2020大数据面试题真题总结(附答案)

原创 大数据私房菜 大数据私房菜 1周前

一.Hadoop

1.hdfs写流程
2.hdfs读流程
3.hdfs的体系结构
4.一个datanode 宕机,怎么一个流程恢复
5.hadoop 的 namenode 宕机,怎么解决
6.namenode对元数据的管理
7.元数据的checkpoint
8.yarn资源调度流程
9.hadoop中combiner和partition的作用
10.用mapreduce怎么处理数据倾斜问题?
11.shuffle 阶段,你怎么理解的
12.Mapreduce 的 map 数量 和 reduce 数量是由什么决定的 ,怎么配置
13.MapReduce优化经验
14.分别举例什么情况要使用 combiner，什么情况不使用?
15.MR运行流程解析
16.简单描述一下HDFS的系统架构，怎么保证数据安全?
17.在通过客户端向hdfs中写数据的时候，如果某一台机器宕机了，会怎么处理
18.Hadoop优化有哪些方面
19.大量数据求topN(写出mapreduce的实现思路)
20.列出正常工作的hadoop集群中hadoop都分别启动哪些进程以及他们的作用
21.Hadoop总job和Tasks之间的区别是什么?
22.Hadoop高可用HA模式
23.简要描述安装配置一个hadoop集群的步骤
24.fsimage和edit的区别
25.yarn的三大调度策略
26.hadoop的shell命令用的多吗?,说出一些常用的

二.Hive

1.大表join小表产生的问题，怎么解决?
2.udf udaf udtf区别
3.hive有哪些保存元数据的方式，个有什么特点。
4.hive内部表和外部表的区别
5.生产环境中为什么建议使用外部表?
6.insert into 和 override write区别?

7.hive的判断函数有哪些
8.简单描述一下HIVE的功能？用hive创建表有几种方式？hive表有几种?
9.线上业务每天产生的业务日志（压缩后>=3G），每天需要加载到hive的log表中，将每天产生的业务日志在压缩之后load到hive的log表时，最好使用的压缩算法是哪个,并说明其原因
10.若在hive中建立分区仍不能优化查询效率，建表时如何优化
11.union all和union的区别
12.如何解决hive数据倾斜的问题
13.hive性能优化常用的方法
14.简述delete，drop，truncate的区别
15.四个by的区别
16.Hive 里边字段的分隔符用的什么？为什么用\t？有遇到过字段里 边有\t 的情况吗，怎么处理的？为什么不用 Hive 默认的分隔符，默认的分隔符是什么？
17.分区分桶的区别，为什么要分区
18.mapjoin的原理
19.在hive的row_number中distribute by 和 partition by的区别
20.hive开发中遇到什么问题?
21.什么时候使用内部表,什么时候使用外部表
22.hive都有哪些函数，你平常工作中用到哪些

三.Spark

1.rdd的属性
2.算子分为哪几类(RDD支持哪几种类型的操作)
3.创建rdd的几种方式
4.spark运行流程
5.Spark中coalesce与repartition的区别
6.sortBy 和 sortByKey的区别
7.map和mapPartitions的区别
8.数据存入Redis 优先使用map mapPartitions foreach foreachPartions哪个
9.reduceByKey和groupBykey的区别
10.cache和checkPoint的比较
11.spark streaming流式统计单词数量代码
12.简述map和flatMap的区别和应用场景
13.计算曝光数和点击数
14.分别列出几个常用的transformation和action算子
15.按照需求使用spark编写以下程序，要求使用scala语言
16.spark应用程序的执行命令是什么?
17.Spark应用执行有哪些模式，其中哪几种是集群模式
18.请说明spark中广播变量的用途

19.以下代码会报错吗？如果会怎么解决 val arr = new ArrayList[String]; arr.foreach(println)

20.写出你用过的spark中的算子，其中哪些会产生shuffle过程

21.Spark中rdd与partition的区别

22.请写出创建Dateset的几种方式

23.描述一下RDD，DataFrame，DataSet的区别?

24.描述一下Spark中stage是如何划分的？描述一下shuffle的概念

25.Spark 在yarn上运行需要做哪些关键的配置工作？如何kill -个Spark在yarn运行中Application

26.通常来说，Spark与MapReduce相比，Spark运行效率更高。请说明效率更高来源于Spark内置的哪些机制？并请列举常见spark的运行模式?

27.RDD中的数据在哪?

28.如果对RDD进行cache操作后，数据在哪里?

29.Spark中Partition的数量由什么决定

30.Scala里面的函数和方法有什么区别

31.SparkStreaming怎么进行监控?

32.Spark判断Shuffle的依据?

33.Scala有没有多继承？可以实现多继承么?

34.Sparkstreaming和flink做实时处理的区别

35.Sparkcontext的作用

36.Sparkstreaming读取kafka数据为什么选择直连方式

37.离线分析什么时候用sparkcore和sparksql

38.Sparkstreaming实时的数据不丢失的问题

39.简述宽依赖和窄依赖概念，groupByKey,reduceByKey,map,filter,union五种操作哪些会导致宽依赖，哪些会导致窄依赖

40.数据倾斜可能会导致哪些问题，如何监控和排查，在设计之初，要考虑哪些来避免

41.有一千万条短信，有重复，以文本文件的形式保存，一行一条数据，请用五分钟时间，找出重复出现最多的前10条

42.现有一文件，格式如下，请用spark统计每个单词出现的次数

43.共享变量和累加器

44.当 Spark 涉及到数据库的操作时，如何减少 Spark 运行中的数据库连接数?

45.特别大的数据，怎么发送到excutor中?

46.spark调优都做过哪些方面?

47.spark任务为什么会被yarn kill掉?

48.Spark on Yarn作业执行流程？yarn-client和yarn-cluster有什么区别?

49.Flatmap底层编码实现?

四.Kafka

1.Kafka名词解释和工作方式

2.Consumer与topic关系

3.kafka中生产数据的时候，如何保证写入的容错性?

4.如何保证kafka消费者消费数据是全局有序的

5.有两个数据源，一个记录的是广告投放给用户的日志，一个记录用户访问日志，另外还有一个固定的用户基础表记录用户基本信息（比如学历，年龄等等）。现在要分析广告投放对与哪类用户更有效，请采用熟悉的技术描述解决思路。另外如果两个数据源都是实时数据源（比如来自kafka），他们数据在时间上相差5分钟，需要哪些调整来解决实时分析问题?

6.Kafka和SparkStreaing如何集成?

7.列举Kafka的优点，简述Kafka为什么可以做到每秒数十万甚至上百万消息的高效分发?

8.为什么离线分析要用kafka?

9.Kafka怎么进行监控?

10.Kafka与传统的消息队列服务有很么不同

11.Kafka api low-level与high-level有什么区别，使用low-level需要处理哪些细节

12.Kafka的ISR副本同步队列

13.Kafka消息数据积压，Kafka消费能力不足怎么处理?

14.Kafka中的ISR、AR又代表什么?

15.Kafka中的HW、LEO等分别代表什么?

16.哪些情景会造成消息漏消费?

17.当你使用kafka-topics.sh创建了一个topic之后，Kafka背后会执行什么逻辑?

18.topic的分区数可不可以增加？如果可以怎么增加？如果不可以，那又是为什么?

19.topic的分区数可不可以减少？如果可以怎么减少？如果不可以，那又是为什么?

20.Kafka有内部的topic吗？如果有是什么？有什么所用?

21.聊一聊Kafka Controller的作用?

22.失效副本是指什么？有那些应对措施?

五.Hbase

1.Hbase调优

2.hbase的rowkey怎么创建好？列族怎么创建比较好?

3.hbase过滤器实现用途

4.HBase宕机如何处理

5.hive跟hbase的区别是?

6.hbase写流程

7.hbase读流程

8.hbase数据flush过程

9.数据合并过程

10.Hmaster和Hgionserver职责

11.HBase列族和region的关系?
12.请简述Hbase的物理模型是什么
13.请问如果使用Hbase做即席查询，如何设计二级索引
14.如何避免读、写HBaes时访问热点问题?
15.布隆过滤器在HBASE中的应用
16.Hbase是用来干嘛的?什么样的数据会放到hbase

六.数仓

1.维表和宽表的考查（主要考察维表的使用及维度退化手法）
2.数仓表命名规范
3.拉链表的使用场景
4.一亿条数据查的很慢,怎么查快一点
5.有什么维表
6.数据源都有哪些
7.你们最大的表是什么表,数据量多少
8.数仓架构体系
9.数据平台是怎样的，用到了阿里的那一套吗?
10.你了解的调度系统有那些？，你们公司用的是哪种调度系统
11.你们公司数仓底层是怎么抽数据的?
12.为什么datax抽数据要比sqoop 快?
13.埋点数据你们是怎样接入的
14.如果你们业务库的表有更新，你们数仓怎么处理的?
15.能独立搭建数仓吗
16.搭建过CDH 集群吗
17.说一下你们公司的大数据平台架构？你有参与吗?
18.介绍一下你自己的项目和所用的技术
19.对目前的流和批处理的认识？就是谈谈自己的感受
20.你了解那些OLAP 引擎，MPP 知道一些吗？clickHouse 了解一些吗？你自己做过测试性能吗?
21.Kylin 有了解吗？介绍一下原理
22.datax 源码有改造过吗
23.你们数仓的APP 层是怎么对外提供服务的?
24.数据接入进来，你们是怎样规划的，有考虑数据的膨胀问题吗
25.简述拉链表，流水表以及快照表的含义和特点
26.全量表(df),增量表(di),追加表(da)，拉链表(dz)的区别及使用场景
27.你们公司的数仓分层，每一层是怎么处理数据的
28.什么是事实表，什么是维表
29.星型模型和雪花模型
30.数据建模一般有哪几种方式，你们公司是用哪种方式进行数据建模的

31.有没有实际工作中碰到的sql调优的例子，举例说明
32.你觉得数据仓库应该如何搭建，数据规范和标准如何落地
33.如何保证你们公司的数据质量
34.对元数据的理解，元数据管理的意义及应用场景有哪些
35.如何判别模型的好坏，模型设计的原则有哪些
36.对于数据中台的理解，和数据仓库，数据湖的区别
37.对于数据仓库的理解，数据仓库主要为解决什么问题
38.数据仓库模型的理解，数据仓库分层设计的好处是什么
39.数仓主题划分的标准和依据
40.缓慢变化维如何处理，几种方式

七.Flink

1.Flink实时计算时落磁盘吗
2.日活DAU的统计需要注意什么
3.Flink调优
4.Flink的容错是怎么做的
5.Parquet格式的好处？什么时候读的快什么时候读的慢
6.flink 中 checkPoint 为什么状态有保存在内存中这样的机制？为什么要开启 checkPoint?
7.flink保证Exactly_Once的原理?
8.flink的时间形式和窗口形式有几种？有什么区别，你们用在什么场景下的?
9.flink的背压说下?
10.flink的watermark机制说下，以及怎么解决数据乱序的问题?
11.flink on yarn执行流程
12.说一说spark 和flink 的区别

八.Java

1.hashMap底层源码，数据结构
2.写出你用过的设计模式，并举例说明解决的实际问题
3.Java创建线程的几种方式
4.请简述操作系统的线程和进程的区别
5.Java程序出现OutOfMemoryError:unable to create new native thread 的原因可能有哪些？如何分析和解决?
6.采用java或自己熟悉的任何语言分别实现简单版本的线性表和链表，只需实现add,remove方法即可
7.ArrayList和LinkedList的区别
8.JVM 内存分哪几个区，每个区的作用是什么?
9.Java中迭代器和集合的区别?
10.HashMap 和 HashTable 区别
11.线程池使用注意哪些方面?

12.HashMap和TreeMap的区别？TreeMap排序规则?

13.用java实现单例模式

14.使用递归算法求n的阶乘：n! ,语言不限

15.HashMap和Hashtable的区别是什么

16.TreeSet 和 HashSet 区别

17.Stringbuffer 和 Stringbuild 区别

18.Final、Finally、Finalize

19..==和 Equals 区别

20.比较ArrayList，LinkedList的存储特性和读写性能

21.Java 类加载过程

22.java中垃圾收集的方法有哪些?

23.如何判断一个对象是否存活?(或者GC对象的判定方法)

九.Elasticsearch

1.为什么要用es？存进es的数据是什么格式的，怎么查询

十.Flume

1.什么是flume

2.flume运行机制

3.Flume采集数据到Kafka中丢数据怎么办

4.Flume怎么进行监控?

5.Flume的三层架构，collector、agent、storage

十一.Sqoop

1.Sqoop底层运行的任务是什么

2.sqoop的迁移数据的原理

3.Sqoop参数

4.Sqoop导入导出Null存储一致性问题

5.Sqoop数据导出一致性问题

十二.Redis

1.缓存穿透、缓存雪崩、缓存击穿

2.数据类型

3.持久化

4.悲观锁和乐观锁

5.redis 是单线程的，为什么那么快

6.redis的热键问题？怎么解决?

十三.Mysql

1.请写出mysql登录命令，用户名user，密码123456，地址192.168.1.130

2.为什么MySQL的索引要使用B+树而不是其它树形结构?比如B树?

十四.数据结构与算法

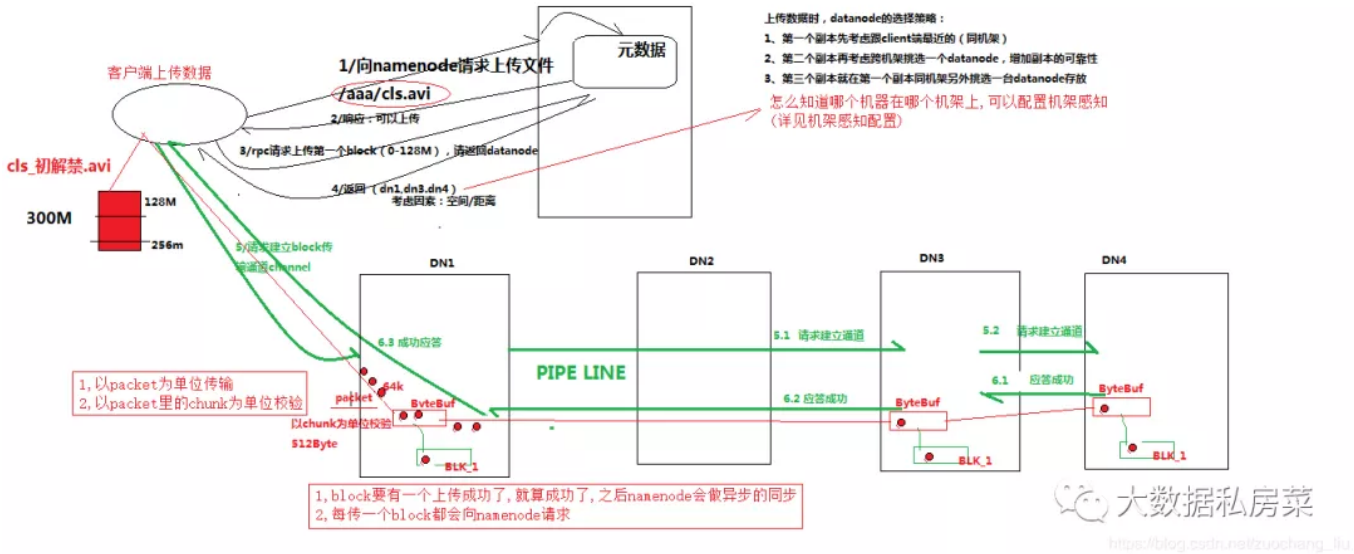
1.二分查找

2.快排
3.归并排序
4.冒泡排序
5.字符串反转
6.Btree简单讲一下
7.动态规划 最大连续子序列和
8.二叉树概念，特点及代码实现
9.链表

十五.Linux

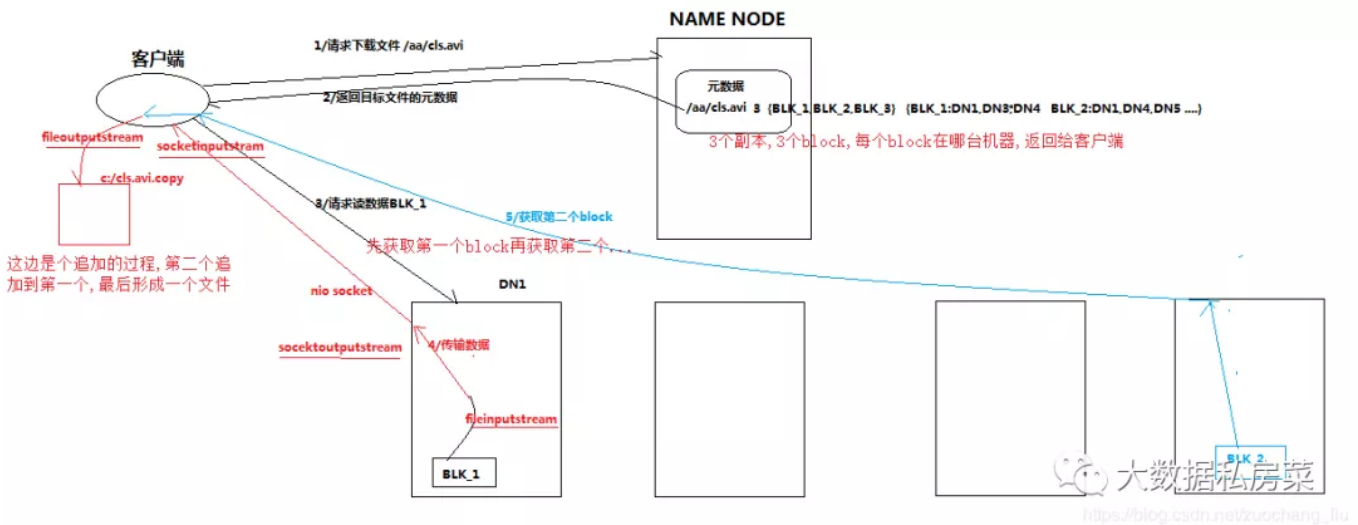
一.Hadoop

1.hdfs写流程



1. 客户端跟namenode通信请求上传文件，namenode检查目标文件是否已存在，父目录是否存在
2. namenode返回是否可以上传
3. client请求第一个 block该传输到哪些datanode服务器上
4. namenode返回3个datanode服务器ABC
5. client请求3台dn中的一台A上传数据（本质上是一个RPC调用，建立pipeline），A收到请求会继续调用B，然后B调用C，将真个pipeline建立完成，逐级返回客户端
6. client开始往A上传第一个block（先从磁盘读取数据放到一个本地内存缓存），以packet为单位，A收到一个packet就会传给B，B传给C；A每传一个packet会放入一个应答队列等待应答
7. 当一个block传输完成之后，client再次请求namenode上传第二个block的服务器。

2.hdfs读流程



1. client跟namenode通信查询元数据，找到文件块所在的datanode服务器
2. 挑选一台datanode（就近原则，然后随机）服务器，请求建立socket流
3. datanode开始发送数据（从磁盘里面读取数据放入流，以packet为单位来做校验）
4. 客户端以packet为单位接收，现在本地缓存，然后写入目标文件

3.hdfs的体系结构

hdfs有namenode、secondraynamenode、datanode组成。为n+1模式

1. NameNode负责管理和记录整个文件系统的元数据
2. DataNode 负责管理用户的文件数据块，文件会按照固定的大小（blocksize）切成若干块后分布式存储在若干台datanode上，每一个文件块可以有多个副本，并存放在不同的datanode上，Datanode会定期向Namenode汇报自身所保存的文件block信息，而namenode则会负责保持文件的副本数量
3. HDFS的内部工作机制对客户端保持透明，客户端请求访问HDFS都是通过向namenode申请来进行
4. secondraynamenode负责合并日志

4.一个datanode 宕机,怎么一个流程恢复

Datanode宕机了后，如果是短暂的宕机，可以实现写好脚本监控，将它启动起来。如果是长时间宕机了，那么datanode上的数据应该已经被备份到其他机器了，那这台datanode就是一台新的datanode了，删除他的所有数据文件和状态文件，重新启动。

5.**hadoop 的 namenode 宕机,怎么解决**

先分析宕机后的损失，宕机后直接导致client无法访问，内存中的元数据丢失，但是硬盘中的元数据应该还存在，如果只是节点挂了，重启即可，如果是机器挂了，重启机器后看节点是否能重启，不能重启就要找到原因修复了。但是最终的解决方案应该是在设计集群的初期就考虑到这个问题，做namenode的HA。

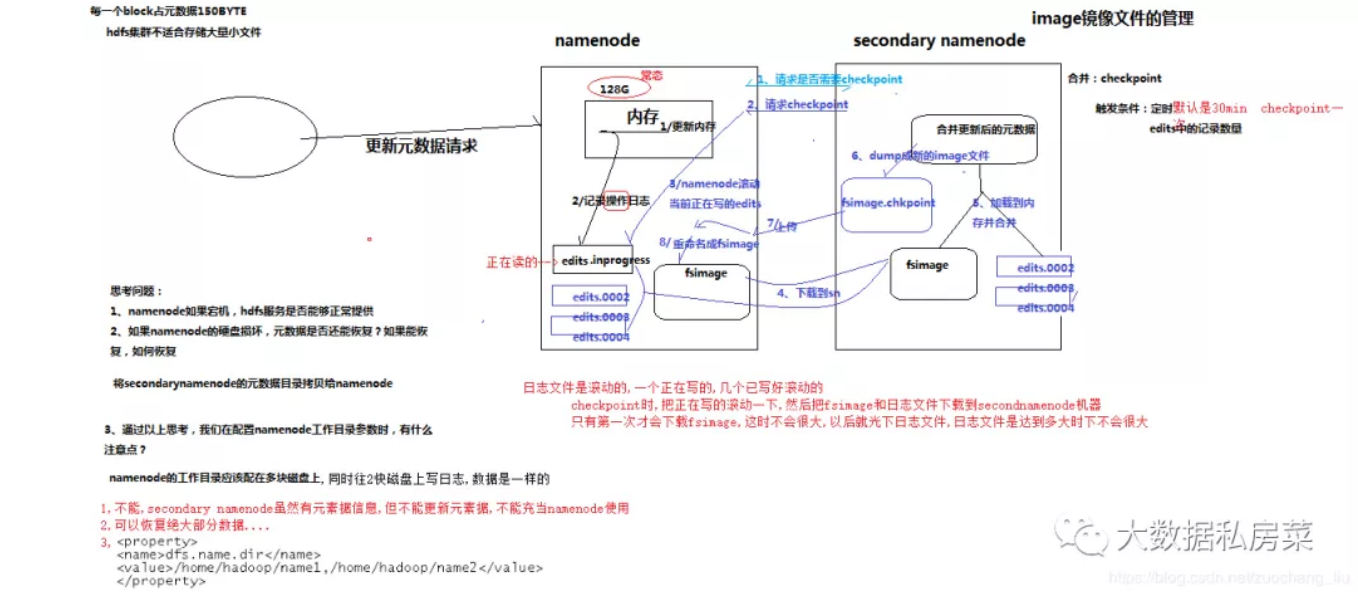
6.namenode对元数据的管理

namenode对数据的管理采用了三种存储形式：

- 内存元数据(NameSystem)
- 磁盘元数据镜像文件(fsimage镜像)
- 数据操作日志文件（可通过日志运算出元数据）(edit日志文件)

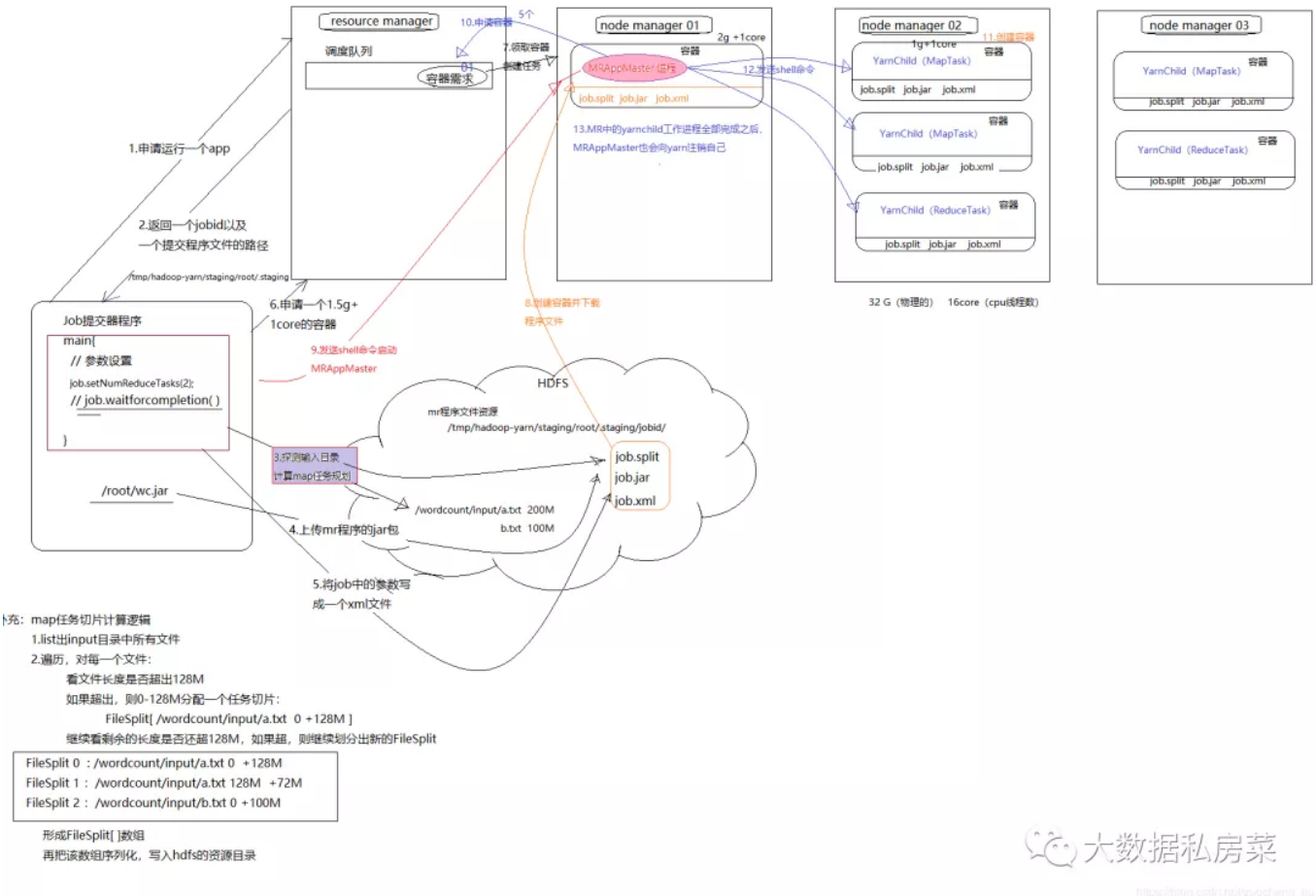
7.元数据的checkpoint

每隔一段时间，会由secondary namenode将namenode上积累的所有edits和一个最新的fsimage下载到本地，并加载到内存进行merge（这个过程称为checkpoint）



namenode和secondary namenode的工作目录存储结构完全相同，所以，当namenode故障退出需要重新恢复时，可以从secondary namenode的工作目录中将fsimage拷贝到namenode的工作目录，以恢复namenode的元数据

8.yarn资源调度流程



1. 用户向YARN中提交应用程序，其中包括ApplicationMaster程序、启动ApplicationMaster的命令、用户程序等。

2. ResourceManager 为该应用程序分配第一个Container， 并与对应的NodeManager 通信，要求它在这个Container 中启动应用程序的ApplicationMaster。
3. ApplicationMaster 首先向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，然后它将为各个任务申请资源，并监控它的运行状态，直到运行结束，即重复步骤4~7。
4. ApplicationMaster 采用轮询的方式通过RPC 协议向ResourceManager 申请和领取资源。
5. 一旦ApplicationMaster 申请到资源后，便与对应的NodeManager 通信，要求它启动任务。
6. NodeManager 为任务设置好运行环境（包括环境变量、JAR 包、二进制程序等）后，将任务启动命令写到一个脚本中，并通过运行该脚本启动任务。
7. 各个任务通过某个RPC 协议向ApplicationMaster 汇报自己的状态和进度，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。在应用程序运行过程中，用户可随时通过RPC 向ApplicationMaster 查询应用程序的当前运行状态。
8. 应用程序运行完成后，ApplicationMaster 向ResourceManager 注销并关闭自己。

9.hadoop中combiner和partition的作用

- combiner是发生在map的最后一个阶段，父类就是Reducer，意义就是对每一个maptask的输出进行局部汇总，以减小网络传输量，缓解网络传输瓶颈，提高reducer的执行效率。
- partition的主要作用将map阶段产生的所有kv对分配给不同的reducer task处理，可以将reduce阶段的处理负载进行分摊

10.用mapreduce怎么处理数据倾斜问题?

数据倾斜：map /reduce程序执行时，reduce节点大部分执行完毕，但是有一个或者几个reduce节点运行很慢，导致整个程序的处理时间很长，这是因为某一个key的条数比其他key多很多（有时是百倍或者千倍之多），这条key所在的reduce节点所处理的数据量比其他节点就大很多，从而导致某几个节点迟迟运行不完，此称之为数据倾斜。

（1）局部聚合加全局聚合。

第一次在 map 阶段对那些导致了数据倾斜的 key 加上 1 到 n 的随机前缀，这样本来相同的 key 也会被分到多个 Reducer 中进行局部聚合，数量就会大大降低。

第二次 mapreduce，去掉 key 的随机前缀，进行全局聚合。

思想：二次 mr，第一次将 key 随机散列到不同 reducer 进行处理达到负载均衡目的。第二次再根据去掉 key 的随机前缀，按原 key 进行 reduce 处理。

这个方法进行两次 mapreduce，性能稍差。

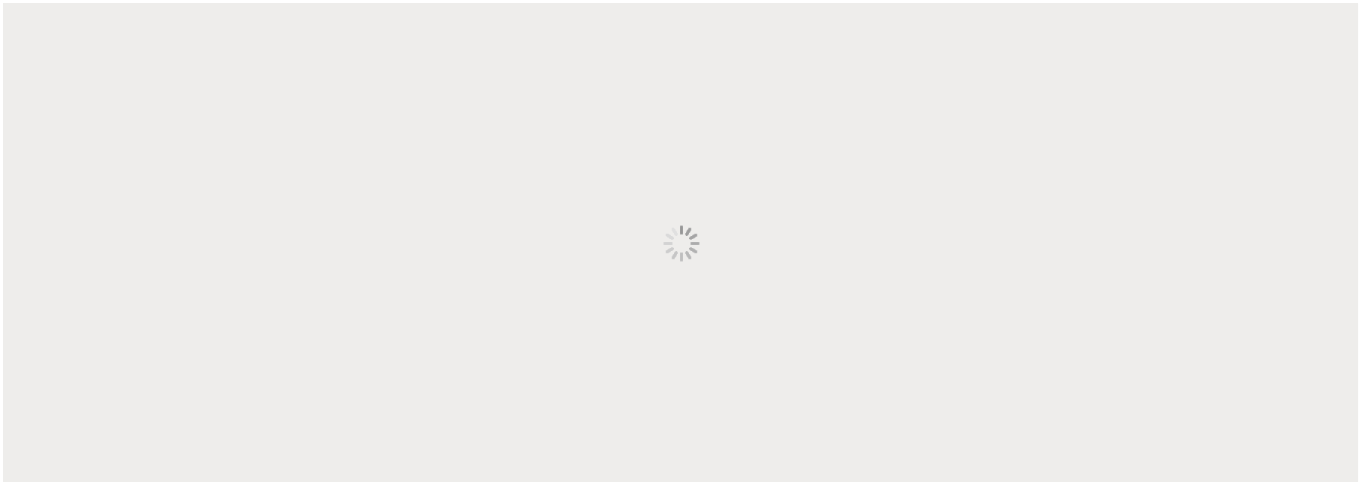
（2）增加 Reducer，提升并行度

JobConf.setNumReduceTasks(int)

（3）实现自定义分区

根据数据分布情况，自定义散列函数，将 key 均匀分配到不同 Reducer

11.**shuffle 阶段,你怎么理解的**



shuffle: 洗牌、发牌——（核心机制：缓存，数据分区，排序，Merge进行局部value的合并）；

具体来说：就是将maptask输出的处理结果数据，分发给reducetask，并在分发的过程中，对数据按key进行了分区和排序；

1）Map 方法之后 Reduce 方法之前这段处理过程叫 Shuffle

2）Map 方法之后，数据首先进入到分区方法，把数据标记好分区，然后把数据发送到 环形缓冲区；环形缓冲区默认大小 100m，环形缓冲区达到 80%时，进行溢写；溢写前对数据进行排序，排序按照对 key 的索引进行字典顺序排序，排序的手段快排；溢写产生大量溢 写文件，需要对溢写文件进行归并排序；对溢写的文件也可以进行 Combiner 操作，前提是汇总操作，求平均值不行。最后将文件按照分区存储到磁盘，等待 Reduce 端拉取。

3）每个 Reduce 拉取 Map 端对应分区的数据。拉取数据后先存储到内存中，内存不够了，再存储到磁盘。拉取完所有数据后，采用归并排序将内存和磁盘中的数据都进行排序。

在进入 Reduce 方法前，可以对数据进行分组操作。

12.Mapreduce 的 map 数量 和 reduce 数量是由什么决定的 ,怎么配置

map的数量由输入切片的数量决定，128M切分一个切片，只要是文件也分为一个切片，有多少个切片就有多少个map Task。

reduce数量自己配置。

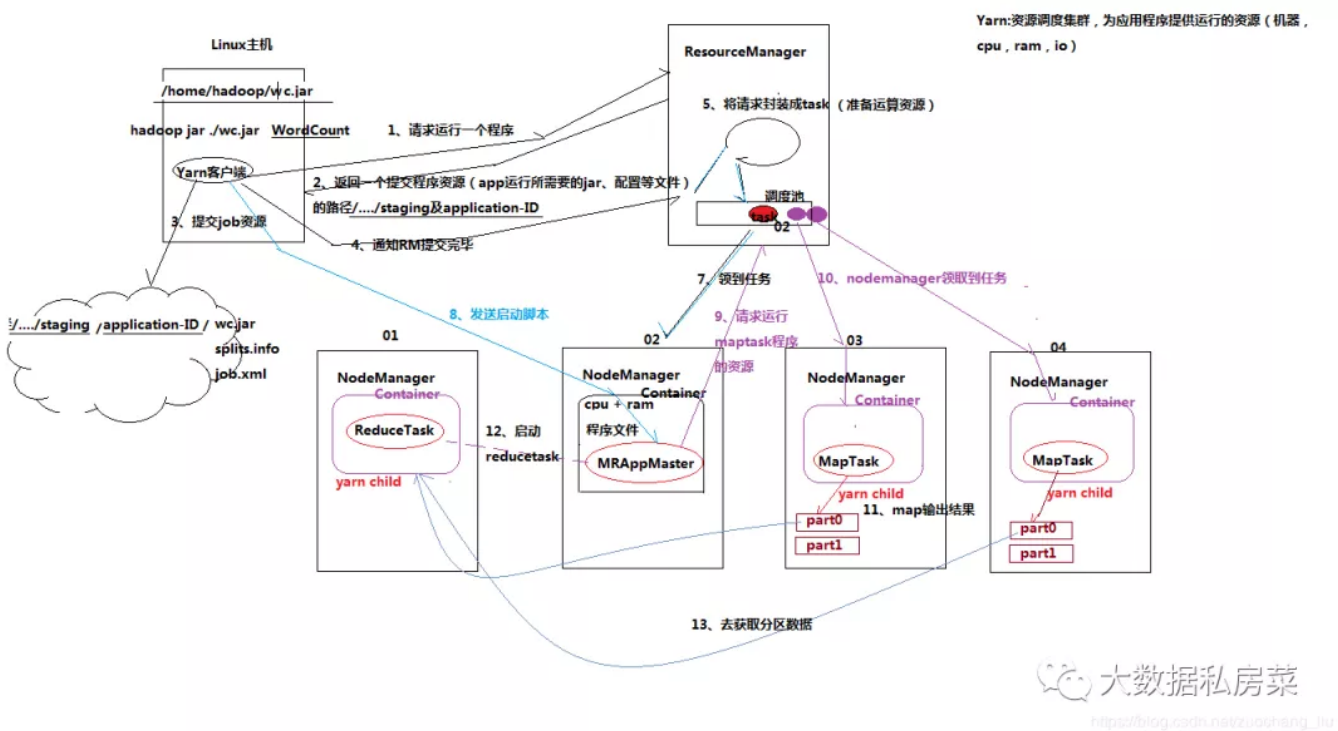
13.**MapReduce优化经验**

1. 设置合理的map和reduce的个数。合理设置blocksize
2. 避免出现数据倾斜
3. combine函数
4. 对数据进行压缩
5. 小文件处理优化：事先合并成大文件，combineTextInputformat，在hdfs上用mapreduce将小文件合并成SequenceFile大文件（key:文件名，value：文件内容）
6. 参数优化

14.分别举例什么情况要使用 combiner，什么情况不使用?

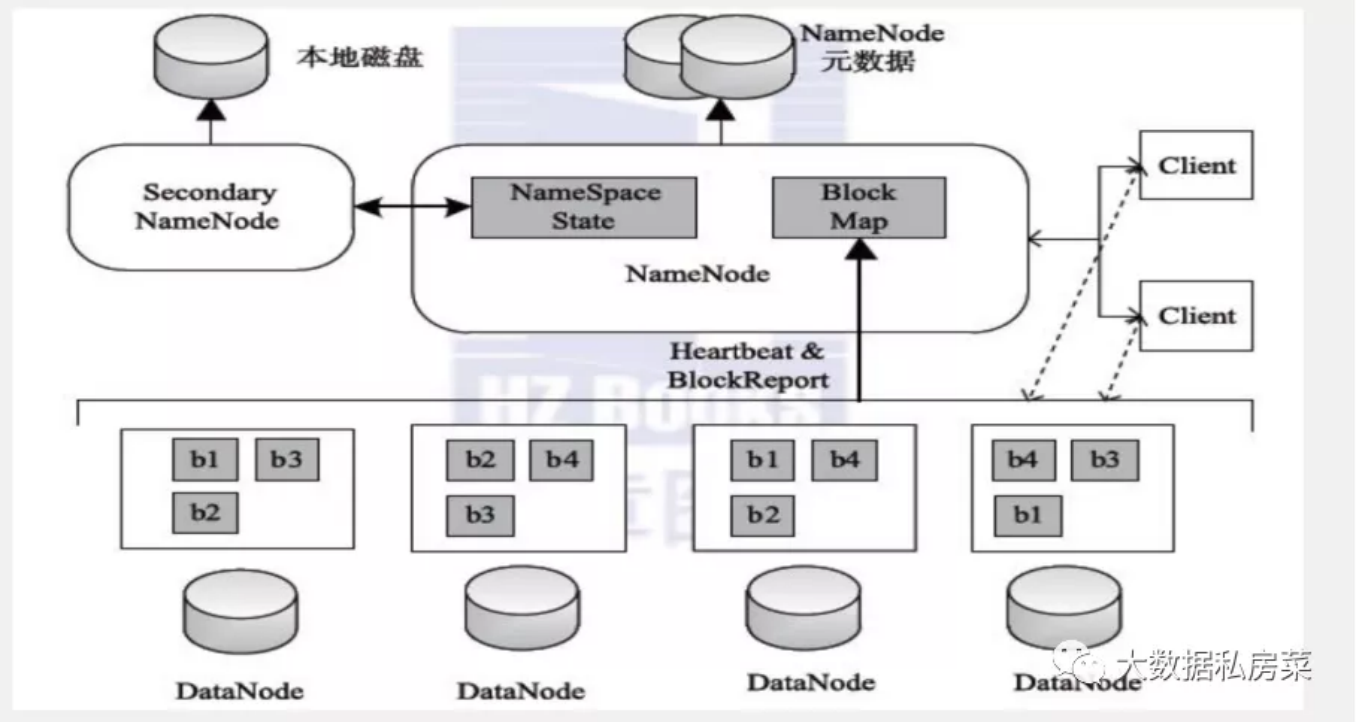
求平均数的时候就不需要用combiner，因为不会减少reduce执行数量。在其他的时候，可以依据情况，使用combiner，来减少map的输出数量，减少拷贝到reduce的文件，从而减轻reduce的压力，节省网络开销，提升执行效率

15.MR运行流程解析



1. 一个mr程序启动的时候，最先启动的是MRAppMaster，MRAppMaster启动后根据本次job的描述信息，计算出需要的maptask实例数量，然后向集群申请机器启动相应数量的maptask进程
2. maptask进程启动之后，根据给定的数据切片范围进行数据处理，主体流程为：
   a. 利用客户指定的inputformat来获取RecordReader读取数据，形成输入KV对
   b. 将输入KV对传递给客户定义的map()方法，做逻辑运算，并将map()方法输出的KV对收集到缓存
   c. 将缓存中的KV对按照K分区排序后不断溢写到磁盘文件
3. MRAppMaster监控到所有maptask进程任务完成之后，会根据客户指定的参数启动相应数量的reducetask进程，并告知reducetask进程要处理的数据范围（数据分区）
4. Reducetask进程启动之后，根据MRAppMaster告知的待处理数据所在位置，从若干台maptask运行所在机器上获取到若干个maptask输出结果文件，并在本地进行重新归并排序，然后按照相同key的KV为一个组，调用客户定义的reduce()方法进行逻辑运算，并收集运算输出的结果KV，然后调用客户指定的outputformat将结果数据输出到外部存储

16.简单描述一下HDFS的系统架构，怎么保证数据安全？



HDFS数据安全性如何保证?

1. 存储在HDFS系统上的文件，会分割成128M大小的block存储在不同的节点上，block的副本数默认3份，也可配置成更多份；
2. 第一个副本一般放置在与client（客户端）所在的同一节点上（若客户端无datanode，则随机放），第二个副本放置到与第一个副本同一机架的不同节点，第三个副本放到不同机架的datanode节点，当取用时遵循就近原则；
3. datanode已block为单位，每3s报告心跳状态，做10min内不报告心跳状态则namenode认为block已死掉，namonode会把其上面的数据备份到其他一个datanode节点上，保证数据的副本数量；
4. datanode会默认每小时把自己节点上的所有块状态信息报告给namenode；
5. 采用safemode模式：datanode会周期性的报告block信息。Namenode会计算block的损坏率，当阀值<0.999f时系统会进入安全模式，HDFS只读不写。HDFS元数据采用secondaryname备份或者HA备份

17.在通过客户端向hdfs中写数据的时候，如果某一台机器宕机了，会怎么处理

在写入的时候不会重新重新分配datanode。如果写入时，一个datanode挂掉，会将已经写入的数据放置到queue的顶部，并将挂掉的datanode移出pipline，将数据写入到剩余的datanode，在写入结束后， namenode会收集datanode的信息，发现此文件的replication没有达到配置的要求（default=3）,然后寻找一个datanode保存副本。

18.Hadoop优化有哪些方面

0）HDFS 小文件影响

（1）影响 NameNode 的寿命，因为文件元数据存储在 NameNode 的内存中

（2）影响计算引擎的任务数量，比如每个小的文件都会生成一个 Map 任务

1）数据输入小文件处理：

（1）合并小文件：对小文件进行归档（Har）、自定义 Inputformat 将小文件存储成 SequenceFile 文件。

（2）采用 ConbinFileInputFormat 来作为输入，解决输入端大量小文件场景。

（3）对于大量小文件 Job，可以开启 JVM 重用。

2）Map 阶段

（1）增大环形缓冲区大小。由 100m 扩大到 200m

（2）增大环形缓冲区溢写的比例。由 80%扩大到 90%

（3）减少对溢写文件的 merge 次数。（10 个文件，一次 20 个 merge）

（4）不影响实际业务的前提下，采用 Combiner 提前合并，减少 I/O。

3）Reduce 阶段

（1）合理设置 Map 和 Reduce 数：两个都不能设置太少，也不能设置太多。太少，会导致 Task 等待，延长处理时间；太多，会导致 Map、Reduce 任务间竞争资源，造成处理超时等错误。

（2）设置 Map、Reduce 共存：调整 slowstart.completedmaps 参数，使 Map 运行到一定程度后，Reduce 也开始运行，减少 Reduce 的等待时间。

（3）规避使用 Reduce，因为 Reduce 在用于连接数据集的时候将会产生大量的网络消耗。

（4）增加每个 Reduce 去 Map 中拿数据的并行数

（5）集群性能可以的前提下，增大 Reduce 端存储数据内存的大小。

4）IO 传输

（1）采用数据压缩的方式，减少网络 IO 的的时间。安装 Snappy 和 LZOP 压缩编码器。

（2）使用 SequenceFile 二进制文件

5）整体

（1）MapTask 默认内存大小为 1G，可以增加 MapTask 内存大小为 4-5g

（2）ReduceTask 默认内存大小为 1G，可以增加 ReduceTask 内存大小为 4-5g

（3）可以增加 MapTask 的 cpu 核数，增加 ReduceTask 的 CPU 核数

（4）增加每个 Container 的 CPU 核数和内存大小

（5）调整每个 Map Task 和 Reduce Task 最大重试次数

19.大量数据求topN(写出mapreduce的实现思路)

20.列出正常工作的hadoop集群中hadoop都分别启动哪些进程以及他们的作用

1.NameNode它是hadoop中的主服务器，管理文件系统名称空间和对集群中存储的文件的访问，保存有metadate。

2.SecondaryNameNode它不是namenode的冗余守护进程，而是提供周期检查点和清理任务。帮助NN合并editslog，减少NN启动时间。

3.DataNode它负责管理连接到节点的存储（一个集群中可以有多个节点）。每个存储数据的节点运行一个datanode守护进程。

4.ResourceManager（JobTracker）JobTracker负责调度DataNode上的工作。每个DataNode有一个TaskTracker，它们执行实际工作。

5.NodeManager（TaskTracker）执行任务

6.DFSZKFailoverController高可用时它负责监控NN的状态，并及时的把状态信息写入ZK。它通过一个独立线程周期性的调用NN上的一个特定接口来获取NN的健康状态。FC也有选择谁作为Active NN的权利，因为最多只有两个节点，目前选择策略还比较简单（先到先得，轮换）。

7.JournalNode 高可用情况下存放namenode的editlog文件.

21.Hadoop总job和Tasks之间的区别是什么?

Job是我们对一个完整的mapreduce程序的抽象封装

Task是job运行时，每一个处理阶段的具体实例，如map task，reduce task，maptask和reduce task都会有多个并发运行的实例

22.**Hadoop高可用HA模式**

HDFS高可用原理：

Hadoop HA（High Available）通过同时配置两个处于Active/Passive模式的Namenode来解决上述问题，状态分别是Active和Standby. Standby Namenode作为热备份，从而允许在机器发生故障时能够快速进行故障转移，同时在日常维护的时候使用优雅的方式进行Namenode切换。Namenode只能配置一主一备，不能多于两个Namenode。

主Namenode处理所有的操作请求（读写），而Standby只是作为slave，维护尽可能同步的状态，使得故障时能够快速切换到Standby。为了使Standby Namenode与Active Namenode数据保持同步，两个Namenode都与一组Journal Node进行通信。当主Namenode进行任务的namespace操作时，都会确保持久会修改日志到Journal Node节点中。Standby Namenode持续监控这些edit，当监测到变化时，将这些修改同步到自己的namespace。

当进行故障转移时，Standby在成为Active Namenode之前，会确保自己已经读取了Journal Node中的所有edit日志，从而保持数据状态与故障发生前一致。

为了确保故障转移能够快速完成，Standby Namenode需要维护最新的Block位置信息，即每个Block副本存放在集群中的哪些节点上。为了达到这一点，**Datanode同时配置主备两个Namenode，并同时发送Block报告和心跳到两台Namenode。**

确保任何时刻只有一个Namenode处于Active状态非常重要，否则可能出现数据丢失或者数据损坏。当两台Namenode都认为自己的Active Namenode时，会同时尝试写入数据（不会再去检测和同步数据）。为了防止这种脑裂现象，Journal Nodes只允许一个Namenode写入数据，内部通过维护epoch数来控制，从而安全地进行故障转移。

23.简要描述安装配置一个hadoop集群的步骤

1. 使用root账户登录。
2. 修改IP。
3. 修改Host主机名。
4. 配置SSH免密码登录。
5. 关闭防火墙。
6. 安装JDK。
7. 上传解压Hadoop安装包。

8. 配置Hadoop的核心配置文件hadoop-evn.sh，core-site.xml，mapred-site.xml，hdfs-site.xml，yarn-site.xml
9. 配置hadoop环境变量
10. 格式化hdfs # bin/hadoop namenode -format
11. 启动节点start-all.sh

24.fsimage和edit的区别

fsimage：filesystem image 的简写，文件镜像。

客户端修改文件时候，先更新内存中的metadata信息,只有当对文件操作成功的时候，才会写到editlog。

fsimage是文件meta信息的持久化的检查点。secondary namenode会定期的将fsimage和editlog合并dump成新的fsimage

25.yarn的三大调度策略

FIFO Scheduler把应用按提交的顺序排成一个队列，这是一个先进先出队列，在进行资源分配的时候，先给队列中最头上的应用进行分配资源，待最头上的应用需求满足后再给下一个分配，以此类推。

Capacity（容量）调度器，有一个专门的队列用来运行小任务，但是为小任务专门设置一个队列会预先占用一定的集群资源，这就导致大任务的执行时间会落后于使用FIFO调度器时的时间。

在Fair（公平）调度器中，我们不需要预先占用一定的系统资源，Fair调度器会为所有运行的job动态的调整系统资源。当第一个大job提交时，只有这一个job在运行，此时它获得了所有集群资源；当第二个小任务提交后，Fair调度器会分配一半资源给这个小任务，让这两个任务公平的共享集群资源。

需要注意的是，在下图Fair调度器中，从第二个任务提交到获得资源会有一定的延迟，因为它需要等待第一个任务释放占用的Container。小任务执行完成之后也会释放自己占用的资源，大任务又获得了全部的系统资源。最终的效果就是Fair调度器即得到了高的资源利用率又能保证小任务及时完成。

26.hadoop的shell命令用的多吗?,说出一些常用的

二.Hive

1.大表join小表产生的问题，怎么解决?

mapjoin方案

join因为空值导致长尾(key为空值是用随机值代替)

join因为热点值导致长尾，也可以将热点数据和非热点数据分开处理，最后合并

2.udf udaf udtf区别

UDF操作作用于单个数据行，并且产生一个数据行作为输出。大多数函数都属于这一类（比如数学函数和字符串函数）。

UDAF 接受多个输入数据行，并产生一个输出数据行。像COUNT和MAX这样的函数就是聚集函数。

UDTF 操作作用于单个数据行，并且产生多个数据行-------一个表作为输出。lateral view explore()

简单来说：

UDF:返回对应值，一对一

UDAF：返回聚类值，多对一

UDTF：返回拆分值，一对多

3.hive有哪些保存元数据的方式，个有什么特点。

- 内存数据库derby，安装小，但是数据存在内存，不稳定
- mysql数据库，数据存储模式可以自己设置，持久化好，查看方便。

4.hive内部表和外部表的区别

内部表：加载数据到hive所在的hdfs目录，删除时，元数据和数据文件都删除

外部表：不加载数据到hive所在的hdfs目录，删除时，只删除表结构。

这样外部表相对来说更加安全些，数据组织也更加灵活，方便共享源数据。

5.生产环境中为什么建议使用外部表?

1. 因为外部表不会加载数据到hive，减少数据传输、数据还能共享。
2. hive不会修改数据，所以无需担心数据的损坏
3. 删除表时，只删除表结构、不删除数据。

6.insert into 和 override write区别?

insert into：将数据写到表中

override write：覆盖之前的内容。

7.hive的判断函数有哪些

hive 的条件判断（if、coalesce、case）

8.简单描述一下HIVE的功能？用hive创建表有几种方式？hive表有几种?

hive主要是做离线分析的

hive建表有三种方式

- **直接建表法**
- **查询建表法(**通过AS 查询语句完成建表：将子查询的结果存在新表里，有数据，一般用于中间表**)**
- **like建表法**(会创建结构完全相同的表，但是没有数据)

hive表有2种：内部表和外部表

9.线上业务每天产生的业务日志（压缩后>=3G），每天需要加载到hive的log表中，将每天产生的业务日志在压缩之后load到hive的log表时，最好使用的压缩算法是哪个,并说明其原因

10.若在hive中建立分区仍不能优化查询效率，建表时如何优化

11.union all和union的区别

union 去重

union oll 不去重

12.如何解决hive数据倾斜的问题

1）group by

注：group by 优于 distinct group

情形：group by 维度过小，某值的数量过多

后果：处理某值的 reduce 非常耗时

解决方式：采用 sum() group by 的方式来替换 count(distinct)完成计算。

2）count(distinct)

count(distinct xx)

情形：某特殊值过多

后果：处理此特殊值的 reduce 耗时；只有一个 reduce 任务

解决方式：count distinct 时，将值为空的情况单独处理，比如可以直接过滤空值的行，在最后结果中加 1。如果还有其他计算，需要进行 group by，可以先将值为空的记录单独处

理，再和其他计算结果进行 union。

3）mapjoin

4）不同数据类型关联产生数据倾斜

情形：比如用户表中 user_id 字段为 int，log 表中 user_id 字段既有 string 类型也有 int 类型。当按照 user_id 进行两个表的 Join 操作时。

后果：处理此特殊值的 reduce 耗时；只有一个 reduce 任务

默认的 Hash 操作会按 int 型的 id 来进行分配，这样会导致所有 string 类型 id 的记录都分配

到一个 Reducer 中。

解决方式：把数字类型转换成字符串类型

```
select * from users a
left outer join logs b
on a.usr_id = cast(b.user_id as string)
```

5）开启数据倾斜时负载均衡

```
set hive.groupby.skewindata=true;
```

思想：就是先随机分发并处理，再按照 key group by 来分发处理。

操作：当选项设定为 true，生成的查询计划会有两个 MRJob。

第一个 MRJob 中，Map 的输出结果集合会随机分布到 Reduce 中，每个 Reduce 做部分聚合操作，并输出结果，这样处理的结果是相同的 GroupBy Key 有可能被分发到不同的 Reduce 中，从而达到负载均衡的目的；

第二个 MRJob 再根据预处理的数据结果按照 GroupBy Key 分布到 Reduce 中（这个过程可以保证相同的原始 GroupBy Key 被分布到同一个 Reduce 中），最后完成最终的聚合操

作。

点评：它使计算变成了两个 mapreduce，先在第一个中在 shuffle 过程 partition 时随机给 key 打标记，使每个 key 随机均匀分布到各个 reduce 上计算，但是这样只能完成部分计算，因为相同 key 没有分配到相同 reduce 上。

所以需要第二次的 mapreduce,这次就回归正常 shuffle,但是数据分布不均匀的问题在第一次 mapreduce 已经有了很大的改善，因此基本解决数据倾斜。因为大量计算已经在第一次

mr 中随机分布到各个节点完成。

6）控制空值分布

将为空的 key 转变为字符串加随机数或纯随机数，将因空值而造成倾斜的数据分不到多个 Reducer。

注：对于异常值如果不需要的话，最好是提前在 where 条件里过滤掉，这样可以使计算量大大减少

13.hive性能优化常用的方法

1）MapJoin

如果不指定 MapJoin 或者不符合 MapJoin 的条件，那么 Hive 解析器会将 Join 操作转换成 Common Join，即：在 Reduce 阶段完成 join。容易发生数据倾斜。可以用 MapJoin 把小

表全部加载到内存在 map 端进行 join，避免 reducer 处理。

2）行列过滤

列处理：在 SELECT 中，只拿需要的列，如果有，尽量使用分区过滤，少用 SELECT *。

行处理：在分区剪裁中，当使用外关联时，如果将副表的过滤条件写在 Where 后面，那么就会先全表关联，之后再过滤。

3）列式存储

4）采用分区技术

5）合理设置 Map 数

（1）通常情况下，作业会通过 input 的目录产生一个或者多个 map 任务。

主要的决定因素有：input 的文件总个数，input 的文件大小，集群设置的文件块大小。

（2）是不是 map 数越多越好?

答案是否定的。如果一个任务有很多小文件（远远小于块大小 128m），则每个小文件也会被当做一个块，用一个 map 任务来完成，而一个 map 任务启动和初始化的时间远远大

于逻辑处理的时间，就会造成很大的资源浪费。而且，同时可执行的 map 数是受限的。

（3）是不是保证每个 map 处理接近 128m 的文件块，就高枕无忧了?

答案也是不一定。比如有一个 127m 的文件，正常会用一个 map 去完成，但这个文件只有一个或者两个小字段，却有几千万的记录，如果 map 处理的逻辑比较复杂，用一个 map

任务去做，肯定也比较耗时。

针对上面的问题 2 和 3，我们需要采取两种方式来解决：即减少 map 数和增加 map 数；

6）小文件进行合并

在 Map 执行前合并小文件，减少 Map 数：CombineHiveInputFormat 具有对小文件进行

合并的功能（系统默认的格式）。HiveInputFormat 没有对小文件合并功能。

7）合理设置 Reduce 数

Reduce 个数并不是越多越好

（1）过多的启动和初始化 Reduce 也会消耗时间和资源；

（2）另外，有多少个 Reduce，就会有多少个输出文件，如果生成了很多个小文件，那么如果这些小文件作为下一个任务的输入，则也会出现小文件过多的问题；

在设置 Reduce 个数的时候也需要考虑这两个原则：处理大数据量利用合适的 Reduce 数；使单个 Reduce 任务处理数据量大小要合适；

8）常用参数

// 输出合并小文件

SET hive.merge.mapfiles = true; -- 默认 true，在 map-only 任务结束时合并小文件

SET hive.merge.mapredfiles = true; -- 默认 false，在 map-reduce 任务结束时合并小文件

SET hive.merge.size.per.task = 268435456; -- 默认 256M

SET hive.merge.smallfiles.avgsize = 16777216; -- 当输出文件的平均大小小于 16m 该值时，启动一个独立的 map-reduce 任务进行文件 merge

9）开启 map 端 combiner（不影响最终业务逻辑）

set hive.map.aggr=true；

10）压缩（选择快的）

设置 map 端输出、中间结果压缩。（不完全是解决数据倾斜的问题，但是减少了 IO 读写和网络传输，能提高很多效率）

11）开启 JVM 重用

14.简述delete，drop，truncate的区别

delet 删除数据

drop 删除表

truncate 摧毁表结构并重建

15.四个by的区别

1. Sort By：分区内有序；
2. Order By：全局排序，只有一个 Reducer；
3. Distrbute By：类似 MR 中 Partition，进行分区，结合 sort by 使用。
4. Cluster By：当 Distribute by 和 Sorts by 字段相同时，可以使用 Cluster by 方式。Cluster by 除了具有 Distribute by 的功能外还兼具 Sort by 的功能。但是排序只能是升序排序，不能 指定排序规则为 ASC 或者 DESC。

16.**Hive 里边字段的分隔符用的什么？为什么用\t？有遇到过字段里 边有\t 的情况吗，怎么处理的？为什么不用 Hive 默认的分隔符，默认的分隔符是什么？**

hive 默认的字段分隔符为 ascii 码的控制符\001（^A）,建表的时候用 fields terminated by '\001'

遇到过字段里边有\t 的情况，自定义 InputFormat，替换为其他分隔符再做后续处理

17.分区分桶的区别，为什么要分区

分区表：原来的一个大表存储的时候分成不同的数据目录进行存储。如果说是单分区表，那么在表的目录下就只有一级子目录，如果说是多分区表，那么在表的目录下有多少分区

就有多少级子目录。不管是单分区表，还是多分区表，在表的目录下，和非最终分区目录下是不能直接存储数据文件的

分桶表：原理和hashpartitioner 一样，将hive中的一张表的数据进行归纳分类的时候，归纳分类规则就是hashpartitioner。（需要指定分桶字段，指定分成多少桶）

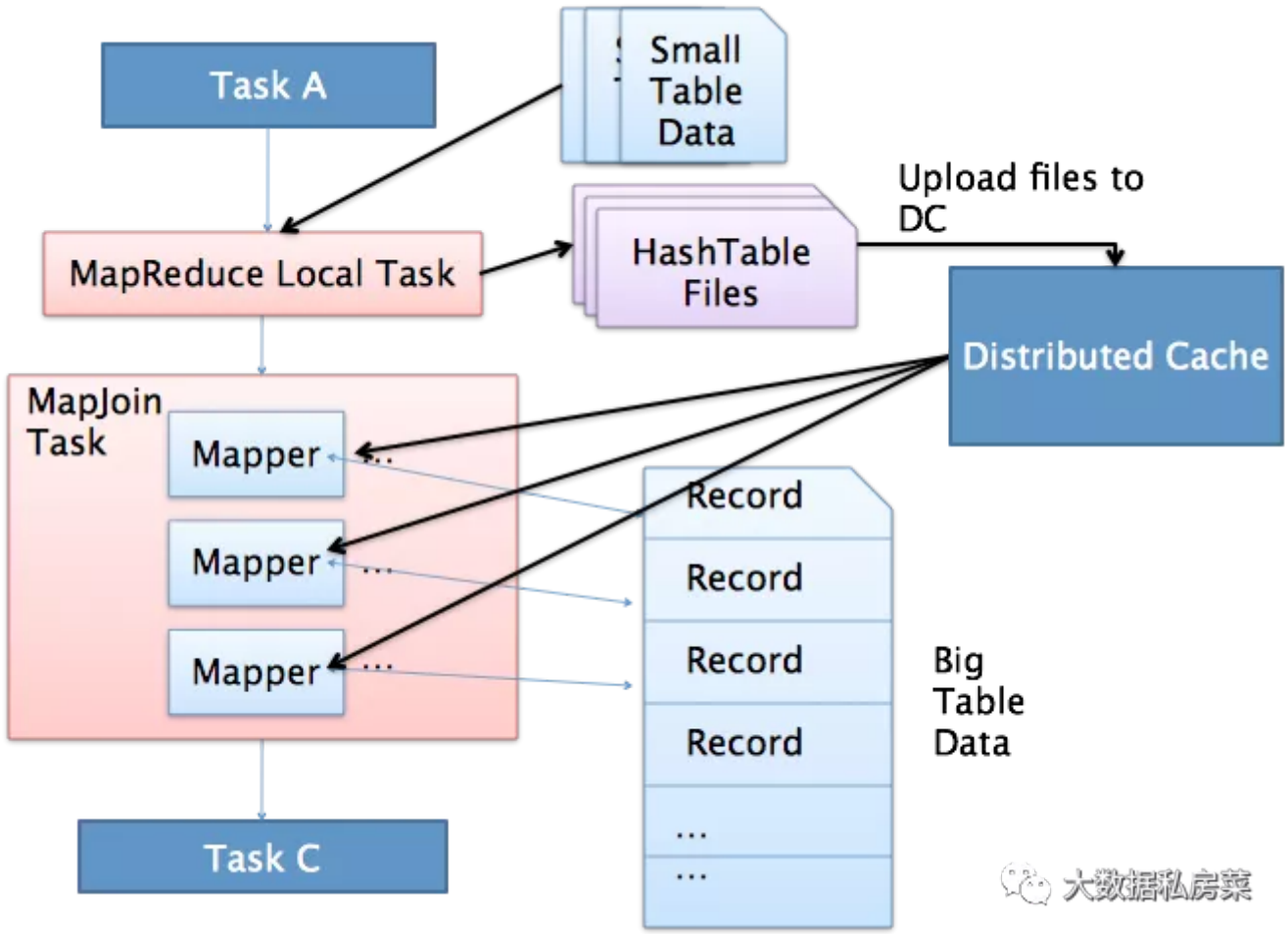
分区表和分桶的区别除了存储的格式不同外，最主要的是作用：

- 分区表：细化数据管理，缩小mapreduce程序 需要**扫描的数据量**。
- 分桶表：**提高join查询的效率**，在一份数据会被经常用来做连接查询的时候建立分桶，分桶字段就是连接字段；**提高采样的效率**。

分区表和分桶的区别除了存储的格式不同外，最主要的是作用：

- 分区表：细化数据管理，缩小mapreduce程序 需要**扫描的数据量**。
- 分桶表：**提高join查询的效率**，在一份数据会被经常用来做连接查询的时候建立分桶，分桶字段就是连接字段；**提高采样的效率**。

18.mapjoin的原理



MapJoin通常用于一个很小的表和一个大表进行join的场景，具体小表有多小，由参数hive.mapjoin.smalltable.filesize来决定，该参数表示小表的总大小，默认值为25000000字节，即25M。

Hive0.7之前，需要使用hint提示 /*+ mapjoin(table) */才会执行MapJoin,否则执行Common Join，但在0.7版本之后，默认自动会转换Map Join，由参数hive.auto.convert.join来控制，默认为true.

假设a表为一张大表，b为小表，并且hive.auto.convert.join=true,那么Hive在执行时候会自动转化为MapJoin。

> MapJoin简单说就是在Map阶段将小表读入内存，顺序扫描大表完成Join。减少昂贵的shuffle操作及reduce操作
> MapJoin分为两个阶段：
> - 通过MapReduce Local Task，将小表读入内存，生成HashTableFiles上传至Distributed Cache中，这里会HashTableFiles进行压缩。
> - MapReduce Job在Map阶段，每个Mapper从Distributed Cache读取HashTableFiles到内存中，顺序扫描大表，在Map阶段直接进行Join，将数据传递给下一个MapReduce任务。

19.在hive的row_number中distribute by 和 partition by的区别

20.hive开发中遇到什么问题?

21.什么时候使用内部表,什么时候使用外部表

每天收集到的ng日志和埋点日志数据,需要做大量的统计数据分析,所以可以使用外部表进行存储，方便数据的共享，并且在对表做操作的时候不会误删原始数据。

在做统计分析时候用到的中间表，结果表可以使用内部表，因为这些数据不需要共享，使用内部表更为合适。并且很多时候分区表我们只需要保留最近3天的数据，用外部表的时候删除分区时无法删除数据

22.hive都有哪些函数，你平常工作中用到哪些

- 数学函数

round(DOUBLE a)
floor(DOUBLE a)
ceil(DOUBLE a)
rand()

- 集合函数

size(Map<K.V>)
map_keys(Map<K.V>)
map_values(Map<K.V>)
array_contains(Array<T>, value)
sort_array(Array<T>)

- 类型转换函数

cast(expr as <type>)

- 日期函数

date_format函数（根据格式整理日期）
date_add、date_sub函数（加减日期）
next_day函数
last_day函数（求当月最后一天日期）
collect_set函数
get_json_object解析json函数

from_unixtime(bigint unixtime, string format)
to_date(string timestamp)
year(string date)
month(string date)
hour(string date)
weekofyear(string date)
datediff(string enddate, string startdate)
add_months(string start_date, int num_months)
date_format(date/timestamp/string ts, string fmt)

- 条件函数

if(boolean testCondition, T valueTrue, T valueFalseOrNull)
nvl(T value, T default_value)
COALESCE(T v1, T v2, ...)
CASE a WHEN b THEN c [WHEN d THEN e]* [ELSE f] END
isnull(a)
isnotnull (a)

- 字符函数

concat(string|binary A, string|binary B...)
concat_ws(string SEP, string A, string B...)
get_json_object(string json_string, string path)
length(string A)
lower(string A) lcase(string A)
parse_url(string urlString, string partToExtract [, string keyToExtract])
regexp_replace(string INITIAL_STRING, string PATTERN, string REPLACEMENT)
reverse(string A)
split(string str, string pat)
substr(string|binary A, int start) substring(string|binary A, int start)

- 聚合函数

count sum min max avg

- 表生成函数

explode(array<Gů Ćc > a)
explode(ARRAY)
json_tuple(jsonStr, k1, k2, ...)
parse_url_tuple(url, p1, p2, ...)
三.Spark
1.rdd的属性

```
* Internally, each RDD is characterized by five main properties:
*
*  - A list of partitions
*  - A function for computing each split
*  - A list of dependencies on other RDDs
*  - Optionally, a Partitioner for key-value RDDs (e.g. to say that the RDD is hash-partitioned)
*  - Optionally, a list of preferred locations to compute each split on (e.g. block locations for
*    an HDFS file)
```


- 一组分片（Partition），即数据集的基本组成单位。对于RDD来说，每个分片都会被一个计算任务处理，并决定并行计算的粒度。用户可以在创建RDD时指定RDD的分片个数，如果没有指定，那么就会采用默认值。默认值就是程序所分配到的CPU Core的数目。
- 一个计算每个分区的函数。Spark中RDD的计算是以分片为单位的，每个RDD都会实现compute函数以达到这个目的。compute函数会对迭代器进行复合，不需要保存每次计算的结果。
- RDD之间的依赖关系。RDD的每次转换都会生成一个新的RDD，所以RDD之间就会形成类似于流水线一样的前后依赖关系。在部分分区数据丢失时，Spark可以通过这个依赖关系重新计算丢失的分区数据，而不是对RDD的所有分区进行重新计算。
- 一个Partitioner，即RDD的分片函数。当前Spark中实现了两种类型的分片函数，一个是基于哈希的HashPartitioner，另外一个是基于范围的RangePartitioner。只有对于于key-value的RDD，才会有Partitioner，非key-value的RDD的Parititioner的值是None。Partitioner函数不但决定了RDD本身的分片数量，也决定了parent RDD Shuffle输出时的分片数量。
- 一个列表，存储存取每个Partition的优先位置（preferred location）。对于一个HDFS文件来说，这个列表保存的就是每个Partition所在的块的位置。按照"移动数据不如移动计算"的理念，Spark在进行任务调度的时候，会尽可能地将计算任务分配到其所要处理数据块的存储位置。

2.算子分为哪几类(RDD支持哪几种类型的操作)

转换（Transformation） 现有的RDD通过转换生成一个新的RDD。lazy模式，延迟执行。

转换函数包括：map，filter，flatMap，groupByKey，reduceByKey，aggregateByKey，union,join, coalesce 等等。

动作（Action） 在RDD上运行计算，并返回结果给驱动程序(Driver)或写入文件系统。

动作操作包括：reduce，collect，count，first，take，countByKey以及foreach等等。

collect 该方法把数据收集到driver端 Array数组类型

所有的transformation只有遇到action才能被执行。

当触发执行action之后，数据类型不再是rdd了，数据就会存储到指定文件系统中，或者直接打印结 果或者收集起来。

3.创建rdd的几种方式

1.集合并行化创建(有数据)

val arr = Array(1,2,3,4,5)

val rdd = sc.parallelize(arr)

val rdd =sc.makeRDD(arr)

2.读取外部文件系统，如hdfs，或者读取本地文件(最常用的方式)(没数据)

val rdd2 = sc.textFile("hdfs://hdp-01:9000/words.txt")

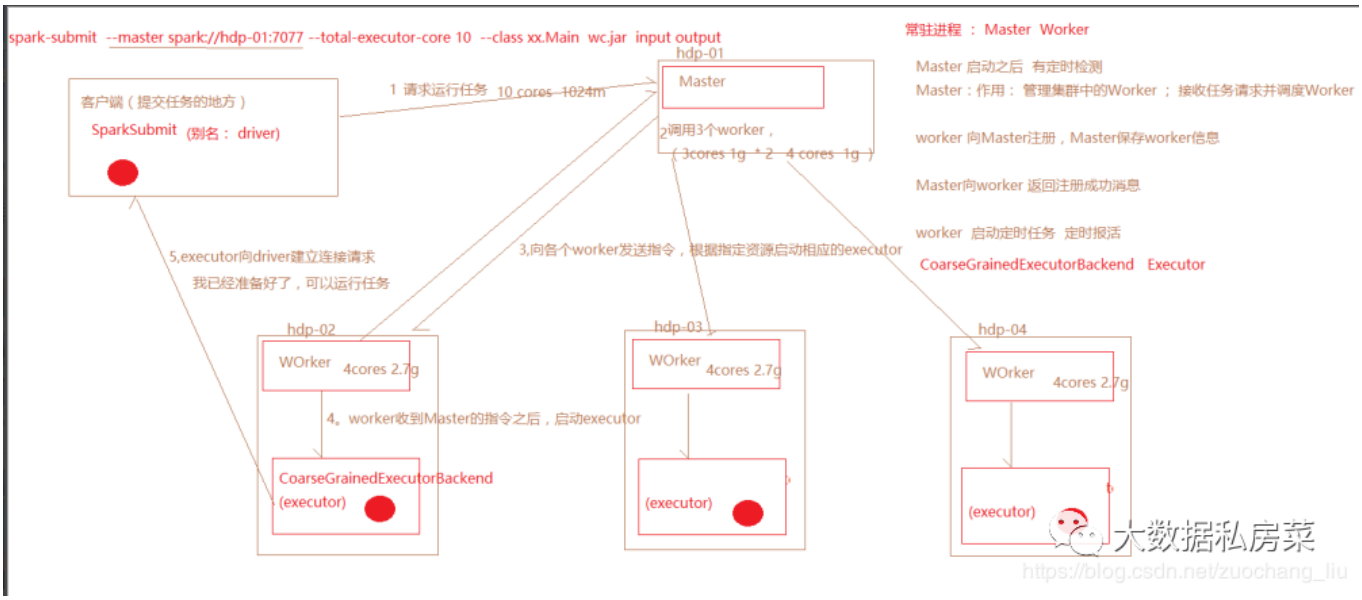
// 读取本地文件

val rdd2 = sc.textFile("file:///root/words.txt")

3.从父RDD转换成新的子RDD

调用Transformation类的方法，生成新的RDD

4.spark运行流程



Worker的功能：定时和master通信；调度并管理自身的executor

executor：由Worker启动的，程序最终在executor中运行，（程序运行的一个容器）

spark-submit命令执行时，会根据master地址去向 Master发送请求，

Master接收到Dirver端的任务请求之后，根据任务的请求资源进行调度，（打散的策略），尽可能的 把任务资源平均分配，然后向WOrker发送指令

Worker收到Master的指令之后，就根据相应的资源，启动executor（cores,memory）

executor会向dirver端建立请求，通知driver，任务已经可以运行了

driver运行任务的时候，会把任务发送到executor中去运行。

5.Spark中coalesce与repartition的区别

1）关系：

两者都是用来改变 RDD 的 partition 数量的，repartition 底层调用的就是 coalesce 方法：coalesce(numPartitions, shuffle = true)

2）区别：

repartition 一定会发生 shuffle，coalesce 根据传入的参数来判断是否发生 shuffle

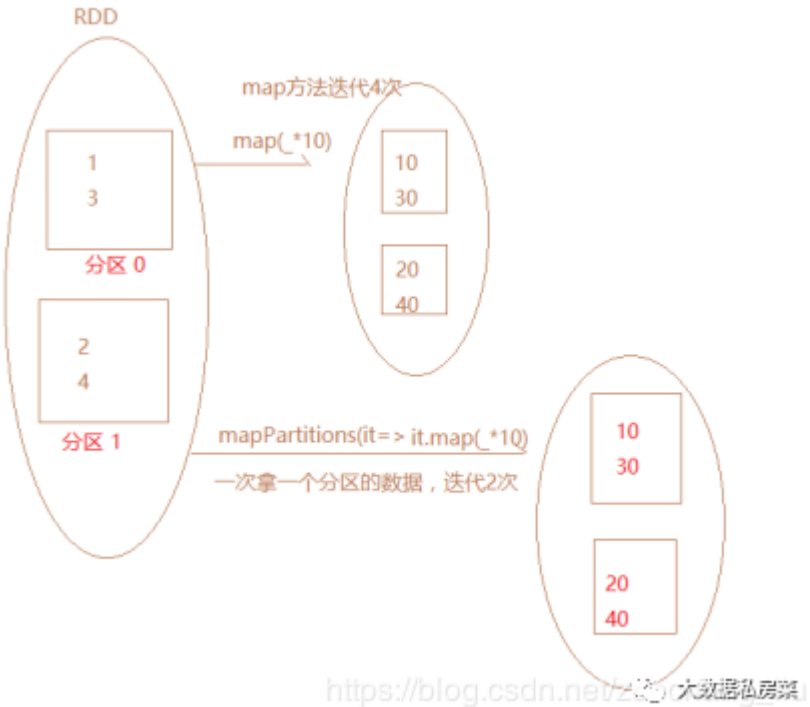
一般情况下增大 rdd 的 partition 数量使用 repartition，减少 partition 数量时使用 coalesce

6.sortBy 和 sortByKey的区别

sortBy既可以作用于RDD[K]， 还可以作用于RDD[(k,v)]

sortByKey 只能作用于 RDD[K,V] 类型上。

7.map和mapPartitions的区别



8.数据存入Redis 优先使用map mapPartitions foreach foreachPartions哪个

使用 foreachPartition

* 1,map mapPartition 是转换类的算子， 有返回值

* 2, 写mysql,redis 的连接

foreach * 100万 100万次的连接

foreachPartions * 200 个分区 200次连接 一个分区中的数据，共用一个连接

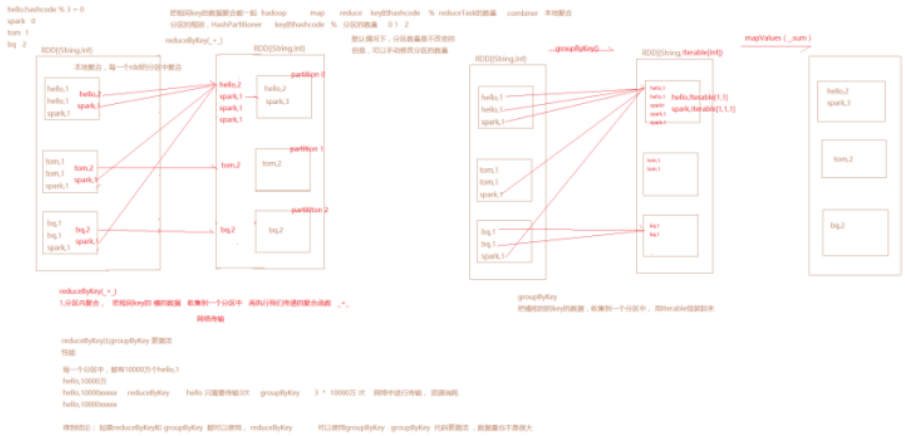
foreachPariti ton 每次迭代一个分区，foreach每次迭代一个元素。

该方法没有返回值，或者Unit

主要作用于，没有返回值类型的操作（打印结果，写入到mysql数据库中）

在写入到redis,mysql的时候，优先使用foreachPartititon

9.**reduceByKey和groupBykey的区别**



reduceByKey会传一个聚合函数， 相当于 groupByKey + mapValues

reduceByKey 会有一个分区内聚合，而groupByKey没有 最核心的区别

结论：reduceByKey有分区内聚合，更高效，优先选择使用reduceByKey。

10.cache和checkPoint的比较

都是做 RDD 持久化的

1.缓存，是在触发action之后，把数据写入到内存或者磁盘中。不会截断血缘关系

（设置缓存级别为memory_only：内存不足，只会部分缓存或者没有缓存，缓存会丢失,memory_and_disk :内存不足，会使用磁盘）

2.checkpoint 也是在触发action之后，执行任务。单独再启动一个job，负责写入数据到hdfs中。（把rdd中的数据，以二进制文本的方式写入到hdfs中，有几个分区，就有几个二进制文件）

3.某一个RDD被checkpoint之后，他的父依赖关系会被删除，血缘关系被截断，该RDD转换成了CheckPointRDD，以后再对该rdd的所有操作，都是从hdfs中的checkpoint的具体目录来读取数据。缓存之后，rdd的依赖关系还是存在的。

11.**spark streaming流式统计单词数量代码**

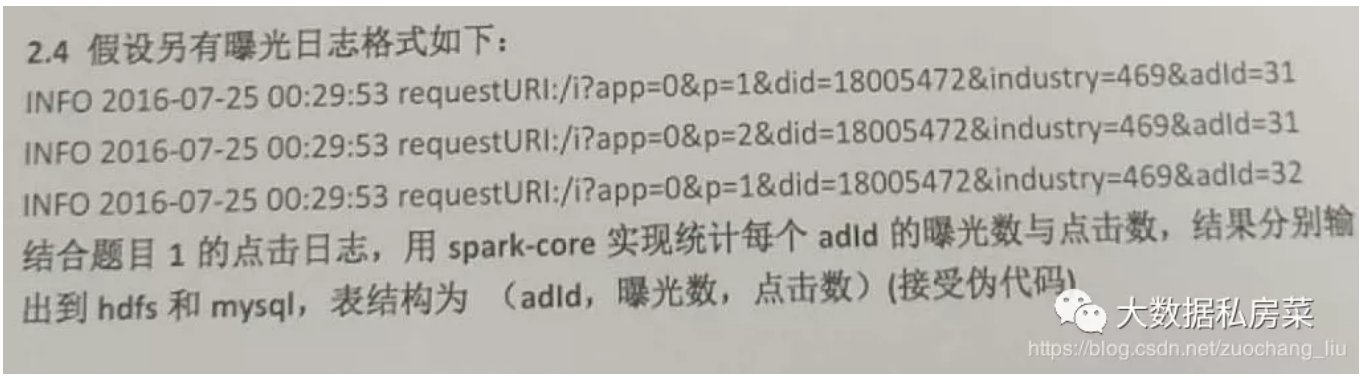
```
object WordCountAll {
  // newValues当前批次的出现的单词次数， runningCount表示之前运行的单词出现的结果
 /* def updateFunction(newValues: Seq[Int], runningCount: Option[Int]):
    val newCount =  newValues.sum + runningCount.getOrElse(0)// 将历史前
    Some(newCount)
  }*/
  /**
    * String : 单词 hello
    * Seq[Int] ：单词在当前批次出现的次数
    * Option[Int] ： 历史结果
    */
  val updateFunc = (iter: Iterator[(String, Seq[Int], Option[Int])]) =>
    //iter.flatMap(it=>Some(it._2.sum + it._3.getOrElse(0)).map(x=>(it._
    iter.flatMap{case(x,y,z)=>Some(y.sum + z.getOrElse(0)).map(m=>(x, m
  }
  // 屏蔽日志
  Logger.getLogger("org.apache").setLevel(Level.ERROR)
  def main(args: Array[String]) {
    // 必须要开启2个以上的线程，一个线程用来接收数据，另外一个线程用来计算
    val conf = new SparkConf().setMaster("local[2]").setAppName("Networ
      // 设置sparkjob计算时所采用的序列化方式
      .set("spark.serializer", "org.apache.spark.serializer.KryoSeriali
      .set("spark.rdd.compress", "true") // 节约大量的内存内容
    // 如果你的程序出现垃圾回收时间过程，可以设置一下java的垃圾回收参数
```

```
    // 同时也会创建sparkContext对象
    // 批次时间 >= 批次处理的总时间（批次数据量，集群的计算节点数量和配置）
    val ssc = new StreamingContext(conf, Seconds(5))

    //做checkpoint 写入共享存储中
    ssc.checkpoint("c://aaa")

    // 创建一个将要连接到 hostname:port 的 DStream，如 localhost:9999
    val lines: ReceiverInputDStream[String] = ssc.socketTextStream("192
    //updateStateByKey结果可以累加但是需要传入一个自定义的累加函数：updateFunc
    val results = lines.flatMap(_.split(" ")).map((_,1)).updateStateByKe
    //打印结果到控制台
    results.print()
    //开始计算
    ssc.start()
    //等待停止
    ssc.awaitTermination()
  }
}
```

12.简述map和flatMap的区别和应用场景

map是对每一个元素进行操作，flatmap是对每一个元素操作后并压平

13.计算曝光数和点击数



14.分别列出几个常用的transformation和action算子

> - 转换算子：map,map,filter,reduceByKey,groupByKey,groupBy
> - 行动算子：foreach，foreachpartition,collect,collectAsMap,take,top,first,count,countByKey

15.按照需求使用spark编写以下程序，要求使用scala语言

当前文件a.txt的格式，请统计每个单词出现的次数

A,b,c

B,b,f,e

```
object WordCount {

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

    val conf = new SparkConf()
      .setAppName(this.getClass.getSimpleName)
      .setMaster("local[*]")
    val sc = new SparkContext(conf)

    var sData: RDD[String] = sc.textFile("a.txt")
    val sortData: RDD[(String, Int)] = sData.flatMap(_.split(",")).map((
    sortData.foreach(print)
  }
}
```

16.spark应用程序的执行命令是什么?

```
/usr/local/spark-current2.3/bin/spark-submit \
--class com.wedoctor.Application \
--master yarn \
--deploy-mode client \
--driver-memory 1g \
--executor-memory 2g \
--queue root.wedw \
--num-executors 200 \
--jars /home/pgxl/liuzc/config-1.3.0.jar,/home/pgxl/liuzc/hadoop-lzo-0.4.20.jar,/home/pgxl/liuzc/elasticsearch-hadoop-hive-2.3.4.jar \
/home/pgxl/liuzc/sen.jar
```

17.Spark应用执行有哪些模式，其中哪几种是集群模式

- 本地local模式
- standalone模式
- spark on yarn模式
- spark on mesos模式

其中，standalone模式，spark on yarn模式，spark on mesos模式是集群模式

18.请说明spark中广播变量的用途

使用广播变量，每个 Executor 的内存中，只驻留一份变量副本，而不是对 每个 task 都传输一次大变量，省了很多的网络传输， 对性能提升具有很大帮助， 而且会通过高效的

广播算法来减少传输代价。

19. 以下代码会报错吗？如果会怎么解决 val arr = new ArrayList[String]; arr.foreach(println)

val arr = new ArrayList[String]; 这里会报错，需要改成 val arr: Array[String] = new Array[String](10)

arr.foreach(println)打印不会报空指针

20.写出你用过的spark中的算子，其中哪些会产生shuffle过程

reduceBykey：

groupByKey：

...ByKey:

21.Spark中rdd与partition的区别

22.请写出创建Dateset的几种方式

23.描述一下RDD，DataFrame，DataSet的区别?

1）RDD

优点:

编译时类型安全

编译时就能检查出类型错误

面向对象的编程风格

直接通过类名点的方式来操作数据

缺点:

序列化和反序列化的性能开销

无论是集群间的通信, 还是 IO 操作都需要对对象的结构和数据进行序列化和反序列化。

GC 的性能开销，频繁的创建和销毁对象, 势必会增加 GC

2）DataFrame

DataFrame 引入了 schema 和 off-heap

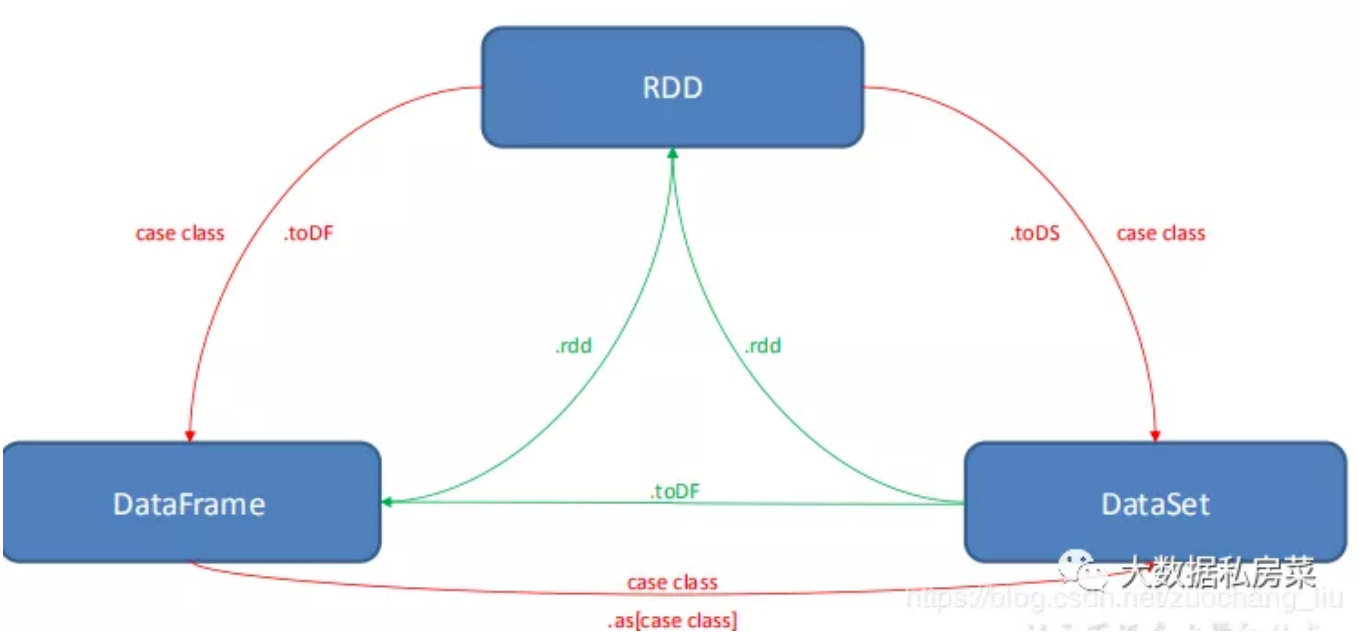
schema : RDD 每一行的数据, 结构都是一样的，这个结构就存储在 schema 中。Spark 通过 schema 就能够读懂数据, 因此在通信和 IO 时就只需要序列化和反序列化数据, 而结构的部分就可以省略了。

3）DataSet

DataSet 结合了 RDD 和 DataFrame 的优点，并带来的一个新的概念 Encoder。

当序列化数据时，Encoder 产生字节码与 off-heap 进行交互，能够达到按需访问数据的效果，而不用反序列化整个对象。Spark 还没有提供自定义 Encoder 的 API，但是未来会加入。

三者之间的转换：



24.描述一下Spark中stage是如何划分的？描述一下shuffle的概念

25.Spark 在yarn上运行需要做哪些关键的配置工作？如何kill -个Spark在yarn运行中Application

26.通常来说，Spark与MapReduce相比，Spark运行效率更高。请说明效率更高来源于Spark内置的哪些机制？并请列举常见spark的运行模式？

27.RDD中的数据在哪?

RDD中的数据在数据源，RDD只是一个抽象的数据集，我们通过对RDD的操作就相当于对数据进行操作。

28.如果对RDD进行cache操作后，数据在哪里?

数据在第一执行cache算子时会被加载到各个Executor进程的内存中，第二次就会直接从内存中读取而不会区磁盘。

29.Spark中Partition的数量由什么决定

和Mr一样，但是Spark默认最少有两个分区。

30.Scala里面的函数和方法有什么区别

31.SparkStreaming怎么进行监控?

32.Spark判断Shuffle的依据?

父RDD的一个分区中的数据有可能被分配到子RDD的多个分区中

33.Scala有没有多继承？可以实现多继承么?

34.Sparkstreaming和flink做实时处理的区别

35.Sparkcontext的作用

36.Sparkstreaming读取kafka数据为什么选择直连方式

37.离线分析什么时候用sparkcore和sparksql

38.Sparkstreaming实时的数据不丢失的问题

39.简述宽依赖和窄依赖概念，groupByKey,reduceByKey,map,filter,union五种操作哪些会导致宽依赖，哪些会导致窄依赖

40.数据倾斜可能会导致哪些问题，如何监控和排查，在设计之初，要考虑哪些来避免

41.有一千万条短信，有重复，以文本文件的形式保存，一行一条数据，请用五分钟时间，找出重复出现最多的前10条

42.现有一文件，格式如下，请用spark统计每个单词出现的次数

```
18619304961，18619304064，186193008，186193009
18619304962，18619304065，186193007，186193008
18619304963，18619304066，186193006，186193010
```

43.共享变量和累加器

累加器（accumulator）是 Spark 中提供的一种分布式的变量机制，其原理类似于mapreduce，即分布式的改变，然后聚合这些改变。累加器的一个常见用途是在调试时对作业执行过程中的事件进行计数。而广播变量用来高效分发较大的对象。

共享变量出现的原因：

通常在向 Spark 传递函数时，比如使用 map() 函数或者用 filter() 传条件时，可以使用驱动器程序中定义的变量，但是集群中运行的每个任务都会得到这些变量的一份新的副本，更新这些副本的值也不会影响驱动器中的对应变量。

Spark 的两个共享变量，累加器与广播变量，分别为结果聚合与广播这两种常见的通信模式突破了这一限制。

44.当 Spark 涉及到数据库的操作时，如何减少 Spark 运行中的数据库连接数?

使用 foreachPartition 代替 foreach，在 foreachPartition 内获取数据库的连接。

45.特别大的数据，怎么发送到excutor中?

46.spark调优都做过哪些方面?

47.spark任务为什么会被yarn kill掉?

48.Spark on Yarn作业执行流程？yarn-client和yarn-cluster有什么区别?

Spark on Yarn作业执行流程?

1.Spark Yarn Client 向 Yarn 中提交应用程序。

2.ResourceManager 收到请求后，在集群中选择一个 NodeManager，并为该应用程序分配一个 Container，在这个 Container 中启动应用程序的 ApplicationMaster，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ResourceManager 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.ApplicationMaster 申请到资源（也就是Container）后，便与对应的 NodeManager 通信，并在获得的 Container 中启动 CoarseGrainedExecutorBackend，启动后会向ApplicationMaster 中的 SparkContext 注册并申请 Task。

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。

6.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有什么区别?

1.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 2. 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 3. 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

49.Flatmap底层编码实现?

Spark flatMap 源码：

```
/**
   *  Return a new RDD by first applying a function to all elements of th
   *  RDD, and then flattening the results.
   */
  def flatMap[U: ClassTag](f: T => TraversableOnce[U]): RDD[U] = withScop
    val cleanF = sc.clean(f)
    new MapPartitionsRDD[U, T](this, (context, pid, iter) => iter.flatMap
  }
```

Scala flatMap 源码：

```
/** Creates a new iterator by applying a function to all values produced
   *  and concatenating the results.
   *
   *  @param f the function to apply on each element.
   *  @return  the iterator resulting from applying the given iterator-v
   *           `f` to each value produced by this iterator and concatena
   *  @note    Reuse: $consumesAndProducesIterator
   */
  def flatMap[B](f: A => GenTraversableOnce[B]): Iterator[B] = new Abstr
    private var cur: Iterator[B] = empty
    private def nextCur() { cur = f(self.next()).toIterator }
    def hasNext: Boolean = {
      // Equivalent to cur.hasNext || self.hasNext && { nextCur(); hasNe
      // but slightly shorter bytecode (better JVM inlining!)
```

```
      while (!cur.hasNext) {
        if (!self.hasNext) return false
        nextCur()
      }
      true
    }
    def next(): B =<span style="color:#ffffff"> <span style="background-
  }
```

flatMap其实就是将RDD里的每一个元素执行自定义函数f，这时这个元素的结果转换成iterator，最后将这些再拼接成一个
新的RDD，也可以理解成原本的每个元素由横向执行函数f后再变为纵向。画红部分一直在回调，当RDD内没有元素为止。

四.Kafka

1.Kafka名词解释和工作方式

1. Producer： 消息生产者，就是向kafka broker发消息的客户端。
2. Consumer： 消息消费者，向kafka broker取消息的客户端
3. Topic： 咋们可以理解为一个队列。
4. Consumer Group （CG）： 这是kafka用来实现一个topic消息的广播（发给所有的consumer）和单播（发给任意一个consumer）的手段。一个topic可以有多个CG。topic的消息会复制（不是真的复制，是概念上的）到所有的CG，但每个partion只会把消息发给该CG中的一个consumer。如果需要实现广播，只要每个consumer有一个独立的CG就可以了。要实现单播只要所有的consumer在同一个CG。用CG还可以将consumer进行自由的分组而不需要多次发送消息到不同的topic。
5. Broker： 一台kafka服务器就是一个broker。一个集群由多个broker组成。一个broker可以容纳多个topic。
6. Partition：为了实现扩展性，一个非常大的topic可以分布到多个broker（即服务器）上，一个topic可以分为多个partition，每个partition是一个有序的队列。partition中的每条消息都会被分配一个有序的id（offset）。kafka只保证按一个partition中的顺序将消息发给consumer，不保证一个topic的整体（多个partition间）的顺序。
7. Offset：kafka的存储文件都是按照offset.kafka来命名，用offset做名字的好处是方便查找。例如你想找位于2049的位置，只要找到2048.kafka的文件即可。当然the first offset就是00000000000.kafka

2.Consumer与topic关系

本质上kafka只支持Topic；

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。

3.kafka中生产数据的时候，如何保证写入的容错性?

设置发送数据是否需要服务端的反馈,有三个值0,1,-1

0: producer不会等待broker发送ack

1: 当leader接收到消息之后发送ack

-1: 当所有的follower都同步消息成功后发送ack

request.required.acks=0

4.如何保证kafka消费者消费数据是全局有序的

伪命题

每个分区内，每条消息都有一个offset，故只能保证分区内有序。

如果要全局有序的，必须保证生产有序，存储有序，消费有序。

由于生产可以做集群，存储可以分片，消费可以设置为一个consumerGroup，要保证全局有序，就需要保证每个环节都有序。

只有一个可能，就是一个生产者，一个partition，一个消费者。这种场景和大数据应用场景相悖。

5.有两个数据源，一个记录的是广告投放给用户的日志，一个记录用户访问日志，另外还有一个固定的用户基础表记录用户基本信息（比如学历，年龄等等）。现在要分析广告投放对与哪类用户更有效，请采用熟悉的技术描述解决思路。另外如果两个数据源都是实时数据源（比如来自kafka），他们数据在时间上相差5分钟，需要哪些调整来解决实时分析问题?

6.Kafka和SparkStreaing如何集成?

7.列举Kafka的优点，简述Kafka为什么可以做到每秒数十万甚至上百万消息的高效分发?

8.为什么离线分析要用kafka?

Kafka的作用是解耦，如果直接从日志服务器上采集的话，实时离线都要采集，等于要采集两份数据，而使用了kafka的话，只需要从日志服务器上采集一份数据，然后在kafka中使用不同的两个组读取就行了

9.Kafka怎么进行监控?

Kafka Manager

10.Kafka与传统的消息队列服务有很么不同

11. Kafka api low-level与high-level有什么区别，使用low-level需要处理哪些细节

12.Kafka的ISR副本同步队列

ISR（In-Sync Replicas），副本同步队列。ISR中包括Leader和Follower。如果Leader进程挂掉，会在ISR队列中选择一个服务作为新的Leader。有replica.lag.max.messages（延迟条数）和replica.lag.time.max.ms（延迟时间）两个参数决定一台服务是否可以加入ISR副本队列，在0.10版本移除了replica.lag.max.messages参数，防止服务频繁的进去队列。

任意一个维度超过阈值都会把Follower剔除出ISR，存入OSR（Outof-Sync Replicas）列表，新加入的Follower也会先存放在OSR中。

13.Kafka消息数据积压，Kafka消费能力不足怎么处理?

1）如果是Kafka消费能力不足，则可以考虑增加Topic的分区数，并且同时提升消费组的消费者数量，消费者数=分区数。（两者缺一不可）

2）如果是下游的数据处理不及时：提高每批次拉取的数量。批次拉取数据过少（拉取数据/处理时间<生产速度），使处理的数据小于生产的数据，也会造成数据积压。

14.Kafka中的ISR、AR又代表什么?

ISR：in-sync replica set (ISR)，与leader保持同步的follower集合

AR：分区的所有副本

15.Kafka中的HW、LEO等分别代表什么?

LEO：每个副本的最后条消息的offset

HW：一个分区中所有副本最小的offset

16.哪些情景会造成消息漏消费?

先提交offset，后消费，有可能造成数据的重复

17.当你使用kafka-topics.sh创建了一个topic之后，Kafka背后会执行什么逻辑?

1）会在zookeeper中的/brokers/topics节点下创建一个新的topic节点，如：/brokers/topics/first

2）触发Controller的监听程序

3）kafka Controller 负责topic的创建工作，并更新metadata cache

18.topic的分区数可不可以增加？如果可以怎么增加？如果不可以，那又是为什么?

可以增加

bin/kafka-topics.sh --zookeeper localhost:2181/kafka --alter --topic topic-config --partitions 3

19.topic的分区数可不可以减少？如果可以怎么减少？如果不可以，那又是为什么?

不可以减少，被删除的分区数据难以处理。

20.Kafka有内部的topic吗？如果有是什么？有什么所用?

__consumer_offsets,保存消费者offset

21.聊一聊Kafka Controller的作用?

负责管理集群broker的上下线，所有topic的分区副本分配和leader选举等工作。

22.失效副本是指什么？有那些应对措施?

不能及时与leader同步，暂时踢出ISR，等其追上leader之后再重新加入

五.Hbase

1.Hbase调优

- 高可用

> 在HBase中Hmaster负责监控RegionServer的生命周期，均衡RegionServer的负载，如果Hmaster挂掉了，那么整个HBase集群将陷入不健康的状态，并且此时的工作状态并不会维持太久。所以HBase支持对Hmaster的高可用配置。

- 预分区

每一个region维护着startRow与endRowKey，如果加入的数据符合某个region维护的rowKey范围，则该数据交给这个region 维护。那么依照这个原则，我们可以将数据所要投放的分区提前大致的规划好，以提高HBase性能。

- 优化RowKey设计

一条数据的唯一标识就是rowkey，那么这条数据存储于哪个分区，取决于rowkey处于哪个一个预分区的区间内，设计rowkey 的主要目的 ，就是让数据均匀的分布于所有的region中，在一定程度上防止数据倾斜

- 内存优化

> HBase操作过程中需要大量的内存开销，毕竟Table是可以缓存在内存中的，一般会分配整个可用内存的70%给HBase的Java堆。但是不建议分配非常大的堆内存，因为GC过程持续太久会导致RegionServer处于长期不可用状态，一般16~48G内存就可以了，如果因为框架占用内存过高导致系统内存不足，框架一样会被系统服务拖死。

2.**hbase的rowkey怎么创建好？列族怎么创建比较好?**

hbase存储时，数据按照Row key的字典序(byte order)排序存储。设计key时，要充分排序存储这个特性，将经常一起读取的行存储放到一起。(位置相关性)

一个列族在数据底层是一个文件，所以将经常一起查询的列放到一个列族中，列族尽量少，减少文件的寻址时间。

设计原则

1）rowkey 长度原则

2）rowkey 散列原则

3）rowkey 唯一原则

如何设计

1）生成随机数、hash、散列值

2）字符串反转

3) 字符串拼接

3.hbase过滤器实现用途

增强hbase查询数据的功能

减少服务端返回给客户端的数据量

4.HBase宕机如何处理

答：宕机分为HMaster宕机和HRegisoner宕机，如果是HRegisoner宕机，HMaster会将其所管理的region重新分布到其他活动的RegionServer上，由于数据和日志都持久在HDFS中，该操作不会导致数据丢失。所以数据的一致性和安全性是有保障的。

如果是HMaster宕机，HMaster没有单点问题，HBase中可以启动多个HMaster，通过Zookeeper的Master Election机制保证总有一个Master运行。即ZooKeeper会保证总会有一个HMaster在对外提供服务。

5.**hive跟hbase的区别是?**

共同点：

1.hbase与hive都是架构在hadoop之上的。都是用hadoop作为底层存储

区别：

2.Hive是建立在Hadoop之上为了减少MapReduce jobs编写工作的批处理系统，HBase是为了支持弥补Hadoop对实时操作的缺陷的项目 。

3.想象你在操作RMDB数据库，如果是全表扫描，就用Hive+Hadoop,如果是索引访问，就用HBase+Hadoop 。

4.Hive query就是MapReduce jobs可以从5分钟到数小时不止，HBase是非常高效的，肯定比Hive高效的多。

5.Hive本身不存储和计算数据，它完全依赖于HDFS和MapReduce，Hive中的表纯逻辑。

6.hive借用hadoop的MapReduce来完成一些hive中的命令的执行

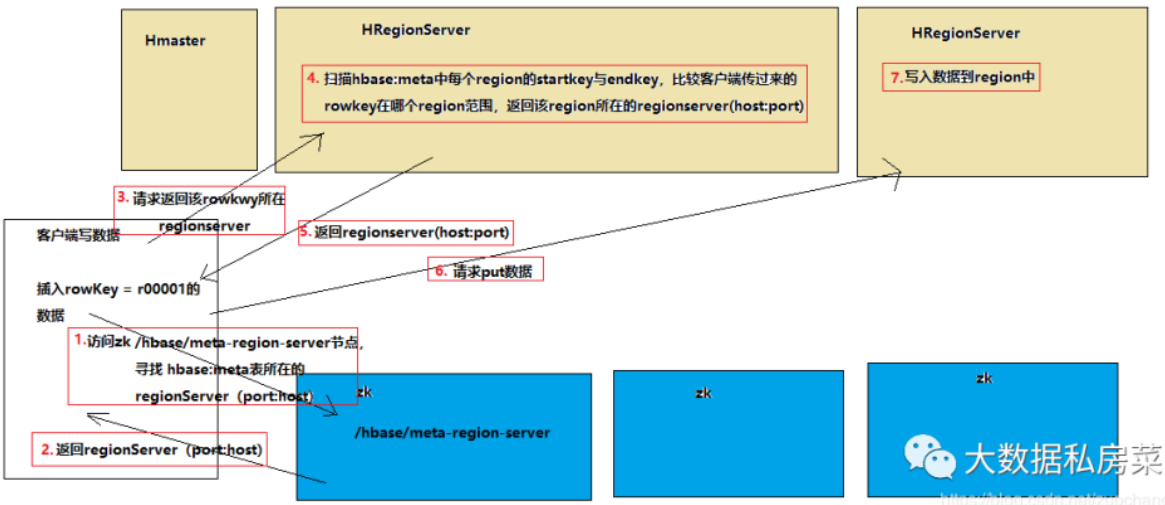
7.hbase是物理表，不是逻辑表，提供一个超大的内存hash表，搜索引擎通过它来存储索引，方便查询操作。

8.hbase是列存储。

9.hdfs作为底层存储，hdfs是存放文件的系统，而Hbase负责组织文件。

10.hive需要用到hdfs存储文件，需要用到MapReduce计算框架。

6.hbase写流程



1/ 客户端要连接zookeeper，从zk的/hbase节点找到hbase:meta表所在的regionserver（host:port）；

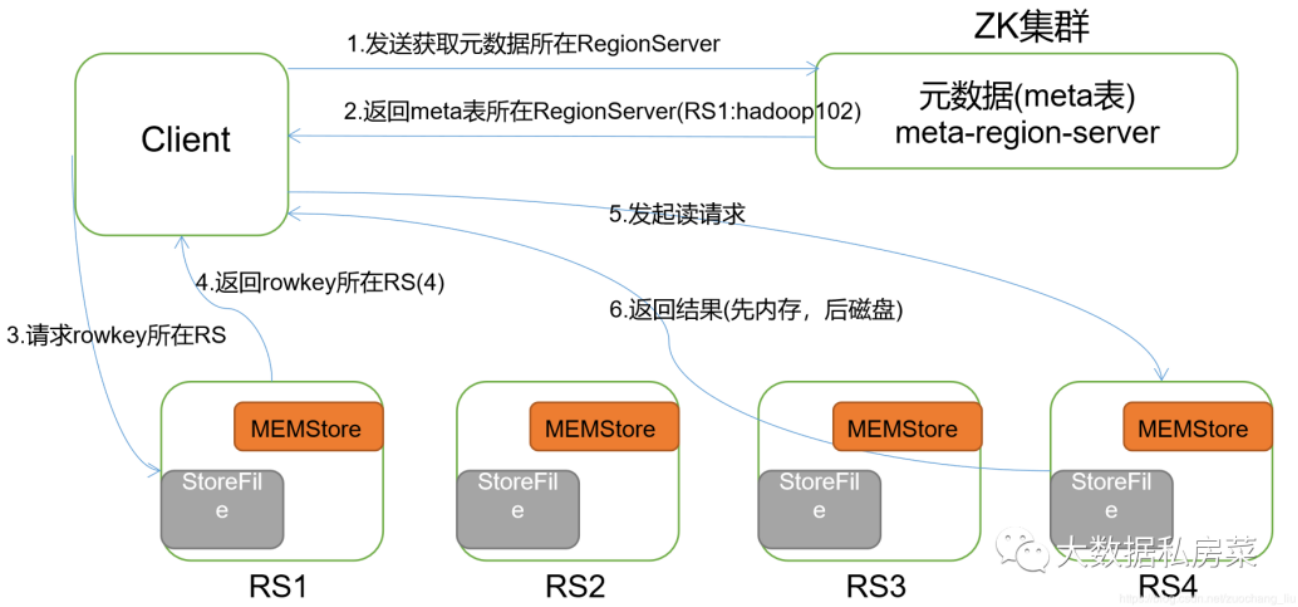
2/ regionserver扫描hbase:meta中的每个region的起始行健，对比r000001这条数据在那个region的范围内；

3/ 从对应的 info:server key中存储了region是有哪个regionserver(host:port)在负责的；

4/ 客户端直接请求对应的regionserver；

5/ regionserver接收到客户端发来的请求之后，就会将数据写入到region中

7.hbase读流程



1/ 首先Client连接zookeeper, 找到hbase:meta表所在的regionserver;

2/ 请求对应的regionserver，扫描hbase:meta表，根据namespace、表名和rowkey在meta表中找到r00001所在的region是由那个regionserver负责的；

3/找到这个region对应的regionserver

4/ regionserver收到了请求之后，扫描对应的region返回数据到Client

(先从MemStore找数据，如果没有，再到BlockCache里面读；BlockCache还没有，再到StoreFile上读(为了读取的效率)；

如果是从StoreFile里面读取的数据，不是直接返回给客户端，而是先写入BlockCache，再返回给客户端。)

8.hbase数据flush过程

1）当MemStore数据达到阈值（默认是128M，老版本是64M），将数据刷到硬盘，将内存中的数据删除，同时删除HLog中的历史数据；

2）并将数据存储到HDFS中；

3）在HLog中做标记点。

9.数据合并过程

1. 当数据块达到4块，hmaster将数据块加载到本地，进行合并
2. 当合并的数据超过256M，进行拆分，将拆分后的region分配给不同的hregionserver管理

3. 当hregionser宕机后，将hregionserver上的hlog拆分，然后分配给不同的hregionserver加载，修改.META.
4. 注意：hlog会同步到hdfs

10.Hmaster和Hgionserver职责

Hmaster

1、管理用户对Table的增、删、改、查操作；
2、记录region在哪台Hregion server上
3、在Region Split后，负责新Region的分配；
4、新机器加入时，管理HRegion Server的负载均衡，调整Region分布
5、在HRegion Server宕机后，负责失效HRegion Server 上的Regions迁移。

Hgionserver

HRegion Server主要负责响应用户I/O请求，向HDFS文件系统中读写数据，是HBASE中最核心的模块。

HRegion Server管理了很多table的分区，也就是region。

11.HBase列族和region的关系？

HBase有多个RegionServer，每个RegionServer里有多个Region，一个Region中存放着若干行的行键以及所对应的数据，一个列族是一个文件夹，如果经常要搜索整个一条数据，列族越少越好，如果只有一部分的数据需要经常被搜索，那么将经常搜索的建立一个列族，其他不常搜索的建立列族检索较快。

12.请简述Hbase的物理模型是什么

13.请问如果使用Hbase做即席查询，如何设计二级索引

14.如何避免读、写HBaes时访问热点问题？

（1）加盐

这里所说的加盐不是密码学中的加盐，而是在rowkey的前面增加随机数，具体就是给rowkey分配一个随机前缀以使得它和之前的rowkey的开头不同。给多少个前缀？这个数量应该和我们想要分散数据到不同的region的数量一致（类似hive里面的分桶）。

（自己理解：即region数量是一个范围，我们给rowkey分配一个随机数，前缀（随机数）的范围是region的数量）

加盐之后的rowkey就会根据随机生成的前缀分散到各个region上，以避免热点。

（2）哈希

哈希会使同一行永远用一个前缀加盐。哈希也可以使负载分散到整个集群，但是读却是可以预测的。使用确定的哈希可以让客户端重构完整的rowkey，可以使用get操作准确获取某一个行数据。

（3）反转

第三种防止热点的方法是反转固定长度或者数字格式的rowkey。这样可以使得rowkey中经常改变的部分（最没有意义的部分）放在前面。这样可以有效的随机rowkey，但是牺牲了rowkey的有序性。反转rowkey的例子：以手机号为rowkey，可以将手机号反转后的字符串作为rowkey，从而避免诸如139、158之类的固定号码开头导 致的热点问题。

（4）时间戳反转

一个常见的数据处理问题是快速获取数据的最近版本，使用反转的时间戳作为rowkey的一部分对这个问题十分有用，可以用Long.Max_Value – timestamp追加到key的末尾，例如[key][reverse_timestamp] ,[key] 的最新值可以通过scan [key]获得[key]的第一条记录，因为HBase中rowkey是有序的，第一条记录是最后录入的数据。

（5）尽量减少行和列的大小

在HBase中，value永远和它的key一起传输的。当具体的值在系统间传输时，它的rowkey，列名，时间戳也会一起传输。如果你的rowkey和列名很大，HBase storefiles中的索引（有助于随机访问）会占据HBase分配的大量内存，因为具体的值和它的key很大。可以增加block大小使得storefiles索引再更大的时间间隔增加，或者修改表的模式以减小rowkey和列名的大小。压缩也有助于更大的索引。

（6）其他办法

列族名的长度尽可能小，最好是只有一个字符。冗长的属性名虽然可读性好，但是更短的属性名存储在HBase中会更好。也可以在建表时预估数据规模，预留region数量，例如create ‘myspace:mytable’, SPLITS => [01,02,03,,...99]

15.布隆过滤器在HBASE中的应用

16.Hbase是用来干嘛的?什么样的数据会放到hbase

六.数仓

1.维表和宽表的考查（主要考察维表的使用及维度退化手法）

维表数据一般根据ods层数据加工生成，在设计宽表的时候，可以适当的用一些维度退化手法，将维度退化到事实表中，减少事实表和维表的关联

2.数仓表命名规范

每个公司都会有点差别

ODS

ods.库名_表名_df/di/da/dz

CDM(dwd/dws)

dwd.主题_内容_df

3.拉链表的使用场景

1.数据量比较大

2.表中的部分字段会被更新

3.需要查看某一个时间点或者时间段的历史快照信息

查看某一个订单在历史某一个时间点的状态

某一个用户在过去某一段时间，下单次数

4.更新的比例和频率不是很大

如果表中信息变化不是很大，每天都保留一份全量，那么每次全量中会保存很多不变的信息，对存储是极大的浪费

4.一亿条数据查的很慢,怎么查快一点

5.有什么维表

时间维表，用户维表，医院维表等

6.数据源都有哪些

业务库数据源:mysql,oracle,mongo

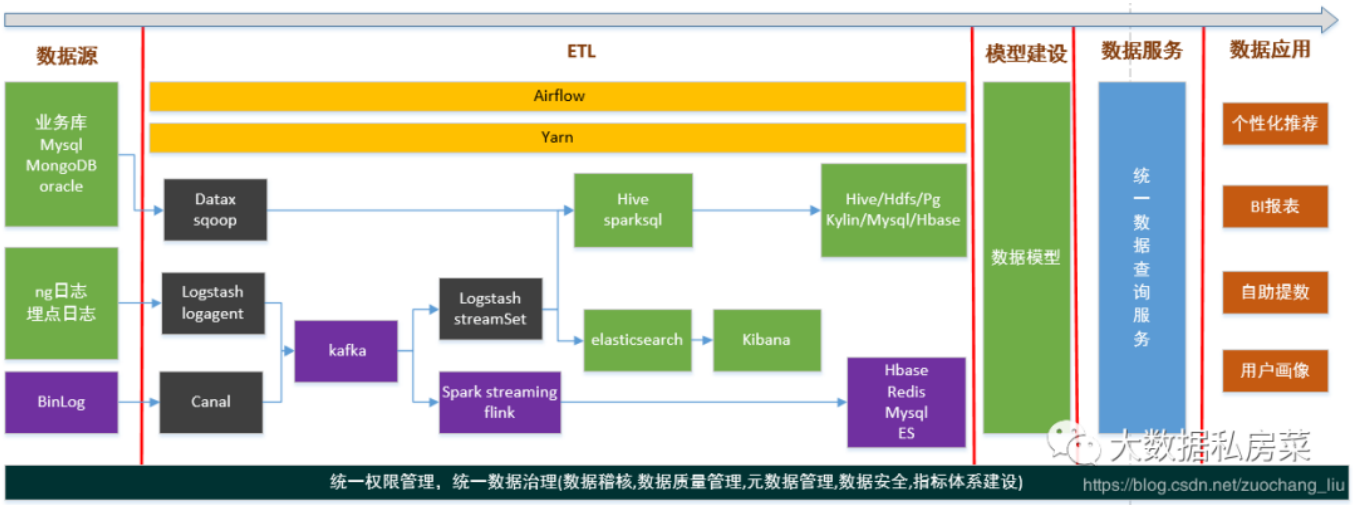
日志数据：ng日志，埋点日志

爬虫数据

7.你们最大的表是什么表,数据量多少

ng日志表，三端(app,web,h5)中app端日志量最大，清洗入库后的数据一天大概xxxxW

8.数仓架构体系



9.数据平台是怎样的，用到了阿里的那一套吗?

没用到阿里那一套，数据平台为自研产品

10.你了解的调度系统有那些？，你们公司用的是哪种调度系统

airflow，azkaban，ooize，我们公司使用的是airflow

11.你们公司数仓底层是怎么抽数据的?

业务数据用的是datax

日志数据用的是logstash

12.为什么datax抽数据要比sqoop 快?

13.埋点数据你们是怎样接入的

logstash-->kafka-->logstash-->hdfs

14.如果你们业务库的表有更新，你们数仓怎么处理的?

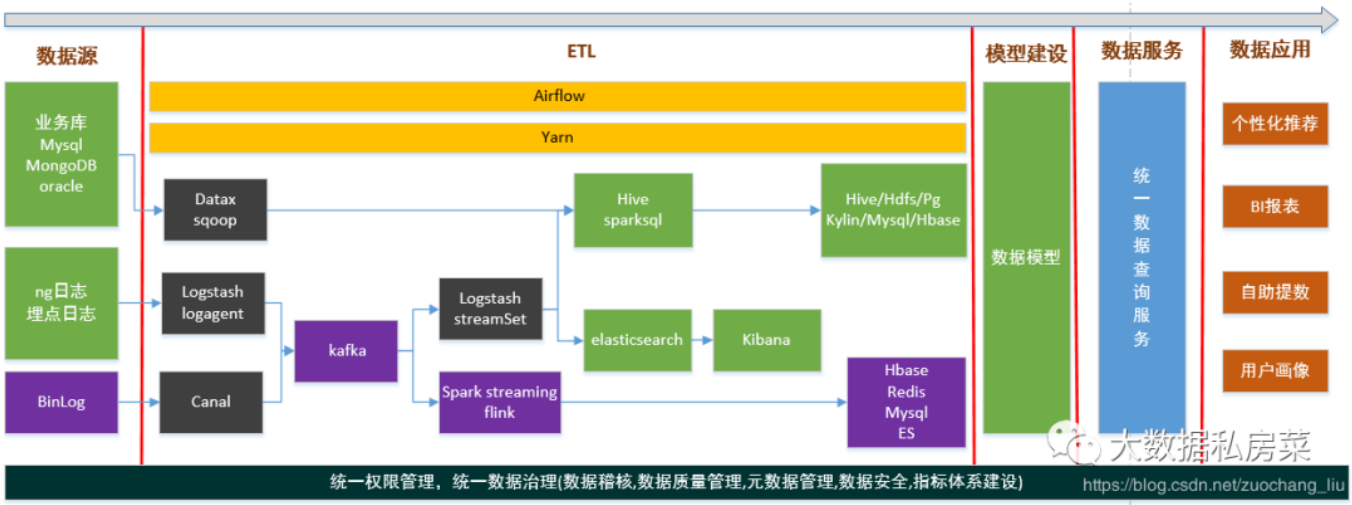
根据表数据量及表特性，选择用全量表，增量表，追加表和拉链表处理

15.能独立搭建数仓吗

可以

16.搭建过CDH 集群吗

17.说一下你们公司的大数据平台架构？你有参与吗?



18.介绍一下你自己的项目和所用的技术

19.对目前的流和批处理的认识？就是谈谈自己的感受

20.你了解那些OLAP 引擎，MPP 知道一些吗？clickHouse 了解一些吗？你自己做过测试性能吗？

21.Kylin 有了解吗？介绍一下原理

22.datax 源码有改造过吗

改造过

23.你们数仓的APP 层是怎么对外提供服务的？

1.直接存入mysql业务库，业务方直接读取

2.数据存入mysql，以接口的形式提供数据

3.数据存入kylin，需求方通过jdbc读取数据

24.数据接入进来，你们是怎样规划的，有考虑数据的膨胀问题吗

25.简述拉链表，流水表以及快照表的含义和特点

拉链表：

（1）记录一个事物从开始，一直到当前状态的所有变化的信息；

（2）拉链表每次上报的都是历史记录的最终状态，是记录在当前时刻的历史总量；

（3）当前记录存的是当前时间之前的所有历史记录的最后变化量（总量）；

（4）封链时间可以是2999，3000，9999等等比较大的年份；拉链表到期数据要报0；

流水表：对于表的每一个修改都会记录，可以用于反映实际记录的变更

区别于拉链表：

拉链表通常是对账户信息的历史变动进行处理保留的结果，流水表是每天的交易形成的历史；

流水表用于统计业务相关情况，拉链表用于统计账户及客户的情况

快照表：

按天分区，每一天的数据都是截止到那一天mysql的全量数据

26.全量表(df),增量表(di),追加表(da)，拉链表(dz)的区别及使用场景

27.你们公司的数仓分层，每一层是怎么处理数据的

28.什么是事实表，什么是维表

29.星型模型和雪花模型

30.数据建模一般有哪几种方式，你们公司是用哪种方式进行数据建模的

ER模型

维度模型

Data Vault模型

Anchor模型

我司用的是维度建模

31.有没有实际工作中碰到的sql调优的例子，举例说明

32.你觉得数据仓库应该如何搭建，数据规范和标准如何落地

33.如何保证你们公司的数据质量

34.对元数据的理解，元数据管理的意义及应用场景有哪些

35.如何判别模型的好坏，模型设计的原则有哪些

基本原则：

- 高内聚和低耦合

一个逻辑或者物理模型由哪些记录和字段组成，应该遵循最基本的软件设计方法论的高内聚和低耦合原则，主要从数据业务特性和访问特性两个角度来考虑：将业务相近或者相关，粒度相同的数据设计为一个逻辑或者物理模型，将高概率同时访问的数据放一起，将低概率同时访问的数据分开存储

- 核心模型与扩展模型分离

建立核心模型与扩展模型体系，核心模型包括的字段支持常用的核心业务，扩展模型包括的字段支持个性化或少量应用的需要，不能让扩展模型的字段过度侵入核心模型，以免破坏核心模型架构简洁性与可维护性

- 公共处理逻辑下沉及单一

越是底层共用的处理逻辑越应该放在数据调度依赖的底层进行封装与实现，不要让共用的处理逻辑暴露给应用层实现，不要让公共逻辑多处同时存在

- 成本与性能平衡

适当的数据冗余可换取查询和刷新性能，不宜过度数据冗余和数据复制

- 数据可回滚

处理逻辑不变，在不同时间多次运行数据结果确定不变

- 一致性

具有相同含义的字段在不同的表中的命名必须相同，必须使用规范定义中的名称

- 命名清晰,可理解

表命名需清晰，一致，表名需易于消费者理解和使用

36.对于数据中台的理解，和数据仓库，数据湖的区别

37.对于数据仓库的理解，数据仓库主要为解决什么问题

38.数据仓库模型的理解，数据仓库分层设计的好处是什么

清晰数据结构：每一个数据分层都有它的作用域和职责，在使用表的时候能更方便地定位和理解

减少重复开发：规范数据分层，开发一些通用的中间层数据，能减少极大的重复计算

统一数据口径：通过数据分层提供统一的数据出口，统一对外输出的数据口径

复杂问题简单化：将一个复杂的任务分解成多个步骤来完成，每一层解决特定的问题

39.数仓主题划分的标准和依据

40.缓慢变化维如何处理，几种方式

七.Flink

1.**Flink实时计算时落磁盘吗**

不落，是内存计算

2.**日活DAU的统计需要注意什么**

3.**Flink调优**

4.Flink的容错是怎么做的

定期checkpoint存储oprator state及keyedstate到stateBackend

5.Parquet格式的好处？什么时候读的快什么时候读的慢

6.flink中checkPoint为什么状态有保存在内存中这样的机制？为什么要开启checkPoint?

开启checkpoint可以容错，程序自动重启的时候可以从checkpoint中恢复数据

7.flink保证Exactly_Once的原理?

1.开启checkpoint

2.source支持数据重发

3.sink支持事务，可以分2次提交，如kafka；或者sink支持幂等，可以覆盖之前写入的数据，如redis

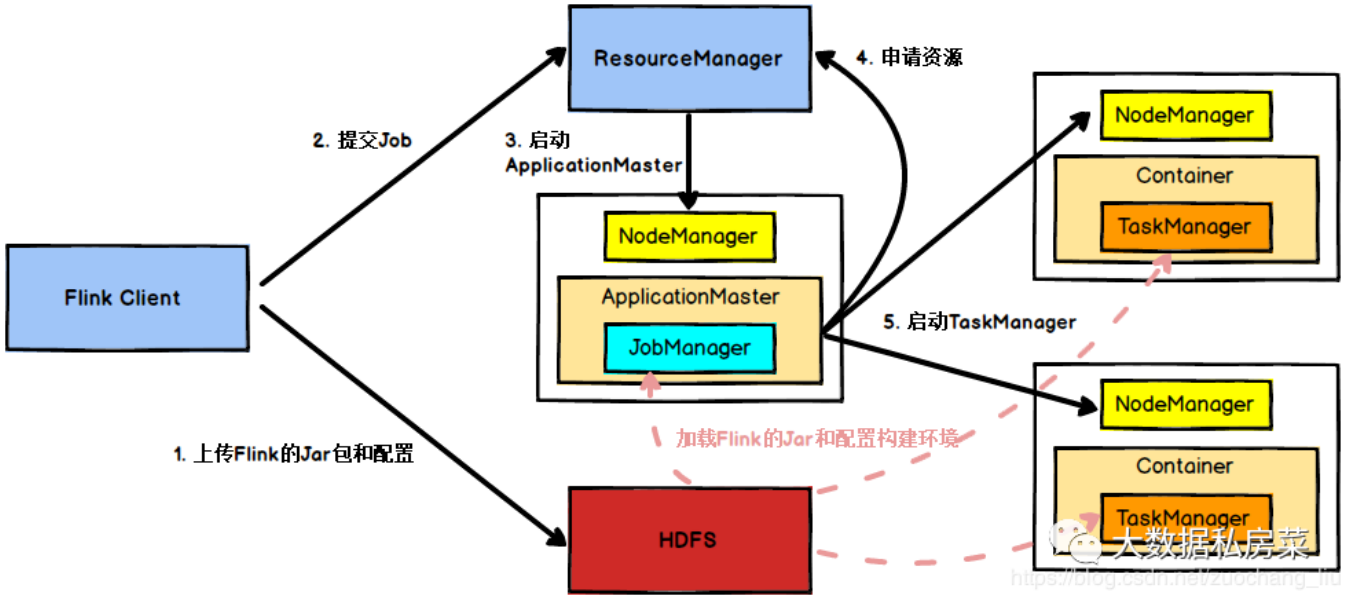
满足以上三点，可以保证Exactly_Once

8.flink的时间形式和窗口形式有几种？有什么区别，你们用在什么场景下的?

9.flink的背压说下?

10.flink的watermark机制说下，以及怎么解决数据乱序的问题?

11.flink on yarn执行流程



Flink任务提交后，Client向HDFS上传Flink的Jar包和配置，之后向Yarn ResourceManager提交任务，ResourceManager分配Container资源并通知对应的NodeManager启动ApplicationMaster，ApplicationMaster启动后加载Flink的Jar包和配置构建环境，然后启动JobManager，之后ApplicationMaster向ResourceManager申请资源启动TaskManager，ResourceManager分配Container资源后，由ApplicationMaster通知资源所在节点的NodeManager启动TaskManager，NodeManager加载Flink的Jar包和配置构建环境并启动TaskManager，TaskManager启动后向JobManager发送心跳包，并等待JobManager向其分配任务。

12.说一说spark 和flink 的区别

八.Java

1.hashMap底层源码，数据结构

2.写出你用过的设计模式，并举例说明解决的实际问题

3.Java创建线程的几种方式

1. 继承Thread类，重写run方法
2. 实现Runnable接口，实现run方法
3. 通过线程池获取线程
4. 实现Callable接口并实现call方法，创建该类的实例，使用FutureTask类包装Callable对象，使用FutureTask对象作为Thread对象的target创建并启用新线程

4.请简述操作系统的线程和进程的区别

5.Java程序出现OutOfMemoryError:unable to create new native thread 的原因可能有哪些？如何分析和解决?

6.采用java或自己熟悉的任何语言分别实现简单版本的线性表和链表，只需实现add,remove方法即可

7.ArrayList和LinkedList的区别

8.JVM 内存分哪几个区，每个区的作用是什么?

9.**Java中迭代器和集合的区别?**

集合是将所有数据加载到内存，然后通过集合的方法去内存中获取，而迭代器是一个对象，实现了Iterator接口，实现了接口的hasNext和Next方法。

10. HashMap 和 HashTable 区别

1) 线程安全性不同

HashMap 是线程不安全的，HashTable 是线程安全的，其中的方法是 Synchronize 的，在多线程并发的情况下，可以直接使用 HashTabl，但是使用 HashMap 时必须自己增加同步

处理。

2) 是否提供 contains 方法

HashMap 只有 containsValue 和 containsKey 方法；HashTable 有 contains、containsKey

和 containsValue 三个方法，其中 contains 和 containsValue 方法功能相同。

3) key 和 value 是否允许 null 值

Hashtable 中，key 和 value 都不允许出现 null 值。HashMap 中，null 可以作为键，这样的键只有一个；可以有一个或多个键所对应的值为 null。

4) 数组初始化和扩容机制

HashTable 在不指定容量的情况下的默认容量为 11，而 HashMap 为 16，Hashtable 不要求底层数组的容量一定要为 2 的整数次幂，而 HashMap 则要求一定为 2 的整数次幂。

Hashtable 扩容时，将容量变为原来的 2 倍加 1，而 HashMap 扩容时，将容量变为原来的 2 倍。

11.线程池使用注意哪些方面?

线程池分为单线程线程池，固定大小线程池，可缓冲的线程池

12.HashMap和TreeMap的区别？TreeMap排序规则?

TreeMap会自动进行排序，根据key的Compare方法进行排序

13.用java实现单例模式

14.使用递归算法求n的阶乘：n! ,语言不限

15.**HashMap和Hashtable的区别是什么**

16.TreeSet 和 HashSet 区别

HashSet 是采用 hash 表来实现的。其中的元素没有按顺序排列，add()、remove()以及 contains()等方法都是复杂度为 O(1)的方法。

TreeSet 是采用树结构实现（红黑树算法）。元素是按顺序进行排列，但是 add()、remove()以及 contains()等方法都是复杂度为 O(log (n))的方法。它还提供了一些方法来处理

排序的 set，如 first()，last()，headSet()，tailSet()等等。

17.Stringbuffer 和 Stringbuild 区别

1、StringBuffer 与 StringBuilder 中的方法和功能完全是等价的。

2、只是 StringBuffer 中的方法大都采用了 synchronized 关键字进行修饰，因此是线程安全的，而 StringBuilder 没有这个修饰，可以被认为是线程不安全的。

3、在单线程程序下，StringBuilder 效率更快，因为它不需要加锁，不具备多线程安全而 StringBuffer 则每次都需要判断锁，效率相对更低

18.Final、Finally、Finalize

final：修饰符（关键字）有三种用法：修饰类、变量和方法。修饰类时，意味着它不能再派生出新的子类，即不能被继承，因此它和 abstract 是反义词。修饰变量时，该变量
使用中不被改变，必须在声明时给定初值，在引用中只能读取不可修改，即为常量。修饰方法时，也同样只能使用，不能在子类中被重写。
finally：通常放在 try...catch 的后面构造最终执行代码块，这就意味着程序无论正常执行还是发生异常，这里的代码只要 JVM 不关闭都能执行，可以将释放外部资源的代码写在
finally 块中。
finalize：Object 类中定义的方法，Java 中允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在销毁对象时调用的，通过重写 finalize() 方法可以整理系统资源或者执行其他清理工作。

19..==和 Equals 区别

==：如果比较的是基本数据类型，那么比较的是变量的值
如果比较的是引用数据类型，那么比较的是地址值（两个对象是否指向同一块内存）
equals:如果没重写 equals 方法比较的是两个对象的地址值。
如果重写了 equals 方法后我们往往比较的是对象中的属性的内容
equals 方法是从 Object 类中继承的，默认的实现就是使用==

```
public boolean equals(Object obj) {
    return (this == obj);
}
```


20.比较ArrayList，LinkedList的存储特性和读写性能

21.Java 类加载过程

Java类加载需要经历一下几个过程：

- 加载

加载时类加载的第一个过程，在这个阶段，将完成一下三件事情：

1. 通过一个类的全限定名获取该类的二进制流。
2. 将该二进制流中的静态存储结构转化为方法去运行时数据结构。
3. 在内存中生成该类的Class对象，作为该类的数据访问入口。

- 验证

验证的目的是为了确保Class文件的字节流中的信息不回危害到虚拟机.在该阶段主要完成以下四钟验证:

1. 文件格式验证：验证字节流是否符合Class文件的规范，如主次版本号是否在当前虚拟机范围内，常量池中的常量是否有不被支持的类型.
2. 元数据验证:对字节码描述的信息进行语义分析，如这个类是否有父类，是否集成了不被继承的类等。

3. 字节码验证：是整个验证过程中最复杂的一个阶段，通过验证数据流和控制流的分析，确定程序语义是否正确，主要针对方法体的验证。如：方法中的类型转换是否正确，跳转指令是否正确等。
4. 符号引用验证：这个动作在后面的解析过程中发生，主要是为了确保解析动作能正确执行。
5. 准备

准备阶段是为类的静态变量分配内存并将其初始化为默认值，这些内存都将在方法区中进行分配。准备阶段不分配类中的实例变量的内存，实例变量将会在对象实例化时随着对象一起分配在Java堆中。

- 解析

该阶段主要完成符号引用到直接引用的转换动作。解析动作并不一定在初始化动作完成之前，也有可能在初始化之后。

- 初始化

初始化时类加载的最后一步，前面的类加载过程，除了在加载阶段用户应用程序可以通过自定义类加载器参与之外，其余动作完全由虚拟机主导和控制。到了初始化阶段，才真正开始执行类中定义的Java程序代码。

22.java中垃圾收集的方法有哪些?

23.如何判断一个对象是否存活?(或者GC对象的判定方法)

判断一个对象是否存活有两种方法:

1. 引用计数法
2. 可达性算法(引用链法)

九.Elasticsearch

1.为什么要用es？存进es的数据是什么格式的，怎么查询

十.Flume

1.什么是flume

a.Flume是一个分布式、可靠、和高可用的海量日志采集、聚合和传输的系统。

b.Flume可以采集文件，socket数据包等各种形式源数据，又可以将采集到的数据输出到HDFS、hbase、hive、kafka等众多外部存储系统中

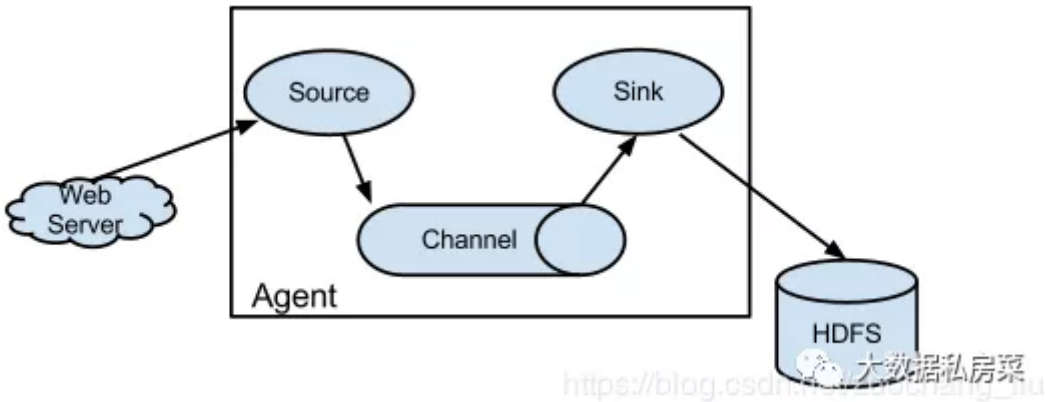
c.一般的采集需求，通过对flume的简单配置即可实现

d.ume针对特殊场景也具备良好的自定义扩展能力，因此，flume可以适用于大部分的日常数据采集场景

2.flume运行机制

1. Flume分布式系统中最**核心的角色是agent**，flume采集系统就是由一个个agent所连接起来形成
2. **每一个agent相当于一个数据传递员，内部有三个组件：**
   a. Source：采集源，用于跟数据源对接，以获取数据
   b. Sink：下沉地，采集数据的传送目的，用于往下一级agent传递数据或者往最终存储系统传递数据
   c. Channel：angent内部的数据传输通道，用于从source将数据传递到sink

d.



3.Flume采集数据到Kafka中丢数据怎么办

4.Flume怎么进行监控?

5.Flume的三层架构，collector、agent、storage

十一.Sqoop

1.**Sqoop底层运行的任务是什么**

只有Map阶段，没有Reduce阶段的任务。

2.sqoop的迁移数据的原理

3.Sqoop参数

```
/opt/module/sqoop/bin/sqoop import \
--connect \
--username \
--password \
--target-dir \
--delete-target-dir \
--num-mappers \
--fields-terminated-by   \
--query   "$2" ' and $CONDITIONS;'
```

4.Sqoop导入导出Null存储一致性问题

Hive中的Null在底层是以“\N”来存储，而MySQL中的Null在底层就是Null，为了保证数据两端的一致性。在导出数据时采用--input-null-string和--input-null-non-string两个参数。导入数据时采用--null-string和--null-non-string。

5.**Sqoop数据导出一致性问题**

1）场景1：如Sqoop在导出到Mysql时，使用4个Map任务，过程中有2个任务失败，那此时MySQL中存储了另外两个Map任务导入的数据，此时老板正好看到了这个报表数据。而开发工程师发现任务失败后，会调试问题并最终将全部数据正确的导入MySQL，那后面老板再次看报表数据，发现本次看到的数据与之前的不一致，这在生产环境是不允许的。

2）场景2：设置map数量为1个（不推荐，面试官想要的答案不只这个）

多个Map任务时，采用–staging-table方式，仍然可以解决数据一致性问题。

十二.Redis

1.缓存穿透、缓存雪崩、缓存击穿

1）缓存穿透是指查询一个一定不存在的数据。由于缓存命不中时会去查询数据库，查不到
数据则不写入缓存，这将导致这个不存在的数据每次请求都要到数据库去查询，造成缓存穿
透。
解决方案：
① 是将空对象也缓存起来，并给它设置一个很短的过期时间，最长不超过 5 分钟
② 采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的 bitmap 中，一个一定不存在的数据会被这个 bitmap 拦截掉，从而避免了对底层存储系统的查询压力
2）如果缓存集中在一段时间内失效，发生大量的缓存穿透，所有的查询都落在数据库上，
就会造成缓存雪崩。
解决方案：
尽量让失效的时间点不分布在同一个时间点
3）缓存击穿，是指一个 key 非常热点，在不停的扛着大并发，当这个 key 在失效的瞬间，
持续的大并发就穿破缓存，直接请求数据库，就像在一个屏障上凿开了一个洞。
可以设置 key 永不过期

2.数据类型

string	字符串
list	可以重复的集合
set	不可以重复的集合
hash	类似于 Map<String,String>
zset(sorted set）	带分数的 set

3.持久化

1）RDB 持久化：
① 在指定的时间间隔内持久化
② 服务 shutdown 会自动持久化
③ 输入 bgsave 也会持久化
2）AOF：以日志形式记录每个更新操作
Redis 重新启动时读取这个文件，重新执行新建、修改数据的命令恢复数据。
保存策略：
推荐（并且也是默认）的措施为每秒持久化一次，这种策略可以兼顾速度和安全性。
缺点：

1 比起 RDB 占用更多的磁盘空间
2 恢复备份速度要慢
3 每次读写都同步的话，有一定的性能压力
4 存在个别 Bug，造成恢复不能

选择策略：

官方推荐：

string
字符串
list
可以重复的集合
set
不可以重复的集合
hash
类似于 Map<String,String>
zset(sorted set)
带分数的 set

如果对数据不敏感，可以选单独用 RDB；不建议单独用 AOF，因为可能出现 Bug;如果只是做纯内存缓存，可以都不用

4.悲观锁和乐观锁

悲观锁：执行操作前假设当前的操作肯定（或有很大几率）会被打断（悲观）。基于这个假设，我们在做操作前就会把相关资源锁定，不允许自己执行期间有其他操作干扰。

乐观锁：执行操作前假设当前操作不会被打断（乐观）。基于这个假设，我们在做操作前不会锁定资源，万一发生了其他操作的干扰，那么本次操作将被放弃。Redis 使用的就是乐观锁。

5.redis 是单线程的，为什么那么快

1)完全基于内存，绝大部分请求是纯粹的内存操作，非常快速。
2)数据结构简单，对数据操作也简单，Redis 中的数据结构是专门进行设计的
3)采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或者多线程导致的切换而消耗 CPU，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗
4)使用多路 I/O 复用模型，非阻塞 IO
5)使用底层模型不同，它们之间底层实现方式以及与客户端之间通信的应用协议不一样，Redis 直接自己构建了 VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求

6.redis的热键问题？怎么解决？

十三.Mysql

1.请写出mysql登录命令，用户名user，密码123456，地址192.168.1.130

```
mysql -h 192.168.1.130 -uuser -p123456 -P3306 -Dwemeta_test
```

2.为什么MySQL的索引要使用B+树而不是其它树形结构?比如B树?

B树

B树不管叶子节点还是非叶子节点，都会保存数据，这样导致在非叶子节点中能保存的指针数量变少（有些资料也称为扇出）

指针少的情况下要保存大量数据，只能增加树的高度，导致IO操作变多，查询性能变低；

B+树

1.单一节点存储更多的元素，使得查询的IO次数更少。

2.所有查询都要查找到叶子节点，查询性能稳定。

3.所有叶子节点形成有序链表，便于范围查询,远远高于B-树

十四.数据结构与算法

1.二分查找

```
package com.wedoctor.search;

public class Binarysearch {
   public static int bsearchWithoutRecursion(int arr[], int key) {
        int low = 0;
        int high = arr.length - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] > key)
                high = mid - 1;
            else if (arr[mid] < key)
                low = mid + 1;
            else
                return mid;
        }
        return -1;
    }

    public static void main(String[] args) {
        int arr[] = {1,3,5,6,8,9,11,14,23};
        int num = bsearchWithoutRecursion(arr, 9);
        System.out.println(num);
    }
}
```

2.快排

3.归并排序

4.冒泡排序

```java
package com.wedoctor.sort;

import java.util.Arrays;

public class BubbleSort {
    public static void main(String[] args) {
        int[] arr = new int[] { 2, 8, 7, 9, 4, 1, 5, 0 };
        bubbleSort(arr);
    }

    public static void bubbleSort(int[] arr) {
        //控制多少轮
        for (int i = 1; i < arr.length; i++) {
            //控制每一轮的次数
            for (int j = 0; j <= arr.length -1 - i; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp;
                    temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
        System.out.println(Arrays.toString(arr));

    }
}
```

5.字符串反转

```java
package com.wedoctor.str;

public class StrReverse {
    public  static  String getNewStr(String str){
        StringBuffer sb = new StringBuffer(str);
        String newStr = sb.reverse().toString();
```

```
        return newStr;
    }

    public static void main(String[] args) {
        System.out.println(getNewStr("thjymhr"));
    }
}
```

6.Btree简单讲一下

B树(B-树)是一种适合外查找的搜索树，是一种平衡的多叉树

B树的每个结点包含着结点的值和结点所处的位置

7.动态规划 最大连续子序列和

```
package com.wedoctor;

import java.util.Arrays;

public class MaxSum {
    public static int findMax(int arr[]){
         if (arr.length == 1){
             return arr[0];
         }
        int mid = (arr.length) / 2;
        int[] leftArr = Arrays.copyOfRange(arr, 0, mid);
        int[] rightArr = Arrays.copyOfRange(arr, mid, arr.length);

        int lenLeft = findMax(leftArr);
        int lenRight = findMax(rightArr);
        int lenMid = maxInMid(leftArr, rightArr);

        int max = Math.max(Math.max(lenLeft,lenRight),lenMid);
        return max;
    }

    public static int maxInMid(int left[],int right[]){
        int maxLeft = 0;
        int maxRight = 0;
        int tmpLeft = 0;
```

```
        int tmpRight = 0;
        for (int i = 0;i< left.length;i++){
            tmpLeft = tmpLeft + left[left.length - 1 - i];
            maxLeft = Math.max(tmpLeft,maxLeft);
        }

        for (int i = 0;i< right.length;i++){
            tmpRight = tmpRight + right[i];
            maxRight = Math.max(tmpRight,maxRight);
        }
        return  maxRight + maxLeft;
    }

    public static void main(String[] args) {
        int arr[] = {3,-1,10};
        System.out.println(findMax(arr));
    }
}
```

8.二叉树概念，特点及代码实现

二叉树是n(n>=0)个结点的有限集合，该集合或者为空集（称为空二叉树），或者由一个根结点和两棵互不相交的、分别称为根结点的左子树和右子树组成。

特点：

- 每个结点最多有两颗子树，所以二叉树中不存在度大于2的结点。
- 左子树和右子树是有顺序的，次序不能任意颠倒。
- 即使树中某结点只有一棵子树，也要区分它是左子树还是右子树。

实现：

```
package com.wedoctor;

public class BinaryTreeNode {
    int data;
    BinaryTreeNode left;
    BinaryTreeNode right;

    BinaryTreeNode (int x) {
        data= x;
    }
```

```
11
12      public BinaryTreeNode(int data, BinaryTreeNode left, BinaryTreeNode
13          this.data = data;
14          this.left = left;
15          this.right = right;
16      }
17
18      public int getData() {
19          return data;
20      }
21
22      public void setData(int data) {
23          this.data = data;
24      }
25
26      public BinaryTreeNode getLeft() {
27          return left;
28      }
29
30      public void setLeft(BinaryTreeNode left) {
31          this.left = left;
32      }
33
34      public BinaryTreeNode getRight() {
35          return right;
36      }
37
38      public void setRight(BinaryTreeNode right) {
39          this.right = right;
40      }
41  }
```

9.链表

十五.Linux

序号	命令	命令解释
1	top	查看内存
2	df -h	查看磁盘存储情况

3	iotop	查看磁盘IO读写(yum install iotop安装)
4	iotop -o	直接查看比较高的磁盘读写程序
5	netstat -tunlp | grep 端口号	查看端口占用情况
6	uptime	查看报告系统运行时长及平均负载
7	ps aux	查看进程