员和生命周期。如果服务器暂停的时间过长，它将无法发送心跳ping消息到Zookeeper，其余的服务器将假定该服务器已经死亡。这将导致主服务器启动特定的恢复程序，替换被认为死亡的服务器。当这个服务器从暂停中恢复时，会发现所有它拥有的租约都被撤销，进而只好自杀。

1. 问题扩展

由于Java GC导致的心跳包没有及时响应问题，在对延时要求敏感的场景非常普遍。集群中的服务器每500ms发送一次心跳包到mater服务器，而master服务器由于GC导致没有及时响应心跳包，进而认为服务器死亡，导致故障

Hadoop这个回收时间比较恐怖，在线上服务分配36G Heap，平均FullGC暂停时间是6-8秒，因此万恶不是“大堆”为首，而是“内存使用方式不当”为首.

1. 结合项目中使用

通过设置的CMSInitiatingOccupancyFraction，一些用户能够避免GC的问题。但对于其他的场景，GC会经常发生，无论CMSInitiatingOccupancyFraction设置的多么低。我们则经常看到在这些GC停顿时，堆还有几个GB的自由空间！鉴于这些情况，我们推测，我们的问题应该是由碎片引起的，而不是一些内存泄漏或不当调整。

Hbase GC配置参考:

export HBASE\_REGIONSERVER\_OPTS="-Xmx16g

-Xms16g -Xmn1g -XX:+UseParNewGC

-XX:+UseConcMarkSweepGC

-XX:CMSInitiatingOccupancyFraction=70

-Djava.net.preferIPv4Stack=true

参数解释：

-Xms16g、-Xmx16g：内存大小，根据集群情况设置，建议不要超过18G。

-Xmn1g：年轻代大小。

-XX:+UseParNewGC：设置年轻代为并发回收

-XX:+UseConcMarkSweepGC：启用CMS算法

-XX:CMSInitiatingOccupancyFraction=70：年老代使用了70%时回收内存

-Djava.net.preferIPv4Stack=true：禁用IPv6

可选参数：

-XX:-CMSConcurrentMTEnabled（-号代表否认）

当该标志被启用时，并发的CMS阶段将以多线程执行(因此，多个GC线程会与所有的应用程序线程并行工作)。

该标志已经默认开启，如果顺序执行更好，这取决于所使用的硬件，多线程执行可以通过-XX：-CMSConcurremntMTEnabled禁用。

-XX:+CMSIncrementalMode. 在增量模式下，CMS 收集器在并发阶段，不会独占整个周期，

而会周期性的暂停，唤醒应用线程。收集器把并发阶段工作，划分为片段，安排在次级(minor) 回收之间运行。

这对需要低延迟，运行在少量CPU服务器上的应用很有用。

## 题目44：HBase的读写机制

1. 问题分析：考官主要想让从HBase的读写过程,来判断面试者对HBase的流程了解程度,可以从HBase的读和写两方面来做解答。
2. 核心答案讲解：

HBase写数据流程

1、Client先访问zookeeper，从meta表获取相应region信息，然后找到meta表的数据

2、根据namespace、表名和rowkey根据meta表的数据找到写入数据对应的region信息

3、找到对应的regionserver

4、把数据分别写到HLog和MemStore上一份

5、MemStore达到一个阈值后则把数据刷成一个StoreFile文件。（若MemStore中的数据有丢失，则可以总HLog上恢复）

6、 当多个StoreFile文件达到一定的大小后，会触发Compact合并操作，合并为一个StoreFile，（这里同时进行版本的合并和数据删除。）

7、 当Storefile大小超过一定阈值后，会把当前的Region分割为两个（Split），这里相当于把一个大的region分割成两个region，并由Hmaster分配到相应的HRegionServer，实现负载均衡。

HBase读取数据流程

1、Client先访问zookeeper，从zookeeper中找到meta表region的位置，然后读取meta表中的数据。meta中又存储了用户表的region信息。

2、根据namespace、表名和rowkey在meta表中找到对应的region信息

3、找到这个region对应的regionserver

4、查找对应的region

5、先从MemStore找数据，如果没有，再到StoreFile上读(为了读取的效率)

1. 问题扩展

hbase 使用 MemStore 和 StoreFile 存储对表的更新。

数据在更新时首先写入 Log(WAL log)和内存(MemStore)中，MemStore 中的数据

是排序的，当 MemStore 累计到一定阈值时，就会创建一个新的 MemStore，并且

将老的 MemStore 添加到 flush 队列，由单独的线程 flush 到磁盘上，成为一个

StoreFile。于此同时，系统会在 zookeeper 中记录一个 redo point，表示这个

时刻之前的变更已经持久化了。

当系统出现意外时，可能导致内存(MemStore)中的数据丢失，此时使用 Log(WAL

log)来恢复 checkpoint 之后的数据。

StoreFile 是只读的，一旦创建后就不可以再修改。因此 Hbase 的更新其实是不断追加的操作。当一个 Store 中的 StoreFile 达到一定的阈值后，就会进行一次合并(minor\_compact,major\_compact),将对同一个key的修改合并到一起，形成一个大的StoreFile，当StoreFile的大小达到一定阈值后，又会对 StoreFile 进行 split，等分为两个 StoreFile。由于对表的更新是不断追加的，compact 时，需要访问 Store 中全部的 StoreFile 和MemStore，将他们按 row key 进行合并，由于 StoreFile 和 MemStore 都是经过排序的，并且 StoreFile 带有内存中索引，合并的过程还是比较快。

1. 结合项目中使用

在开发过程中,写请求与读请求都比较高，业务往往接受：写请求慢点可以，读请求越快越好，最好有单独的资源保障

scan与get都比较多，业务希望scan不影响get（因为scan比较消耗资源）

hbase.ipc.server.callqueue.read.ratio 设置为0.5，代表有50%的线程数处理读请求

如果再设置hbase.ipc.server.callqueue.scan.ratio 设置为0.5，则代表在50%的读线程之中，再有50%的线程处理scan，也就是全部线程的25%

根据实际的业务配置以上数值，默认情况下是没有配置的，也就是读写都共享。

## 题目45：HBase如何设计rowkey，如何在负载均衡和读写性能之间做出平衡

1. 问题分析：本题主要考察rowkey的设计原则,以及在处理负载均衡时对读写性能的影响.
2. 核心答案讲解：

由于在开始建表时，表只会有一个region，并随着region增大而拆分成更多的region，这些region才能分布在多个regionserver上从而使负载均分。对于写负载很大的业务，如果一开始所有负载都在一个regionserver上，则该regionserver会承受不了而导致数据丢失。因此，有必要在一开始就将HBase的负载均摊到每个regionserver。要将负载均摊，可用的方法就是建表时将表分区，将这些分区均匀地放到每个regionserver上，然后客户端在进行写操作的时候，将这些写操作均匀分布到各个分区上.

Rowkey设计的3个原则

1 rowkey 长度原则

rowkey 是一个二进制码流，可以是任意字符串，最大长度 64kb，实际应

用中一般为 10-100bytes，以 byte[]形式保存，一般设计成定长。建议越短越好，不要超过 16 个字节, 原因如下：

数据的持久化文件 HFile 中是按照 KeyValue 存储的，如果 rowkey 过长会

极大影响 HFile 的存储效率MemStore 将缓存部分数据到内存，如果 rowkey 字段过长，内存的有效利用率就会降低，系统不能缓存更多的数据，这样会降低检索效率

2 rowkey 散列原则

如果 rowkey 按照时间戳的方式递增，不要将时间放在二进制码的前面，建议将

rowkey 的高位作为散列字段，由程序随机生成，低位放时间字段，这样将提高数据均衡分布在每个 RegionServer，以实现负载均衡的几率。如果没有散列字段，首字段直接是时间信息，所有的数据都会集中在一个 RegionServer 上，这一方面不能发挥整个集群的并发处理能力，另一方面势必造成此台RegionServer资源严重消耗（比如IO耗尽、handler耗尽等），落在该台RegionServer上的其他业务会因此受到很大的波及。可见，读请求不均衡不仅会造成本身业务性能很差，还会严重影响其他业务。当然，写请求不均衡也会造成类似的问题，可见负载不均衡是HBase的大忌。

3 rowkey 唯一原则

必须在设计上保证其唯一性，rowkey 是按照字典顺序排序存储的，因此，设计

rowkey 的时候，要充分利用这个排序的特点，将经常读取的数据存储到一块，将最近可能会被访问的数据放到一块。

1. 问题扩展

HBase流量限制和表负载均衡

为什么要做流量限制，无限制全量跑不是更好吗？举个例子，比如今天的双十一日，数据流量是非常大的。如果不限制用户和表的流量，某些重要的核心业务，需要在资源有限的情况下优先保证正常运行。如果非核心业务在此期间其QPS一直降不下来，严重消耗系统资源，影响核心业务的正常运作。

针对上述问题，可以采取以下方案来解决：

资源限制：针对用户、命名空间及表的请求大小和QPS进行限制。

资源隔离：将不同表中的数据通过物理隔离，均衡到不同的RegionServer上。

1. 结合项目中使用

在实际的开发中,需要结合业务来设计rowkey:

事务数据是带时间属性的，建议将时间信息存入到Rowkey中，这有助于提示查询检索速度。对于事务数据建议缺省就按天为数据建表，这样设计的好处是多方面的。按天分表后，时间信息就可以去掉日期部分只保留小时分钟毫秒，这样4个字节即可搞定。加上散列字段2个字节一共6个字节即可组成唯一Rowkey。这样的设计从操作系统内存管理层面无法节省开销，因为64位操作系统是必须8字节对齐。但是对于持久化存储中Rowkey部分可以节省25%的开销。也许有人要问为什么不将时间字段以主机字节序保存，这样它也可以作为散列字段了。这是因为时间范围内的数据还是尽量保证连续，相同时间范围内的数据查找的概率很大，对查询检索有好的效果，因此使用独立的散列字段效果更好，对于某些应用，我们可以考虑利用散列字段全部或者部分来存储某些数据的字段信息，只要保证相同散列值在同一时间（毫秒）唯一。

统计数据也是带时间属性的，统计数据最小单位只会到分钟（到秒预统计就没意义了）。同时对于统计数据我们也缺省采用按天数据分表，这样设计的好处无需多说。按天分表后，时间信息只需要保留小时分钟，那么0~1400只需占用两个字节即可保存时间信息。由于统计数据某些维度数量非常庞大，因此需要4个字节作为序列字段，因此将散列字段同时作为序列字段使用也是6个字节组成唯一Rowkey。同样这样的设计从操作系统内存管理层面无法节省开销，因为64位操作系统是必须8字节对齐。但是对于持久化存储中Rowkey部分可以节省25%的开销。预统计数据可能涉及到多次反复的重计算要求，需确保作废的数据能有效删除，同时不能影响散列的均衡效果，因此要特殊处理。

通用数据采用自增序列作为唯一主键，用户可以选择按天建分表也可以选择单表模式。这种模式需要确保同时多个入库加载模块运行时散列字段（序列字段）的唯一性。可以考虑给不同的加载模块赋予唯一因子区别。

## 题目46： Hive和hbase的区别。

(1)问题分析：

基础考核，概念和特性都需要回答，考察大数据基础。同时可以延伸业务场景，考察学生在项目实战中不同服务如何区分配合使用。

(2)核心答案讲解：

hive

hive是基于Hadoop的数据仓库工具，可以将结构化数据文件映射为数据库表。并提供简单的sql功能，可以将sql转化为mr任务运行。因为sql学习成本低，不必专门开发mr应用，十分适合数据仓库的统计分析。

hbase

HBase是建立在HDFS之上,提供高可靠性的列存储，实时读写的数据库系统。它介于Nosql和关系型数据库之间，仅通过主键和主键的range来检索数据，仅支持单行事务。主要用来存储非结构化和半结构化的松散数据。

区别

Hive和Hbase是两种基于Hadoop的不同技术：Hive是一种类SQL的引擎，并且运行MapReduce任务，Hbase是一种在Hadoop之上的NoSQL 的Key/vale数据库，只支持简单的行列操作。当然，这两种工具是可以同时使用的。Hive可以用来进行统计查询，HBase可以用来进行快速的实时查询，数据也可以从Hive写到Hbase，设置再从Hbase写回Hive。

(3)问题扩展:

类似的存储服务特别多，nosql的关系型的文件型的。比如谈谈es和hbase的区别，mysql和hbase的区别，hbase和hdfs的区别等等。需要先答出两种存储服务的概念，再根据核心功能，特性的不同答出两者区别。

(4)项目应用:

在哪种场景下，使用哪种技术。当所存储数据需要快速插入查询时，使用hbase，所以storm或sparksteaming常常存储入hbase。当数据需要大量聚合运算，计算分析结果时，使用hive存储，所以hive是数仓，etl的常用工具。

## 题目47： 介绍hbase的协处理器。

(1)问题分析：

考官主要考核面试人员对Hbase组成原理即协处理器的了解程度,以此判断大数据组件基础知识是否扎实。

(2)核心答案讲解：

1.在0.92版本后新增，允许在region服务器上运行自己的代码，更准确地说是允许用户执行**region级的操作**，并且可以使用与RDBMS中触发器类似的功能。可理解为服务端的拦截器，可根据需求确定拦截点，再重写这些拦截点对应的方法，做到客户端的读取API配合筛选机制可控制返回的数据量，进一步优化，例，数据的处理流程直接放到服务器端执行，然后仅返回一个小的处理结果集。类似于一个小型的MapReduce 框架，该框架将工作分发到整个集群节点。

2.处理器框架提供了两大类Observer、endPoint通过继承这两大类来扩展自己的功能。其中endpoint还需要通过protocl来定义接口实现客户端代码进行rpc通信以此来进行数据的搜集归并。而Observer则不需要客户端代码，只在特定操作发生的时候触发服务端代码的实现。

(3)问题扩展:

协处理器的意义，这样设计的好处。类似这些问题，考察对hbase的整体的特性的理解。答：不设置存在的问题是，二级索引较难，很难进行简单的排序、求和、计数等操作，版本限制下难以进行上述操作，不是不行。为了降低难度，才提出了协处理器的概念。

(4)项目应用:

在向hbase中put一条数据时同时也要put一个标识符，或者需要put两个单词，单词二是单词一的大写格式。类似业务不需要在存储前准备数据使用协处理器更高效。

## 题目48： Spark，hbase集群如何对jvm调优。

(1)问题分析：

Jvm是spark基于yarn、hive的RegionServer底层服务进程。对jvm的调优考察jvm的知识入垃圾回收机制，收集器调优等，同时考察这些分布式集群的特性。

(2)核心答案讲解：

Spark：1. 对于垃圾回收的性能问题，首先要做的就是，使用高效的数据结构。

2. 持久化RDD时，使用序列化的持久化级别，而且使用Kyro序列化类库，这样partition就只是一个对象，一个字节数组。

3. 给Eden区域分配更大的空间，使用-Xmn即可，通常建议给Eden区域，预计大小的4/3

如果使用的是HDFS文件，那么很好估计Eden区域的大小，如果每个executor有4个task，然后每个hdfs压缩块解压缩后大小是3倍，此外每个hdfs块的大小是64M，那么Eden区域的预计大小就是：4\*3\*64M，然后通过-Xmn参数，将Eden区域的大小设置为4\*3\*64\*3/4。

Hbase: 对垃圾回收的优化，RegionServer因为GC的原因不能分配太大的堆内存，20~24GB或者更小比较适合。堆的大小通过export HBASE\_HEAPSIZE=xxx来设置。指定新生代的空间：-Xmn512m年轻代的空间不能太小，也不能太大。太小的话容易在老生代中产生很多碎片，太大的话会使年轻代中的垃圾收集时间过长。HBase官网给出的一个建议是512M。提高IHOP：-XX:InitiatingHeapOccupancyPercent=xx（默认为45）。对CMS收集器的优化，在CMS工作时，不断有对象进入持久代，导致持久代空间不足，这时会放弃原本并行执行的垃圾收集，转而使用stop-the-world的复制算法。为了避免是由老年代内存碎片问题导致的空间不足问题。可以配置以下参数：  
-XX:CMSInitiatingOccupancyFraction=70  
这个参数的意思是，当老年代的空间分配了70%的时候就开始进行CMS垃圾收集，在一定程度上避免提升失败

(3)问题扩展:

调优可以扩展非常多的方向，比如spark的shuffle调优，spark的运行节点调优甚至是算子代码调优等等，需要全面了解spark的运行机，并把调优方向区分，会更方便记忆。

(4)项目应用:

通过yarn去运行的话，那么就通过yarn的[界面](https://www.baidu.com/s?wd=%E7%95%8C%E9%9D%A2&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)，可以看到每个stage的运行情况，包括每个task的运行时间，GC时间，如果发现gc太频繁，时间太长，此时就可以适当调节这个比例。降低cache操作的内存占比，或者直接使用persisit操作。

## 题目49： 怎么把数据放到hbase的，hive整合hbase。

(1)问题分析：

更偏向于项目实战问题，考察学生究竟有没有实际操作能力，非理论知识。同时要清楚大数据体系流程，这两种服务的作用，适用场景。

(2)核心答案讲解：

如何把数据放到hbse的：除了使用api编写程序完成数据写入hbase外，还有多种数据直接转移到hbase的方式，比如将HDFS上的文件中的数据导入到hbase中，有两种办法，一种是使用hbase提供好的import工具。还可以使用sqoop工具写对应sqoop脚本完成导入。从mysql 导入数据到hbase，也可以使用sqoop。

Hive与HBase整合的实现是利用两者本身对外的API接口互相通信来完成的，其具体工作交由Hive的lib目录中的hive-hbase-handler-\*.jar工具类来实现，然后再hive shell使用整合命令创建hive表即可整合成功。通过Hive把数据加载到HBase中，数据源可以是文件也可以是Hive中的表。通过整合，让HBase支持JOIN、GROUP等SQL查询语法。不仅可完成HBase的数据实时查询，也可以使用Hive查询HBase中的数据完成复杂的数据分析。

(3)问题扩展:

数据如何在不同的库中转移，如Hbase，hive，mysql数据怎么跨不同结构数据库移动：使用开源工具有sqoop或者一些非开源云服务软件如datax完成数据同步。

(4)项目应用:

如果数据在hbase中查询不方便但又需要快速查询出结果或者进行一些简单聚合预算，可以把hive和hbase整合。对此也有常用第三方工具如finex等也可以快速执行查询操作。

## 题目50： 谈谈对kafka的理解，kafka如何保证数据不丢失

(1)问题分析：

对kafka基础概念的理解，考察一定的语言表达能力，需要思路清晰，可以把kafka的各种特点列举并一一解释。数据不丢失可以分两个点回答，在消费时不丢失和kafka的容灾能力。

(2)核心答案讲解：

对kafka的理解：Kafka是一个分布式数据流平台，可以运行在单台上，也可多台服务器形成集群。它提供发布和订阅功能，使用者可以发送数据到Kafka中，也可以从Kafka中读取数据。Kafka具有高吞吐、低延迟、高容错等特点。0.8版本后，陆续加入了一些复制、应答和故障转移等机制以后，才可以让我们在其他关键性业务中使用。

消息不丢失：通过request.required.acks属性进行配置，有三个选项：

0代表：不进行消息接收是否成功的确认(默认值)；

1代表：当Leader副本接收成功后，返回接收成功确认信息；

-1代表：当Leader和Follower副本都接收成功后，返回接收成功确认信息；acks设置为0时，不和Kafka集群进行消息接受确认，当网络发生异常等情况时，存在消息丢失的可能；想要不丢失消息数据就选：同步、ack=-1的策略。

(3)问题扩展:

同消息不丢失伴生问题，如何避免重复消费：数据重复消费的情况，如果处理  
去重：将消息的唯一标识保存到外部介质中，每次消费处理时判断是否处理过；  
不管：大数据场景中，[报表系统](https://www.baidu.com/s?wd=%E6%8A%A5%E8%A1%A8%E7%B3%BB%E7%BB%9F&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)或者日志信息丢失几条都无所谓，不会影响最终的统计分析结

(4)项目应用:

在kafka下游经常出现系统崩溃，需要回滚的问题，如何做到消息不重复消费是项目中很重要的一部分。可以修改offet从制定位置消费，也可以根据消息内容，从头消费toptic。对唯一字段进行过滤，做到消费过的字段不再消费。

## 题目51：Kafka和sparkStreaming的整合，手动提交的offset调用了什么方法？

## 问题分析

考官主要想要考核的是学员对于流对接方面的了解。kafka与sparkStreaming的整合可以参考官网资料：

http://spark.apache.org/docs/latest/streaming-kafka-0-10-integration.html

## 核心问题讲解

在spark streaming读取kafka的数据中，spark streaming提供了两个接口读取kafka中的数据，分别是KafkaUtils.createDstream，KafkaUtils.createDirectStream，前者会自动把offset更新到zk中，默认会丢数据，效率低，后者不会经过zk，效率更高，需要自己手动维护offset，通过维护offset写到zk中，保障数据零丢失，只处理一次，下面来看看KafkaUtils.createDirectStream的使用，把zk的端口改成了9999，防止和kakfa自带的zk的端口产生冲突，下面是一些测试代码，即使spark streaming挂了 ，另一方往topic中写数据，下次启动streaming程序也能读取，做到数据零丢失，不同的group.id下只读取一次

package com.jingde.sparkstreamlast

import kafka.serializer.StringDecoder

import org.apache.log4j.{ Level, Logger }

import org.apache.spark.SparkConf

import org.apache.spark.rdd.RDD

import org.apache.spark.streaming.kafka.\_

import org.apache.spark.streaming.{ Seconds, StreamingContext }

import org.apache.spark.streaming.kafka.KafkaUtils

import org.apache.spark.streaming.kafka.OffsetRange

import org.apache.log4j.{ Level, Logger }

import org.I0Itec.zkclient.ZkClient

import org.I0Itec.zkclient.exception.ZkMarshallingError

import org.I0Itec.zkclient.serialize.ZkSerializer

import kafka.utils.ZkUtils

import kafka.utils.ZKGroupTopicDirs

import org.apache.spark.streaming.dstream.InputDStream

import kafka.common.TopicAndPartition

import kafka.message.MessageAndMetadata

import kafka.api.OffsetRequest

import kafka.api.PartitionOffsetRequestInfo

import kafka.consumer.SimpleConsumer

import kafka.api.TopicMetadataRequest

object StreamingFromKafka {

val groupId = "logs"

val topic = "streaming"

val zkClient = new ZkClient("localhost:9999", 60000, 60000, new ZkSerializer {

override def serialize(data: Object): Array[Byte] = {

try {

return data.toString().getBytes("UTF-8")

} catch {

case e: ZkMarshallingError => return null

}

}

override def deserialize(bytes: Array[Byte]): Object = {

try {

return new String(bytes, "UTF-8")

} catch {

case e: ZkMarshallingError => return null

}

}

})

val topicDirs = new ZKGroupTopicDirs("spark\_streaming\_test", topic)

val zkTopicPath = s"${topicDirs.consumerOffsetDir}"

def main(args: Array[String]): Unit = {

Logger.getLogger("org.apache.spark").setLevel(Level.WARN)

Logger.getLogger("org.eclipse.jetty.server").setLevel(Level.OFF)

val sparkConf = new SparkConf().setAppName("DirectKafkaWordCount")

sparkConf.setMaster("local[\*]")

sparkConf.set("spark.streaming.kafka.maxRatePerPartition", "2")

sparkConf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")

val ssc = new StreamingContext(sparkConf, Seconds(2))

val kafkaParams = Map("metadata.broker.list" -> "localhost:9092", "group.id" -> groupId, "zookeeper.connect"->"localhost:9999",

"auto.offset.reset" -> kafka.api.OffsetRequest.SmallestTimeString )

val topics = Set(topic)

val children = zkClient.countChildren(s"${topicDirs.consumerOffsetDir}")

var kafkaStream: InputDStream[(String, String)] = null

var fromOffsets: Map[TopicAndPartition, Long] = Map()

if (children > 0) {

//---get partition leader begin----

val topicList = List(topic)

val req = new TopicMetadataRequest(topicList,0) //得到该topic的一些信息，比如broker,partition分布情况

val getLeaderConsumer = new SimpleConsumer("localhost",9092,10000,10000,"OffsetLookup") // brokerList的host 、brokerList的port、过期时间、过期时间

val res = getLeaderConsumer.send(req) //TopicMetadataRequest topic broker partition 的一些信息

val topicMetaOption = res.topicsMetadata.headOption

val partitions = topicMetaOption match{

case Some(tm) =>

tm.partitionsMetadata.map(pm=>(pm.partitionId,pm.leader.get.host)).toMap[Int,String]

case None =>

Map[Int,String]()

}

for (i <- 0 until children) {

val partitionOffset = zkClient.readData[String](s"${topicDirs.consumerOffsetDir}/${i}")

val tp = TopicAndPartition(topic, i)

//---additional begin-----

val requestMin = OffsetRequest(Map(tp -> PartitionOffsetRequestInfo(OffsetRequest.EarliestTime,1))) // -2,1

val consumerMin = new SimpleConsumer(partitions(i),9092,10000,10000,"getMinOffset")

val curOffsets = consumerMin.getOffsetsBefore(requestMin).partitionErrorAndOffsets(tp).offsets

var nextOffset = partitionOffset.toLong

if(curOffsets.length >0 && nextOffset < curOffsets.head){ //如果下一个offset小于当前的offset

nextOffset = curOffsets.head

}

//---additional end-----

fromOffsets += (tp -> nextOffset)

fromOffsets += (tp -> partitionOffset.toLong) //将不同 partition 对应的 offset 增加到 fromOffsets 中

}

val messageHandler = (mmd: MessageAndMetadata[String, String]) => (mmd.topic, mmd.message()) //这个会将 kafka 的消息进行 transform，最终 kafak 的数据都会变成 (topic\_name, message) 这样的 tuple

kafkaStream = KafkaUtils.createDirectStream[String, String, StringDecoder, StringDecoder, (String, String)](ssc, kafkaParams, fromOffsets, messageHandler)

} else {

kafkaStream = KafkaUtils.createDirectStream[String, String, StringDecoder, StringDecoder](ssc, kafkaParams, topics)

}

var offsetRanges = Array[OffsetRange]()

kafkaStream.transform { rdd =>

offsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges

rdd

}.foreachRDD {

rdd =>

{

rdd.map(\_.\_2).foreachPartition { element => element.foreach { println } }

for (o <- offsetRanges) {

ZkUtils.updatePersistentPath(zkClient, s"${topicDirs.consumerOffsetDir}/${o.partition}", o.fromOffset.toString)

}

}

}

ssc.start()

ssc.awaitTermination()

}

}

## 问题扩展

暂无

## 结合项目中使用

暂无

## 题目52：hive怎么消费kafka的数据的

## 问题分析

考官主要想要考核的是学员对知识的活用能力。由于flume可以作为一个数据传输工具使用，flume本身可以整合hive与kafka二者，所以可以考虑从这个方面来解题。

## 核心问题讲解

Flume整合kafka与hive，达到hive消费kafka中的数据的步骤：

1. hive建表
2. 配置flume（可以参考官网配置flume）
3. 启动hive前要设置hive.txn.manager：

第一种方式：脚本命令

set hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;

 第二种方式：配置文件（hive-site.xml）

<property>

<name>hive.txn.manager</name>

<value>org.apache.hadoop.hive.ql.lockmgr.DbTxnManager</value>

</property>

1. 依次启动hive、flume
2. 创建topic
3. 启动kafka生产者生产消息即可。

## 问题扩展

暂无

## （4）结合项目中使用

暂无

## 题目53：.Kafka如何管理自身的offset

## 问题分析

考官主要想要考核的是kafka不同版本的offset管理有所不同。考研学员对kafka这个框架的设计上的一种优化思想的了解。

## 核心问题讲解

早在 0.8.2.2 版本，kafka已支持存入消费的 offset 到Topic中，只是那时候默认是将消费的 offset 存放在 Zookeeper 集群中。现在0.10.1.1及以后的版本中，官方默认将消费的offset存储在 Kafka 的Topic中，同时，也保留了存储在 Zookeeper 的接口，通过 offsets.storage 属性来进行设置。

之前版本，Kafka其实存在一个比较大的隐患，就是利用 Zookeeper 来存储记录每个消费者/组的消费进度。虽然，在使用过程当中，JVM帮助我们完成了一些优化，但是消费者需要频繁的去与 Zookeeper 进行交互，而利用ZKClient的API操作Zookeeper频繁的Write其本身就是一个比较低效的Action，对于后期水平扩展也是一个比较头疼的问题。如果期间 Zookeeper 集群发生变化，那 Kafka 集群的吞吐量也跟着受影响。

在新版 Kafka 以及之后的版本，Kafka 消费的offset都会默认存放在 Kafka 集群中的一个叫 \_\_consumer\_offsets 的topic中。

当然，其实她实现的原理也让我们很熟悉，利用 Kafka 自身的 Topic，以消费的Group，Topic，以及Partition做为组合 Key。所有的消费offset都提交写入到上述的Topic中。因为这部分消息是非常重要，以至于是不能容忍丢数据的，所以消息的 acking 级别设置为了 -1，生产者等到所有的 ISR 都收到消息后才会得到 ack（数据安全性极好，当然，其速度会有所影响）。所以 Kafka 又在内存中维护了一个关于 Group，Topic 和 Partition 的三元组来维护的 offset 信息，消费者获取的offset的时候会直接从内存中获取。

kafka 提供三种语义的传递：

1至少一次

2至多一次

3较精确一次

首先在 producer 端保证1和2的语义是非常简单的，至少一次只需要同步确认即可（确认方式分为只需要 leader 确认以及所有副本都确认，第二种更加具有容错性），至多一次最简单只需要异步不断的发送即可，效率也比较高。目前在 producer 端还不能保证较精确一次，在未来有可能实现，实现方式如下：在同步确认的基础上为每一条消息加一个主键，如果发现主键曾经接受过，则丢弃。

在 consumer 端，大家都知道可以控制 offset，所以可以控制消费，其实 offset 只有在重启的时候才会用到。在机器正常运行时我们用的是 position，我们实时消费的位置也是 position 而不是 offset。我们可以得到每一条消息的 position。如果我们在处理消息之前就将当前消息的 position 保存到 zk 上即 offset，这就是只多一次消费，因为我们可能保存成功后，消息还没有消费机器就挂了，当机器再打开时此消息就丢失了；或者我们可以先消费消息然后保存 position 到 zk 上即 offset，此时我们就是至少一次，因为我们可能在消费完消息后offset 没有保存成功。而较精确一次的做法就是让 position的保存和消息的消费成为原子性操作，比如将消息和 position 同时保存到 hdfs 上 ，此时保存的 position 就称为 offset，当机器重启后，从 hdfs重新读入offset，这就是较精确一次。

## 问题扩展

暂无

## （4）结合项目中使用

暂无

# hive与数据仓库

## 题目54： kafka如何保证数据不会出现丢失或者重复消费的情况？

## 问题分析

考官主要想要考核的是考生是否了解kafka消息不丢失机制的运作原理以及造成消息重复消费或者丢失的原因。了解了问题发生的原因才能够针对性的解决问题。

## （2）核心问题讲解

使用同步模式的时候，有3种状态保证消息被安全生产，在配置为1（只保证写入leader成功）的话，如果刚好leader partition挂了，数据就会丢失。

还有一种情况可能会丢失消息，就是使用异步模式的时候，当缓冲区满了，如果配置为0（还没有收到确认的情况下，缓冲池一满，就清空缓冲池里的消息），

数据就会被立即丢弃掉。

在数据生产时避免数据丢失的方法：

只要能避免上述两种情况，那么就可以保证消息不会被丢失。

就是说在同步模式的时候，确认机制设置为-1，也就是让消息写入leader和所有的副本。

还有，在异步模式下，如果消息发出去了，但还没有收到确认的时候，缓冲池满了，在配置文件中设置成不限制阻塞超时的时间，也就说让生产端一直阻塞，这样也能保证数据不会丢失。

在数据消费时，避免数据丢失的方法：如果使用了storm，要开启storm的ackfail机制；如果没有使用storm，确认数据被完成处理之后，再更新offset值。低级API中需要手动控制offset值。

## （3）问题扩展

暂无

## 结合项目中使用

暂无

## 题目55： kafka消费数据是怎么消费的,用的是什么方式?

## 问题分析

考官主要想要考核的是考生对kafka消费消息的几种方式的理解。

## 核心问题讲解

Kafka消费数据的方式主要包含如下几种：

1、指定多主题消费

    consumer.subscribe(Arrays.asList(“t4”，“t5”))；

2、指定分区消费

    consumer.assign(list)；

3、手动修改偏移量

    consumer.commitSync()；                //提交当前消费偏移量

    consumer.commitSync(Map<TopicPartition, OffsetAndMetadata>)    //提交指定偏移量

    consumer.assign(Arrays.asList(tp))；

4、seek，修改偏移量搜索指针，顺序读取数据

    consumer.assign(Arrays.asList(tp))；

        consumer.seek(tp，0)；

代码：

import org.apache.kafka.clients.consumer.ConsumerRecord;

import org.apache.kafka.clients.consumer.ConsumerRecords;

import org.apache.kafka.clients.consumer.KafkaConsumer;

import org.apache.kafka.clients.consumer.OffsetAndMetadata;

import org.apache.kafka.common.TopicPartition;

import java.util.\*;

public class NewConsumer {

public static void main(String[] args) {

Properties props = new Properties();

props.put("bootstrap.servers","s102:9092,s103:9092,s104:9092");

props.put("group.id", "g3");

props.put("enable.auto.commit", "false");

props.put("auto.commit.interval.ms", "100");

props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);

//通过subscribe方法，指定多主题消费

//consumer.subscribe(Arrays.asList("t4","t5"));

//指定分区消费

// ArrayList<TopicPartition> list = new ArrayList<TopicPartition>();

// TopicPartition tp = new TopicPartition("t1", 0);

// TopicPartition tp2 = new TopicPartition("t4", 0);

// TopicPartition tp3 = new TopicPartition("t4", 1);

// TopicPartition tp4 = new TopicPartition("t4", 2);

// list.add(tp);

// list.add(tp2);

// list.add(tp3);

// list.add(tp4);

// consumer.assign(list);

Map<TopicPartition, OffsetAndMetadata> offset = new HashMap<TopicPartition, OffsetAndMetadata>();

//指定分区

TopicPartition tp = new TopicPartition("t4", 0);

//指定偏移量

OffsetAndMetadata metadata = new OffsetAndMetadata(3);

offset.put(tp,metadata);

//修改偏移量

consumer.commitSync(offset);

//订阅主题

//consumer.subscribe(Arrays.asList("t4"));

consumer.assign(Arrays.asList(tp));

//指定分区

// TopicPartition tp = new TopicPartition("t4", 0);

// consumer.assign(Arrays.asList(tp));

// //修改搜索指针

// consumer.seek(tp,10);

while (true) {

ConsumerRecords<String, String> records = consumer.poll(5000);

for (ConsumerRecord<String, String> record : records)

System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(), record.value());

consumer.commitSync();

}

}

}

## 问题扩展

暂无

## （4）结合项目中使用

暂无

## 题目56： 在hive中如何处理小文件合并问题

1. 问题分析

考官主考的是对hive中常见问题解决方案的使用。

1. 核心问题讲解

配置Map输入合并

-- 每个Map最大输入大小，决定合并后的文件数

set mapred.max.split.size=256000000;

-- 一个节点上split的至少的大小 ，决定了多个data node上的文件是否需要合并

set mapred.min.split.size.per.node=100000000;

-- 一个交换机下split的至少的大小，决定了多个交换机上的文件是否需要合并

set mapred.min.split.size.per.rack=100000000;

-- 执行Map前进行小文件合并

set hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;

配置Hive结果合并

我们可以通过一些配置项来使Hive在执行结束后对结果文件进行合并：

hive.merge.mapfiles 在map-only job后合并文件，默认true

hive.merge.mapredfiles 在map-reduce job后合并文件，默认false

hive.merge.size.per.task 合并后每个文件的大小，默认256000000

hive.merge.smallfiles.avgsize 平均文件大小，是决定是否执行合并操作的阈值，默认16000000

Hive在对结果文件进行合并时会执行一个额外的map-only脚本，mapper的数量是文件总大小除以size.per.task参数所得的值，触发合并的条件是：

根据查询类型不同，相应的mapfiles/mapredfiles参数需要打开；

结果文件的平均大小需要大于avgsize参数的值。

1. 问题扩展

对于输出结果为压缩文件形式存储的情况，要解决小文件问题，如果在Map输入前合并，对输出的文件存储格式并没有限制。但是如果使用输出合并，则必须配合SequenceFile来存储，否则无法进行合并

1. 结合项目中使用

## 题目57： hive的存储格式，以及压缩算法

1. 问题分析

考官主要想考的是对hive的掌握程度。

1. 核心问题讲解

文件格式

Textfile

SequenceFile

RCFile

ORCFile

Parquet

Avro

压缩算法的编解码器



1. 问题扩展

https://yq.aliyun.com/articles/179745

（4）结合项目中使用

## 题目58： 是用什么ETL工具进行hive中数据的ETL

1. 问题分析

考官想考的是面试考的开发经验

1. 核心问题讲解

 Hive 构建于传统的数据库和数据仓库理念之上。它对待数据的方式就像是它有一个基于 SQL 或基于架构的结构。在 Hive 中，您可以将数据加载到 HDFS 中，或者将数据直接加载到 Hive 表中。不过，Pig 更加类似于标准的 ETL 脚本语言。在 Pig 中，您的心里可能有一个模式，但您更关心的是如何利用更复杂的功能在 HDFS 中转换和集成数据，而不是简单地将它们放进一个特定的表或数据库。因为 Pig 和 Hive 都使用了 MapReduce 功能，所以它们在执行非面向批量的处理时可能并不是那么快。有些开源工具试图改变这种限制，但问题依然存在。

1. 问题扩展

Apache Hive 数据仓库软件有助于查询和管理位于分布式存储中的大型数据集。对于 ETL 而言，Hive 是一个强大的工具，而对于 Hadoop，它既是数据仓库，也是 Hadoop 的数据库。不过，相对于传统的数据库，它是相对缓慢的。它没有提供所有的 SQL 特性，甚至没有提供与传统的数据库相同的数据库特性。但它支持 SQL，它的确像一个数据库那样工作，它让更多的人（即使那些不是程序员的人）可以获得 Hadoop 技术。它提供了一种将非结构化和半结构化数据转化为基于模式的可用数据的方法。要建立一个主数据管理系统？您可以利用 Hive。要建立一个数据仓库？您也可以利用 Hive，但您需要学习一些技巧，使 Hive 成为一个强大的 ETL 工具。

相对于 Apache Pig 和 MapReduce，Hive 让传统的 RDBMS 数据库开发人员或了解 SQL 的其他人可以更容易访问和转换 Hadoop 中的数据。然而，Pig 不太容易理解，对于那些没有软件开发背景的人来说道，学习曲线是陡峭的。MapReduce 是 Java™、C ++ 和 Python 程序员可以相对迅速学会的技术。但是，如果没有一项技术（如 Java）基础，几乎不可能学会 MapReduce。因此，如果您知道 SQL，那么学习和使用 Hive 就会比较容易。

（4）结合项目中使用

## 题目59： 如何保证hive中数据的质量

1. 问题分析

考官想考面试者在工作中的具体工作经验。

1. 核心问题讲解

首先，数据出现质量问题有哪些原因或者情况？

其次，针对这些原因，制定清洗策略。

一般的数据质量出现问题的有：无效，重复，缺失，不一致，错误值，格式出错，业务逻辑规则有问题，抽取数据程序有错等，另外还有就是统计口径不一致，也会导致看到的数据不是想要的。

根据这些情况，如何清洗？人工，还是编写程序？这个依据数据量大小及挖掘系统要求看吧。如果出现这类型的错误很多，一定要写程序自动清洗，如果只是小量的不影响的，可以忽略不计。

1. 问题扩展

企业不同的时期业务系统处理方式上逐步优化产生的数据差异：

产生原因：企业在不同的发展时间，系统处理会有所差异，特别是二开比较多的公司

解决方案：A.后续规范的数据与前面不规范的数据，看是否可以通过相对应的关系，进行整理统一；

B.如果上述都不能处理的话，我想还是对前面的一些数据进行分开统计分析，否则两者不一样统计了来会误导业务人员

以前在一通讯行业工作的时候，原来在联通新用户（存费送机、购机送费、单开户）、老用户等等以前都是通过一个或几个字段的状态标志进行区别，后来业务发展，发现这样太复杂，后来做了一个政策层级的分类，统一规范。在处理前面数据的时候，对以前的数据进行修复处理，以保证与后续的数据统计方式一致。否则区别两个统计方式。

因为实际业务过程中无法规范而产生的数据质量问题：

问题举例：在一服装制造行业工作的时候，来统计产品的实际工时，因为是A产品完工、B产品新生产，在这一交接阶段，同时进行生产，无法正确的统计实际的生产工时，这是正常的实际情况。

解决方案：后与业务部门沟通，将当天的实际工时根据当天完工产品的理论工价来按比例分配，这样对统计分析虽然会有不真实的情况，但也是能相对真实。

所以碰到问题的时候，可以是否可以折中处理，只要不完全违背统计分析的原则，还要以考虑相应的处理方式。

（4）结合项目中使用

## 题目60： hive数据仓库的设计，项目中分了几层，每层有什么意义

1. 问题分析

考官考的是公司中数仓的真实使用场景，和面试者的理解。

1. 核心问题讲解

数据仓库的数据来源于不同的源数据，并提供多样的数据应用，数据自下而

上流入数据仓库后向上层开放应用，而数据仓库只是中间集成化数据管理的一个

平台。

 源数据层（ODS）：此层数据无任何更改，直接沿用外围系统数据结构和数据，

不对外开放；为临时存储层，是接口数据的临时存储区域，为后一步的数据

处理做准备。

 数据仓库层（DW）：也称为细节层，DW 层的数据应该是一致的、准确的、干

净的数据，即对源系统数据进行了清洗（去除了杂质）后的数据。

 数据应用层（DA 或 APP）：前端应用直接读取的数据源；根据报表、专题分析

需求而计算生成的数据。

数据仓库从各数据源获取数据及在数据仓库内的数据转换和流动都可以认

为是 ETL（抽取 Extra, 转化 Transfer, 装载 Load）的过程，ETL 是数据仓库的

流水线，也可以认为是数据仓库的血液，它维系着数据仓库中数据的新陈代谢，

而数据仓库日常的管理和维护工作的大部分精力就是保持 ETL 的正常和稳定。

为什么要对数据仓库分层？

用空间换时间，通过大量的预处理来提升应用系统的用户体验（效率），因

此数据仓库会存在大量冗余的数据；不分层的话，如果源业务系统的业务规则发

生变化将会影响整个数据清洗过程，工作量巨大。

通过数据分层管理可以简化数据清洗的过程，因为把原来一步的工作分到了

多个步骤去完成，相当于把一个复杂的工作拆成了多个简单的工作，把一个大的

黑盒变成了一个白盒，每一层的处理逻辑都相对简单和容易理解，这样我们比较

容易保证每一个步骤的正确性，当数据发生错误的时候，往往我们只需要局部调

整某个步骤即可

1. 问题扩展

数据仓库元数据管理

元数据（Meta Date），主要记录数据仓库中模型的定义、各层级间的映射关

系、监控数据仓库的数据状态及 ETL 的任务运行状态。一般会通过元数据资料库

（Metadata Repository）来统一地存储和管理元数据，其主要目的是使数据仓

库的设计、部署、操作和管理能达成协同和一致。

元数据是数据仓库管理系统的重要组成部分，元数据管理是企业级数据仓库

中的关键组件，贯穿数据仓库构建的整个过程，直接影响着数据仓库的构建、使

用和维护。

 构建数据仓库的主要步骤之一是 ETL。这时元数据将发挥重要的作用，它定

义了源数据系统到数据仓库的映射、数据转换的规则、数据仓库的逻辑结构、

数据更新的规则、数据导入历史记录以及装载周期等相关内容。数据抽取和

转换的专家以及数据仓库管理员正是通过元数据高效地构建数据仓库。

 用户在使用数据仓库时，通过元数据访问数据，明确数据项的含义以及定制

报表。

 数据仓库的规模及其复杂性离不开正确的元数据管理，包括增加或移除外部

数据源，改变数据清洗方法，控制出错的查询以及安排备份等。

（4）结合项目中使用

## 题目61：hive优化经验

（1）问题分析

考官主要考察你对hive底层的理解，以及你实际的工作能力。 理解hive的实质以及hive在实际处理数据的组件，以及作用，深入理解底层原理，才能处理

hive的一些优化。

（2）核心问题讲解

hive是数据仓库，主要涉及到对海量数据的存储和读取，以及数据的处理。

数据的存储和读取基本是基于hadoop的hdfs，所以要进行的优化就是提高数据的传输

速度，可以通过配置参数（map和reduce阶段），优化hive的性能（如：在map阶段设置task的数量

mapred.min.split.size:通过调整max可以起到调整map数的作用，减小max可以增加map数，

增大max可以减少map数。）。数据的处理就是

hsql，hsql本质上是转换为mapreduce来处理数据，对于性能的优化，就是一些较为

复杂的hsql，实现功能并提高运行效率。（比如大表和小表进行jion，将小表写在前面，通过先将小表读入

内存中，顺序扫描大表，就可以在map端进行jion避免了shuffle）

1. 问题扩展

数据倾斜问题也会影响到我们对于数据处理的时效性，所以因为一些人员开发时语句编写不当或者一些业务问题都可能对我们产生影响；

1. 结合项目中使用

在实际的Hive SQL开发的过程中，Hive SQL 性能的问题上实际上只有一小部分和数据倾斜有关，很多时候，Hive SQL运行慢是由于开发人员对于使用的数据了解不够以及一些不良的习惯引起的。

开发人员需要确定以下几点：

1、 需要计算的指标真的需要从数据仓库公共明细层来自行汇总吗？ 是不是数据公共层团队开发公共汇总层已经可以满足自己的需求？对应大众的、KPI相关的指标等通常设计良好的数据仓库公共层肯定已经包含了，直接使用即可。

2、真的需要扫描那么多分区吗，比如对于销售事务明细表来说，扫描一年的分区和扫描一周的分区所带来的计算、IO开销完全是两个数量级，所耗费时间肯定是不同的，所以开发人员要仔细考虑因为需求，尽量不浪费计算和存储资源。

3、尽量不要使用select \* from your\_table这样的方式，用到哪些列就指定哪些列，另外WHERE条件中尽量添加过滤条件，以去掉无关的行，从而减少整个MapReduce任务宠需要处理、分发的数据量。

4、输入文件不要是大量的小文件，Hive默认的Input Split是128MB（可配置），小文件可先合并成大文件。

## 题目62：hive数据仓库中的建模方式，为什么选择这种建模方式

## 问题分析

考官主要是为了考察 面试人员对于业务的熟悉和理解程度，单讲业务建模类型是相对容易的但是切合业务进行建模就值得我们进行思考了；

## 核心问题讲解

Hive作为数据仓库，同关系型数据库开发过程类似，都需要先进行建模，所谓建模，就是对表之间指定关系方式。建模在hive中大致分为星型、雪花型和星座型。要对建模深入理解，首先需要对hive数仓中的集中表概念进行界定。hive中的表从形态上分内部表、外部表、桶表、分区表。在数据逻辑上划分为维度表和事实表。维度表等价于我们常说的字典表。事实表就是字典表之外的数据表。

1.1 星型

多张维度表，一张事实表，维度表之间没有关系。查询性能要好些,存储有冗余的。星型模型使用的比较多。

1.2 雪花型

雪花型是星型建模的扩展，维度表之间有关系。存储减少冗余，查询性能有损失，需要多级连接。和星型模型的共性就是只有一张是事实表。

1.3 星座型

星座型也是星型模型的扩展，存在多张事实表。

## 问题扩展

扩展性差的时候，有那么一句：当增加新业务的时候，就需要增加相应的表。

## （4）结合项目中使用

比如说某个业务系统的信息没有提供汇总表，而是新增一种新的业务，就需要新增一张数据表。我们统计业务量的时候，就要拿大几十张数据表进行关联，报表执行效率很差，后面就直接在数据仓库上帮源系统做了张新业务的汇总表，提高了后续报表的加工效率，也屏蔽了新增业务品种时对报表的影响。

这里我们可以结合我们能书仓管设计的基本特性来考虑，这里考虑的就是分层的概念：

1. 清晰数据结构
2. 数据血缘追踪
3. 减少重复开发
4. 吧复杂问题简单化
5. 屏蔽原始数据的异常
6. 屏蔽业务的影响，不必改一次业务就需要重新接入

这里我们就可以依照这些概念，我们的哪些基本数据分到数仓层，再结合我们的业务需求逐层构建各个业务上的集市层；可以有效减少我们后期运营维护的各项成本；

## 题目63：分布式数据仓库的整体组织结构

## （1）问题分析

考官主要考察对数据仓库层级关系和作用的理解，以及不同层级数据在数据仓库中的转化

## （2）核心问题讲解

数据仓库的层级分为：

ODS层（源数据层）：用来存放外围系统导入的数据，数据类型包含业务系统数据、网站和app等日志数据，已经其他渠道获得的数据（比如购买的数据），源数据层基本都是贴源数据，除了简单的清洗，一般数据形式和表结构都和数据源保持一致。

DW层：数据仓库层，一般用来存放明细数据，根据不同业务类型将ODS的数据进行关联融合，得到不同业务类型的明细表。明细表可以提供给前端报表直接查询明细使用，也可供后面的数据汇总使用。

DM层：数据集市层，根据不同的分析维度和主题，对数据进行汇总，数据可以是完全汇总，即按需求进行完全汇总成一张表，也可以为了兼容不同维度的分析，将数据数据进行最大限度的压缩汇总，同时兼容分析平台根据不同维度进行查询（例如一份数据要按照地区进行分析，那么数据的颗粒度就只能压缩到地区，不能压缩到省）。

APP层：根据业务需求，将数据粒度高度汇总，基本不需要再进行汇总，数据可以直接应用。

DIM：维度层，DIM层中维护了各个维度数据，如时间、地区、产品、客户属性等维度维度，用于维度分析提供辅助。DIM贯穿这个数仓，任何一个层级都可能用到DIM中的维度表数据。

## （3）问题扩展

数仓建模模型

## （4）结合项目中使用

业务系统产品销售个类型信息数据（如销售流水表、库存表、渠道信息表等）导入ODS，经过融合得到销售明细表（DW），再根据时间维度汇总得到产品销量汇信息汇总表（DM）

## 题目64：数据仓库如何同步，使用什么工具，根据什么进行实时同步

## 问题分析

考官主要的考核点是想要了解我们，是否做过数据迁移工作，如果做过业务数据库和数据仓库进行数据迁移的工作，那么我们基本都会接触到增量数据的迁移问题，因为我们不可能一直以人工的形式为这么多数据总手动迁移的工作；

## 核心问题讲解

这里我们借助开源工具sqoop就可以进行数据的迁移同步工作，不过随着数据量级的不断提升 就会考虑实时的增量数据该要如何处理，这里可以采用canal 解析mysql binlog日志，可以打印解析message生成sql，集成kafka提供生产，供订阅同步数据；，形成同步操作的过程，以达到数据同步的问题；

## 问题扩展

事实上，在生产环境中，系统可能会定期从与业务相关的关系型数据库向Hadoop导入数据，导入数仓后进行后续离线分析。故我们此时不可能再将所有数据重新导一遍，此时我们就需要增量数据导入这一模式了。

增量数据导入分两种，一是基于递增列的增量数据导入（Append方式）。二是基于时间列的增量数据导入（LastModified方式）。

## 结合项目中使用

1.开启mysql binlog日志 安装路径下的my.ini文件添加配置

log-bin=mysql-bin #开启日志

binlog-format=ROW #选择row模式

server\_id=1

开启日志需要重启mysql服务后生效

2.下载canal 地址：https://github.com/alibaba/canal/releases

修改配置conf/canal.properties 全局配置，设置自己的数据库

#################################################

## mysql serverId

canal.instance.mysql.slaveId = 1234

# position info，需要改成自己的数据库信息

canal.instance.master.address = 127.0.0.1:3306

canal.instance.master.journal.name =

canal.instance.master.position =

canal.instance.master.timestamp =

#canal.instance.standby.address =

#canal.instance.standby.journal.name =

#canal.instance.standby.position =

#canal.instance.standby.timestamp =

# username/password，需要改成自己的数据库信息

canal.instance.dbUsername = root

canal.instance.dbPassword = root

canal.instance.defaultDatabaseName = canneltest

canal.instance.connectionCharset = UTF-8

# table regex

canal.instance.filter.regex = .\*\\..\*

#################################################

配置slave，slave可以配置成订阅单表日志，也可以配置订阅多表，或所有的表的日志，也可以配置多个slave，只需修改canal.instance.filter.regex 参数，我这里订阅了所有的表日志

conf\example路径下的

#################################################

## mysql serverId

canal.instance.mysql.slaveId = 1234

# position info

canal.instance.master.address = 127.0.0.1:3306

canal.instance.master.journal.name =

canal.instance.master.position =

canal.instance.master.timestamp =

#canal.instance.standby.address =

#canal.instance.standby.journal.name =

#canal.instance.standby.position =

#canal.instance.standby.timestamp =

# username/password

canal.instance.dbUsername = canal

canal.instance.dbPassword = canal

canal.instance.defaultDatabaseName =canneltest

canal.instance.connectionCharset = UTF-8

# table regex

canal.instance.filter.regex = .\*\\..\*

# table black regex

canal.instance.filter.black.regex =

#################################################

然后用程序需要订阅解析日志，引入相应版本的jar,我这里是1.0.24

<dependency>

<groupId>com.alibaba.otter</groupId>

<artifactId>canal.client</artifactId>

<version>1.0.24</version>

</dependency>

public static void main(String[] args) throws InterruptedException {

// 链接canal

CanalConnector connector = CanalConnectors.newSingleConnector(new InetSocketAddress("127.0.0.1", 11111),

"example", "", "");

connector.connect();

// 开启订阅日志

connector.subscribe();

// 循环订阅

while (true) {

try {

// 每次读取 1000 条

Message message = connector.getWithoutAck(1000);

System.out.println(message);

long batchID = message.getId();

int size = message.getEntries().size();

if (batchID == -1 || size == 0) {

Thread.sleep(1000); // 没有数据

} else {

System.out.println("数据进入===>"+message);

}

connector.ack(batchID);

} catch (Exception e) {

// TODO: handle exception

} finally {

Thread.sleep(1000);

}

}

}

可以打印解析message生成sql，集成kafka提供生产，供订阅同步数据

## 题目65：使用了哪些UDF函数？实现什么功能

## 问题分析

考官主要是考察我们对hive的运用程度，从而进一步考察我们对于hive的认知和使用程度，从而对我们的工作职责和技术能力进行判别，这里我把职责划分为两块：

其一，ETL工程师对于位标准的ETL工程来说，我们的职责更多是把各原始数据进行抽取转换的一个过程，这里我们把不同来源不同格式的数据处理成统一化的结构化数据或半结构化数据后一般就是进行表映射即可，对于hive的UDF函数接触可以较少；

其二，数据分析师这里就我们的工作类型来讲我们是需要紧密结合我们的业务进行数据的消费处理，所以我们的业务复杂度会相对较高。这里我们如果不了解或者不会UDF自定义函数就会是我们的减分项了；

## 核心问题讲解

UDF的话一般是hive提供的函数功能满足不了业务需要，我们就会自己来写UDF函数来辅助完成，对于我们常用的函数而言还是哪些常见的聚合函数，如：count、sum、avg、max、min等，其他的话就要切合我们的需求来进行使用了，不过一般较为常用的有cast(expr as <type>)

可以做： cast('1' as BIGINT) 字符串转换为数字

2、if语句

if(boolean testCondition, T valueTrue, T valueFalseOrNull)

如果 testCondition 为 true 返回 valueTrue， 否则返回 valueFalse 或 Null

如： if(1 == 1, 1, 2) 结果为1

3、case语句

CASE WHEN a THEN b [WHEN c THEN d]\* [ELSE e] END

如：case when a == b then b when a == c then c else d end

4、字符串连接

concat(string1, string2, ...)

如：concat('hello', ' word') 结果为 hello word

5、计算字符串长度

length(string)

如：length('hello') 结果为5

6、查找子串的位置

locate(string substr, string str[, int pos])

如：locate('%', '100%') 返回3

7、聚合某一列数据

collect\_set(col)    会去重

collect\_list(col)    不会去重

## 问题扩展

如何构建UDF？

用户构建的UDF使用过程如下：  
 第一步：继承UDF或者UDAF或者UDTF，实现特定的方法。  
 第二步：将写好的类打包为jar。如hivefirst.jar.  
 第三步：进入到Hive外壳环境中，利用add jar /home/hadoop/hivefirst.jar.注册该jar文件  
 第四步：为该类起一个别名，create temporary function mylength as 'com.whut.StringLength';这里注意UDF只是为这个Hive会话临时定义的。  
 第五步：在select中使用mylength();

## 结合项目中使用

这个 还是结合而业务环境来说的举例说明：

问题

hive无法按照5分钟对日志分组

方案

hive UDF

实现步骤

创建UDF：①extends UDF ②重写evaluate方法

Java代码 收藏代码

package com.xxx.udf;

import java.math.BigDecimal;

import org.apache.hadoop.hive.ql.exec.UDF;

public class UDFTrunc5min extends UDF {

/\*\*

\* truncate 5 minute

\*

\* @param timestamp "1312128177.364"

\* @return

\*/

public String evaluate(String timestamp) {

try {

return new BigDecimal(timestamp).multiply(new BigDecimal("1000")).longValue() / 300000 \* 300000 + "";

} catch (Exception e) {

return null;

}

}

}

将udf打jar包，并上传到hive server上

运行hive cli，执行如下命令：

Java代码 收藏代码

<span style="font-weight: normal;">#添加udf jar

add jar /xxxx/xxx/my\_udf.jar;

#创建临时函数，临时函数，每次打开cli都需要创建function

#如果是系统常用的函数可以发布到hive-exec项目，稍后介绍

create temporary function t5m as 'com.xx.udf.Trunc5min';

#使用udf

select t5m(time) from log group by t5m(time);</span>

## 题目66：根据什么对hive表进行分桶分区，为什么

## 问题分析

考官主要是考察你在日常开发中是否都注重了表查询的效率问题，还有对数据的一些处理方案，依照这些考察你是否具备一名基本大数据人员的基本能力；

## 核心问题讲解

分区

在Hive Select查询中一般会扫描整个表内容，会消耗很多时间做没必要的工作。有时候只需要扫描表中关心的一部分数据，因此建表时引入了partition概念。分区表指的是在创建表时指定的partition的分区空间。   
  
Hive可以对数据按照某列或者某些列进行分区管理，所谓分区我们可以拿下面的例子进行解释。   
当前互联网应用每天都要存储大量的日志文件，几G、几十G甚至更大都是有可能。存储日志，其中必然有个属性是日志产生的日期。在产生分区时，就可以按照日志产生的日期列进行划分。把每一天的日志当作一个分区。   
将数据组织成分区，主要可以提高数据的查询速度。至于用户存储的每一条记录到底放到哪个分区，由用户决定。即用户在加载数据的时候必须显示的指定该部分数据放到哪个分区。

分桶

对于每一个表（table）或者分区， Hive可以进一步组织成桶，也就是说桶是更为细粒度的数据范围划分。Hive也是 针对某一列进行桶的组织。Hive采用对列值哈希，然后除以桶的个数求余的方式决定该条记录存放在哪个桶当中。

把表（或者分区）组织成桶（Bucket）有两个理由：

1）获得更高的查询处理效率。桶为表加上了额外的结构，Hive 在处理有些查询时能利用这个结构。具体而言，连接两个在（包含连接列的）相同列上划分了桶的表，可以使用 Map 端连接 （Map-side join）高效的实现。比如JOIN操作。对于JOIN操作两个表有一个相同的列，如果对这两个表都进行了桶操作。那么将保存相同列值的桶进行JOIN操作就可以，可以大大较少JOIN的数据量。

2）使取样（sampling）更高效。在处理大规模数据集时，在开发和修改查询的阶段，如果能在数据集的一小部分数据上试运行查询，会带来很多方便。

## 问题扩展

如何创建分桶分区？

1 create table student(id INT, age INT, name STRING)

2 partitioned by(stat\_date STRING)

3 clustered by(id) sorted by(age) into 2 buckets

4 row format delimited fields terminated by ',';

5

6 create table student1(id INT, age INT, name STRING)

7 partitioned by(stat\_date STRING)

8 clustered by(id) sorted by(age) into 2 buckets

9 row format delimited fields terminated by ',';

## 结合项目中使用

其实对于数据体量清晰且对于数据业务了解清楚后依照基本规范选择进行分区还是分桶即可；

## 题目67：如何确定起始来源数据？是否有做过元数据管理

## （1）问题分析

考官主要是考核面试人员是否有实际想项目经验，是否熟悉整个一套大数据架构流程。此处主要是考核对数据源的确定

## （2）核心问题讲解

如何确定起始数据来源：

在大数据架构中数据来源比较多样化，例如通过埋点获取用户点击数据或者日志数据，这些数据的采集也可有多种形式，例如通过flume

、ELK；数据源也可以是企业的历史数据，一般存储在hdfs或者hbase，这些都可以作为大数据架构中的数据源。对于确定数据源的形式

就主要看使用架构中数据从何处来即可。

是否做过元数据管理：

在实际项目中一般使用mysql进行元数据的管理

## （3）问题扩展

企业中数据采集的常用形式：

很多互联网企业都有自己的海量数据采集工具，

多用于系统日志采集，如Hadoop的Chukwa，Cloudera的Flume，Facebook的Scribe等，

这些工具均采用分布式架构，能满足每秒数百MB的日志数据采集和传输需求

## （4）结合项目中使用

实数数据平台架构：

flume+kafka+spark+mysql

在这个架构基础中，flume用于采集日志数据作为数据源，在将数据不断发送到kafka，通过实时对数据进行计算处理，计算出例如pv、uv的值，最终将计算值下沉到mysql中

## 题目68：什么是数据仓库，数据集市、事实表、维度表、OLAP

## （1）问题分析

考察你对于这些概念和它们之间关系的理解。逻辑上存在但物理上可能不存在的。

数据仓库：是支持数据存储和进行大批量数据分析的数据环境。

数据集市：按照需求进行处理的后的数据，直接面向决策分析的数据集合在一起。

事实表：事实表其实质就是通过各种维度和一些指标值得组合来确定一个事实的，

比如通过时间维度，地域组织维度，指标值可以去确定在某时某地的一些指标值怎么样的事实。

事实表的每一条数据都是几条维度表的数据和指标值交汇而得到的。

维度表：维度表可以看成是用户用来分析一个事实的窗口，

它里面的数据应该是对事实的各个方面描述，比如时间维度表，它里面的数据就是一些日，周，月，季，年，日期等数据，

维度表只能是事实表的一个分析角度。

OLAP:联机分析处理

## （3）问题扩展

分区表、（是否分区）

分桶表、（是否分桶）

外部表、内部表、（位置）

Text表、ORC表、CSV表（表的存储格式）

## （4）结合项目中使用

在项目中创建的表，并不是单一的某类表，都是一些属性的聚合，比如外部分区表，内部分桶表等，需要根据原始数据

和业务需求确定可以创建什么类型的表。

# Hadoop

## 题目69：哪个程序通常与 NameNode 在一个节点启动？（1）问题分析

考官主要是考察面试者是否真实做过数据开发的工作，因为改点是一名大数据开发工作者最基本要知道的问题；

## 核心问题讲解

jobtracker通常与 NameNode 在一个节点启动

## hadoop的集群是基于master/slave模式，namenode和jobtracker属于master，datanode和tasktracker属于slave；

JobTracker对应于NameNode,TaskTracker对应于DataNode。

## （3）问题扩展

HDFS不太适合小文件的存储 ，这里面提到了文档块的概念，同本地文件系统一样，HDFS也是按块存储的，只不过块的大小设置的相对大一些，默认为128M。如果一个文件不足128M，那么它只存储在一个块中，而且并不会占用128M的磁盘空间，

这一点需要注意，HDFS不适用于小文件存储的原因并不是因为小文件消耗磁盘空间，而是因为小文件占用了太多的块信息，每个文档块的元数据是会存储在namenode的内存里的，因此当文档块较多的时候会十分消耗namenode的内存

## （4）结合项目中使用

NameNode主要是用来保存HDFS的元数据信息，比如命名空间信息，块信息等。当它运行的时候，这些信息是存在内存中的。但是这些信息也可以持久化到磁盘上。

生产集群namenode Full GC 告警频繁

bdp生产集群文件数量达到1.9亿，namenode当前内存64G，已使用约57G，内存不足，GC严重

处理

主机内存共128G，当前namenode内存为64GB，除namenode，resourcemanager，ZK，journalnode，ZKFC等进程已分配的内存外，剩余总内存约40G。

修改namenode JVM内存参数，将内存由64GB改为90GB，并分发;

检查当前active namenode为nn2;

重启standby namenode (nn1),确定已修改的参数已生效;

通过命令“hdfs haadmin -failover nn2 nn1”将active namenode切换至nn1;

待nn1为active,确定日志正常，且正常提供服务后，重启nn2 namenode

确定nn2 UI页面及日志正常，及已修改的参数已生效。

可选优化方向

文件数量达到1.9亿，导致namenode元数据量过大，可合并小文件以减少namenode内存使用，避免Full GC。

在core-site.xml文件中修改ha.health-monitor.rpc-timeout.ms参数值，来扩大zkfc监控检查超时时间。

## 题目70：你所知道的hadoop调度器，并简要说明其工作方法？

## （1）问题分析

## 考官主要是考察面试者对于hadoop个组件的了解程度，

## 增分点：并考察能否结合组件的熟悉程度与具体需求做平衡，做出一个好的技术选型；

## （2）核心问题讲解

## 比较流行的三种调度器有：默认调度器FIFO，计算能力调度器Capacity Scheduler，公平调度器Fair Scheduler 1) 默认调度器FIFO hadoop中默认的调度器，采用先进先出的原则 2) 计算能力调度器Capacity Scheduler 选择占用资源小，优先级高的先执行 3) 公平调度器Fair Scheduler 同一队列中的作业公平共享队列中所有资源

## 问题扩展

资源隔离问题

YARN对内存资源和CPU资源采用了不同的资源隔离方案。对于内存资源，它是一种限制性资源，它的量的大小直接决定应用程序的死活，因为应用程序到达内存限制，会发生OOM，就会被杀死。CPU资源一般用Cgroups进行资源控制，Cgroups控制资源测试可以参见这篇博文[Cgroups控制cpu，内存，io示例](http://www.cnblogs.com/yanghuahui/p/3751826.html)，内存资源隔离除Cgroups之外提供了另外一个更灵活的方案，就是线程监控方案。

默认情况下YARN采用线程监控的方案控制内存使用，采用这种机制的原因有两点：

1.Java创建子进程采用了“fork()+exec()”的方案，子进程启动的瞬间，它使用的内存量和父进程一致。一个进程使用的内存量可能瞬间翻倍，然后又降下来，采用线程监控的方法可防止这种情况下导致的swap操作。

2.通常情况下，Hadoop任务运行在独立的Java虚拟机中，可以达到资源隔离的目的。Hadoop Streaming是Hadoop提供的一个编程工具，它允许用户使用任何可执行文件或者脚本文件作为Mapper和Reducer，通过Hadoop Streaming编写的MapReduce应用程序中每个任务可以由不同的编程语言环境组成，这难以通过创建单独的虚拟机达到资源隔离的效果。

综上，为了获取更加灵活的资源控制效果，Hadoop对内存的资源隔离采用线程监控方案。解决方案具体如下：

1.linux系统的/proc/<pid>/stat文件，实时的反应进程树使用的内存总量，可以基于此判断任务粒度的内存使用量是否超过设定的最大值。getconf PAGESIZE可以获取page大小。

2.为了避免JVM的“fork()+exec()”模型引发的误杀操作，Hadoop赋予每个进程”年龄”属性，并规定刚启动进程的年龄是1，监控线程每更新一次，各个进程年龄加1，在此基础上，选择被杀死进程组的标准如下：如果一个进程组中所有的进程（年龄大于0）总内存超过用户设置的最大值的两倍，或者所有年龄大于1的进程总内存量超过用户设置最大值，则认为该进程组过量使用内存，就将其kill掉。

这种细粒度，更加灵活的线程监控资源隔离方案，还是值得学习与称道的，记录于此，以后设计系统可以参考。

## （4）结合项目中使用

开发环境下调度器考虑的因素

　　1、作业优先级。作业的优先级越高，它能够获取的资源（slot数目)也越多。Hadoop 提供了5种作业优先级，分别为 VERY\_HIGH、HIGH、NORMAL、 LOW、VERY\_LOW，通过mapreduce.job.priority属性来设置。

　　2、作业提交时间。顾名思义，作业提交的时间越早，就越先执行。

　　3、作业所在队列的资源限制。调度器可以分为多个队列，不同的产品线放到不同的队列里运行。不同的队列可以设置一个边缘限制，这样不同的队列有自己独立的资源，不会出现抢占和滥用资源的情况

。

## 题目76：说一下spark的shuffle阶段和mr的shuffle阶段分别是什么？区别是什么？

问题分析：

对hadoop和spark的shullfe过程理解。

不管是mr的shuffle过程，还是spark的shuffle过程，答案都比较多，考验面试者总结能力。

Spark的shuffle和Mr的shuffle也是经典的对比问题。

核心答案讲解：

mr的shuffle 分为map的shuffle和reduce 的shuffle。

Map的Shuffle

数据存到hdfs中是以块进行存储的，每一个块对应一个分片，maptask就是从分片中获取数据的

在某个节点上启动了map Task,map Task读取是通过k-v来读取的,读取的数据会放到环形缓存区，这样做的目的是为了防止IO的访问次数,然后环形缓存区的内存达到一定的阀值的

时候会把文件益写到磁盘，溢出的各种小文件会合并成一个大文件，这个合并的过程中会进行排序，这个排序叫做归并排序

map阶段会涉及到

1.sort排序(默认按字典排序)

2.合并(combiner合并)

3.文件合并(merage 合并 总共有三种，默认是内存到磁盘)

4.压缩（设置压缩就会执行）

reduce Shuffle

归并排序完成后reduce端会拉取map端的数据，拉取的这个过程叫做copy过程，拉取的数据合并成一个文件，GroupComparator(默认,这个我们也可以自定义)是专门对文件夹里面的key进行分组

然后就形成k-List(v1,v2,v3)的形式，然后reduce经过业务处理，最终输出到hdfs，如果设置压缩就会执行，不设置则不执行

reduce阶段会涉及到：

1.sort排序

2.分组（将相同的key的value放到一个容器的过程）

3.merage文件合并

Spark shuffle

与MapReduce完全不一样的是，MapReduce它必须将所有的数据都写入本地磁盘文件以后，才能启动reduce操作，来拉取数据。为什么？因为mapreduce要实现默认的根据key的排序！所以要排序，肯定得写完所有数据，才能排序，然后reduce来拉取。

    但是Spark不需要，spark默认情况下，是不会对数据进行排序的。因此ShuffleMapTask每写入一点数据，ResultTask就可以拉取一点数据，然后在本地执行我们定义的聚合函数和算子，进行计算。

    spark这种机制的好处在于，速度比mapreduce快多了。但是也有一个问题，mapreduce提供的reduce，是可以处理每个key对应的value上的，很方便。但是spark中，由于这种实时拉取的机制，因此提供不了，直接处理key对应的values的算子，只能通过groupByKey，先shuffle，有一个MapPartitionsRDD，然后用map算子，来处理每个key对应的values。就没有mapreduce的计算模型那么方便。

问题扩展

那Spark 程序的shuffle调优。

那Mr程序shuffle怎么调优。

结合项目中使用

项目中，了解shuffle过程，遇到一些性能问题可以参考。

## 题目77：Combiner，partition作用，如何设置Compression

(1)问题分析：

Mr程序的基本问题，需要掌握。

(2)核心答案讲解：

Partition作用

（一）对partition的理解

partition意思为分开，划分。它分割map每个节点的结果，按照key分别映射给不同的reduce，也是可以自定义的。其实可以理解归类。也可以理解为根据key或value及reduce的数量来决定当前的这对输出数据最终应该交由哪个reduce task处理

partition的作用就是把这些数据归类。每个map任务会针对输出进行分区，及对每一个reduce任务建立一个分区。划分分区由用户定义的partition函数控制，默认使用哈希函数来划分分区。

HashPartitioner是mapreduce的默认partitioner。计算方法是

which reducer=(key.hashCode() & Integer.MAX\_VALUE) % numReduceTasks，得到当前的目的reducer。

（二）partition过程

1，计算(key，value)所属与的分区。

当map输出的时候，写入缓存之前，会调用partition函数，计算出数据所属的分区，并且把这个元数据存储起来。

2，把属与同一分区的数据合并在一起。

当数据达到溢出的条件时（即达到溢出比例，启动线程准备写入文件前），读取缓存中的数据和分区元数据，然后把属与同一分区的数据合并到一起。

（三）我们还可以自定义partition

Combiner作用

（一）对combiner的理解

combiner其实属于优化方案，由于带宽限制，应该尽量map和reduce之间的数据传输数量。它在Map端把同一个key的键值对合并在一起并计算，计算规则与reduce一致，所以combiner也可以看作特殊的Reducer。

执行combiner操作要求开发者必须在程序中设置了combiner（程序中通过job.setCombinerClass(myCombine.class)自定义combiner操作）

（二）哪里使用combiner？

1，map输出数据根据分区排序完成后，在写入文件之前会执行一次combine操作（前提是作业中设置了这个操作）；

2，如果map输出比较大，溢出文件个数大于3（此值可以通过属性min.num.spills.for.combine配置）时，在merge的过程（多个spill文件合并为一个大文件）中前还会执行combiner操作；

（三）注意事项

不是每种作业都可以做combiner操作的，只有满足以下条件才可以：

1，combiner只应该用于那种Reduce的输入key/value与输出key/value类型完全一致，因为combine本质上就是reduce操作。

2，计算逻辑上，combine操作后不会影响计算结果，像求和，最大值就不会影响，求平均值就影响了。

MapReduce设置compression

压缩compress：目的是为了压缩map输出的 结果数据，减少网络IO和磁盘IO流

常见的压缩算法：Snappy、Lz4,Lzo

配置压缩：MapReduce ，配置 map端的输出有压缩

mapreduce.map.output.compress = true

mapreduce.map.output.compress.codec="org.apache.hadoop.io.compress.Lz4Codec"

(3)问题扩展

如何自定义partition。

如何自定义combiner。

(4)结合项目中使用

在hadoop离线项目中流量汇总的时候使用了自定义分区。

## 题目78：Hadoop参数调优，性能优化

问题分析：

调优问题、优化问题，是面试常见问题。

核心答案讲解：

Hadoop参数调优

一、 hdfs-site.xml 配置文件

1、 dfs.blocksize

参数：hadoop文件块大小

描述：新文件的默认块大小，以字节为单位，默认 134217728 字节。

可以使用以下后缀(大小写不敏感):k(kilo)、m(mega)、g(giga)、t(tera)、p(peta)、e(exa)来指定大小(如128k、512m、1g等)，

或者以字节为单位提供完整的大小。

2、 dfs.namenode.handler.count

参数：namenode的服务器线程数

描述：NameNode有一个工作线程池用来处理客户端的远程过程调用及集群守护进程的调用。处理程序数量越多意味着要更大的池来处理来自不同DataNode的并发心跳以及客户端并发的元数据操作。对于大集群或者有大量客户端的集群来说，通常需要增大参数dfs.namenode.handler.count的默认值10。设置该值的一般原则是将其设置为集群大小的自然对数乘以20，即20logN，N为集群大小。

3、 dfs.datanode.balance.bandwidthPerSec

参数： datanode 平衡带宽

描述：指定每个datanode可以利用每秒字节数来平衡目标的最大带宽。

4、 dfs.replication

参数：块副本数

描述：默认的块复制。可以在创建文件时指定复制的实际数量。如果在create time中没有指定复制，则使用默认值3。

5、dfs.datanode.max.transfer.threads

参数：datanode 最大传输线程数

描述：指定用于传输数据进出DN的最大线程数。集群中如果不一致，会造成数据分布不均。

二、 core-site.xml 配置文件

1、 io.file.buffer.size

参数：文件的缓冲区大小

描述：用于顺序文件的缓冲区大小。这个缓冲区的大小应该是硬件页面大小的倍数(在Intel x86上是4096)，它决定了在读写操作中缓冲了多少数据。SequenceFiles 读取和写入操作的缓存区大小，还有map的输出都用到了这个缓冲区容量， 可减少 I/O 次数。建议设定为 64KB 到 128KB

三、 yarn-site.xml 配置文件

1、 yarn.nodemanager.resource.memory-mb

参数：该节点 nodemanager 资源池内存

描述：NodeManager节点上可使用的物理内存总量，默认是8192（MB），根据节点所能分配的最大的内存进行分配即可，注意为操作系统与其他服务预留资源。

2、yarn.nodemanager.resource.cpu-vcores

参数：该节点 有多少cpu加入资源池 ， 默认值为8

描述：表示该节点上YARN可使用的虚拟CPU个数，默认是8，注意，目前推荐将该值设值为与物理CPU核数数目相同。如果你的节点CPU核数不够8个，则需要调减小这个值，而YARN不会智能的探测节点的物理CPU总数。

·

1.Jvm重用

JVM重用不是指同一Job的两个或两个以上的Task同时运行于同一JVM上，而是 N个Task按顺序在同一个Jvm上运行，即省去了Jvm关闭和再重启的时间。N值可以在Hadoop的mapre-site.xml文件mapreduce.job.jvm.numtasks(默认1)属性进行设置。也可在hive的执行设置：set mapred.job.reuse.jvm.num.tasks=10;(默认1)

一个TT最多可以同时运行的task数目由mapred-site.xml中mapreduce.tasktracker.map.tasks.maximum 和 mapreduce.tasktracker.reduce.tasks.maximum设置。其他方法，如在JobClient端通过命令行：-D mapred.tasktracker.map.tasks.maximum=number 或者 conf.set("mapred.tasktracker.map.tasks.maximum","number")设置都是【无效的】。

2.哪些因素影响作业的运行效率?

mapper的数量：尽量将输入数据切分成数据块的整数倍。如有太多小文件，则考虑CombineFileInputFormat

reducer的数量：为了达到最高性能，集群中reducer数应该略小于reducer的任务槽数

combiner使用: 充分使用合并函数减少map和reduce之间传递的数据量，combiner在map后运行

中间值的压缩：对map输出值进行压缩减少到reduce前的传递量conf.setCompressMapOutput(true)和setMapOutputCompressorClass(GzipCodec.class)

自定义Writable：如果使用自定义的Writable对象或自定义的comparator，则必须确保已实现RawComparator

调整shuffle参数：MapReduce的shuffle过程可以对一些内存管理的参数进行调整，以弥补性能不足

3.避免context.write(new Text(),new Text())

提倡key.set(); value.set(); output.collect(key,value);

前者会产生大量的Text对象，使用完后Java垃圾回收器会花费大量的时间去收集这些对象

4.使用DistributedCache加载文件

比如配置文件，词典，共享文件，避免使用static变量

5.充分使用Combiner + Parttitioner + Comparator

Combiner: 对map任务进行本地聚合

Parttitioner：合适的Parttitioner避免reduce端负载不均

Comparator：二次排序

6.Uber模式

是Hadoop2.0中实现的一种针对MR小作业的优化机制。即如果作业足够小，则所有task在一个jvm（mrappmaster）中完成要比为每个task启动一个container更划算。下面是该机制的相关参数，这些参数均为客户端配置。

总开关

•mapreduce.job.ubertask.enable 默认值：false

•mapreduce.job.ubertask.maxmaps 最大map数，默认值：9

•mapreduce.job.ubertask.maxreduces 最大reduce数，默认值：1 （社区2.2.0只支持0或1个reduce）

•mapreduce.job.ubertask.maxbytes 最大输入字节数，默认值：默认的blocksize，即64MB

map或reduce的内存需求不大于appmaster的内存需求

•mapreduce.map.memory.mb (默认值：0) <= yarn.app.mapreduce.am.resource.mb （默认值：1536）

•mapreduce.reduce.memory.mb (默认值：0) <= yarn.app.mapreduce.am.resource.mb （默认值：1536）

map或reduce的CPU需求不大于appmaster的CPU需求

•mapreduce.map.cpu.vcores (默认值：1) <= yarn.app.mapreduce.am.resource.cpu-vcores (默认值：1)

•mapreduce.reduce.cpu.vcores (默认值：1) <= yarn.app.mapreduce.am.resource.cpu-vcores (默认值：1)

•mapreduce.job.map.class 不继承于 org.apache.hadoop.mapreduce.lib.chain.ChainMapper

•mapreduce.job.reduce.class 不继承于 org.apache.hadoop.mapreduce.lib.chain.ChainReducer

问题扩展

工作中具体用过哪些调优方式。

结合项目中使用

项目中合理设置reduceTask个数。合理设置快大小。

## 题目79：hadoop三种运行模式的适用场景

问题分析：

掌握hadoop运行的3种模式。

既然有3种模式，我们要学会对比。

核心答案讲解：

    1.独立（本地）运行模式：无需任何守护进程，所有的程序都运行在同一个JVM上执行。在独立模式下调试MR程序非常高效方便。所以一般该模式主要是在学习或者开发阶段调试使用 。

          2.伪分布式模式：  Hadoop守护进程运行在本地机器上，模拟一个小规模的集群，换句话说，可以配置一台机器的Hadoop集群,伪分布式是完全分布式的一个特例。

          3.完全分布式模式：Hadoop守护进程运行在一个集群上。

问题扩展

工作中用哪种模式。

(4)结合项目中使用

测试的时候，用独立模式。

上线的时候，我们用分布式模式。

## 题目80：Hadoop中的Sequence File（序列文件）是什么？

问题分析：

Hadoop可以存储多种文件格式。Sequence File这个格式需要知道。

核心答案讲解：

1.1.sequenceFile文件是Hadoop用来存储二进制形式的[Key,Value]对而设计的一种平面文件(Flat File)。

1.2.可以把SequenceFile当做是一个容器，把所有的文件打包到SequenceFile类中可以高效的对小文件进行存储和处理。

1.3.SequenceFile文件并不按照其存储的Key进行排序存储，SequenceFile的内部类Writer提供了append功能。

1.4.SequenceFile中的Key和Value可以是任意类型Writable或者是自定义Writable。

1.5.在存储结构上，SequenceFile主要由一个Header后跟多条Record组成，Header主要包含了Key classname，value classname，存储压缩算法，用户自定义元数据等信息，此外，还包含了一些同步标识，用于快速定位到记录的边界。每条Record以键值对的方式进行存储，用来表示它的字符数组可以一次解析成：记录的长度、Key的长度、Key值和value值，并且Value值的结构取决于该记录是否被压缩

(3)问题扩展

项目中是否用过Sequence File这个格式。

结合项目中使用

项目中，存储小的二进制文件，用Sequence File这个格式