1.TCP和UDP的区别

答:TCP提供面向连接的、可靠的数据流传输，而UDP提供的是非面向连接的、不可靠的数据流传输。  
TCP传输单位称为TCP报文段，UDP传输单位称为用户数据报。  
TCP注重数据安全性，UDP数据传输快，因为不需要连接等待，少了许多操作，但是其安全性却一般。  
TCP对应的协议和UDP对应的协议  
TCP对应的协议：  
（1） FTP：定义了文件传输协议，使用21端口。  
（2） Telnet：一种用于远程登陆的端口，使用23端口，用户可以以自己的身份远程连接到计算机上，可提供基于DOS模式下的通信服务。  
（3） SMTP：邮件传送协议，用于发送邮件。服务器开放的是25号端口。  
（4） POP3：它是和SMTP对应，POP3用于接收邮件。POP3协议所用的是110端口。  
（5）HTTP：是从Web服务器传输超文本到本地浏览器的传送协议。  
UDP对应的协议：  
（1） DNS：用于域名解析服务，将域名地址转换为IP地址。DNS用的是53号端口。  
（2） SNMP：简单网络管理协议，使用161号端口，是用来管理网络设备的。由于网络设备很多，无连接的服务就体现出其优势。  
（3） TFTP(Trival File Transfer Protocal)，简单文件传输协议，该协议在熟知端口69上使用UDP服务。

1. http和https的区别

HTTP协议传输的数是明文的，因此使用HTTP协议传输隐私信息非常不安全，为据都是未加密的，也就了保证这些隐私数据能加密传输，于是网景公司设计了SSL（Secure Sockets Layer）协议用于对HTTP协议传输的数据进行加密，从而就诞生了HTTPS。简单来说，HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全。

　　HTTPS和HTTP的区别主要如下：

　　1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

　　2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

　　3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

　　4、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

3. spark streaming kafka的两种连接方式和区别

Receiver-based的Kafka读取方式是基于Kafka高阶(high-level) api来实现对Kafka数据的消费。在提交Spark Streaming任务后，Spark集群会划出指定的Receivers来专门、持续不断、异步读取Kafka的数据，读取时间间隔以及每次读取offsets范围可以由参数来配置。读取的数据保存在Receiver中，具体StorageLevel方式由用户指定，诸如MEMORY\_ONLY等。当driver 触发batch任务的时候，Receivers中的数据会转移到剩余的[Executor](https://www.baidu.com/s?wd=Executor&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/stevekangpei/article/details/_blank)s中去执行。在执行完之后，Receivers会相应更新ZooKeeper的offsets。如要确保at least once的读取方式，可以设置spark.streaming.receiver.writeAheadLog.enable为true。

存在的问题是：

配置spark.streaming.receiver.writeAheadLog.enable参数，每次处理之前需要将该batch内的日志备份到checkpoint目录中，这降低了数据处理效率，反过来又加重了Receiver端的压力；另外由于数据备份机制，会受到负载影响，负载一高就会出现延迟的风险，导致应用崩溃。

采用MEMORY\_AND\_DISK\_SER降低对内存的要求。但是在一定程度上影响计算的速度

单Receiver内存。由于receiver也是属于Executor的一部分，那么为了提高吞吐量，提高Receiver的内存。但是在每次batch计算中，参与计算的batch并不会使用到这么多的内存，导致资源严重浪费。

提高并行度，采用多个Receiver来保存Kafka的数据。Receiver读取数据是异步的，并不参与计算。如果开较高的并行度来平衡吞吐量很不划算。

Receiver和计算的Executor的异步的，那么遇到网络等因素原因，导致计算出现延迟，计算队列一直在增加，而Receiver则在一直接收数据，这非常容易导致程序崩溃。

在程序失败恢复时，有可能出现数据部分落地，但是程序失败，未更新offsets的情况，这导致数据重复消费。

Direct 读取方式

Direct方式采用Kafka简单的consumer api方式来读取数据，无需经由ZooKeeper，此种方式不再需要专门Receiver来持续不断读取数据。当batch任务触发时，由Executor读取数据，并参与到其他Executor的数据计算过程中去。driver来决定读取多少offsets，并将offsets交由checkpoints来维护。将触发下次batch任务，再由Executor读取Kafka数据并计算。从此过程我们可以发现Direct方式无需Receiver读取数据，而是需要计算时再读取数据，所以Direct方式的数据消费对内存的要求不高，只需要考虑批量[计算所](https://www.baidu.com/s?wd=%E8%AE%A1%E7%AE%97%E6%89%80&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/stevekangpei/article/details/_blank)需要的内存即可；另外batch任务堆积时，也不会影响数据堆积。其具体读取方式如下图：

存在的问题：

提高成本。Direct需要用户采用checkpoint或者第三方存储来维护offsets，而不像Receiver-based那样，通过ZooKeeper来维护Offsets，此提高了用户的开发成本。

监控可视化。Receiver-based方式指定topic指定consumer的消费情况均能通过ZooKeeper来监控，而Direct则没有这种便利，如果做到监控并可视化，则需要投入人力开发。

4 spark streaming exactly only once怎么实现

使用幂等写入实现 Exactly-once

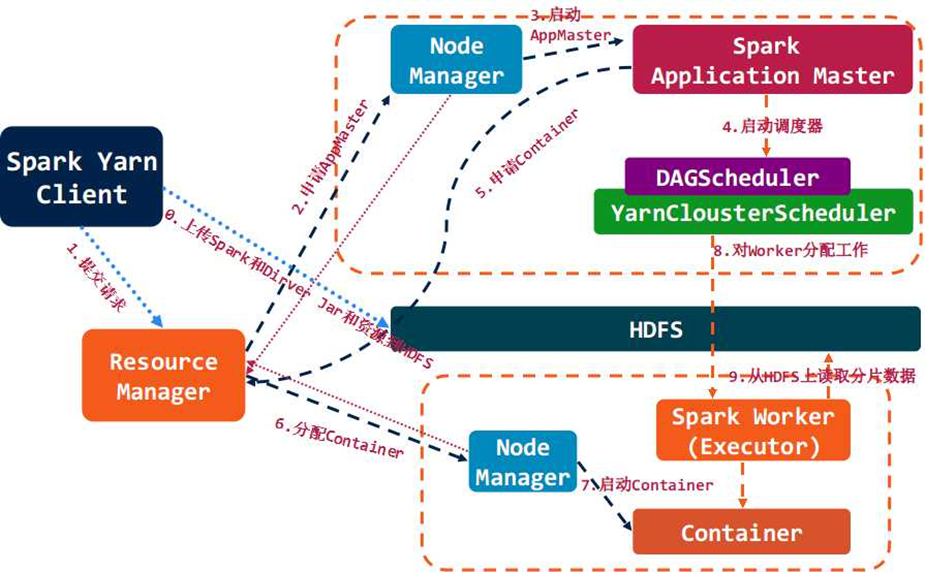
如果多次写入会产生相同的结果数据，我们可以认为这类写入操作是幂等的。saveAsTextFile 就是一种典型的幂等写入。如果消息中包含唯一主键，那么多次写入相同的数据也不会在数据库中产生重复记录。这种方式也就能等价于 Exactly-once 语义了。但需要注意的是，幂等写入只适用于 Map-only 型的计算流程，即没有 Shuffle、Reduce、Repartition 等操作。此外，我们还需对 Kafka DStream 做一些额外设置：

---------------------

使用事务写入实现 Exactly-once

在使用事务型写入时，我们需要生成一个唯一 ID，这个 ID 可以使用当前批次的时间、分区号、或是 Kafka 偏移量来生成。之后，我们需要在一个事务中将处理结果和这个唯一 ID 一同写入数据库。这一原子性的操作将带给我们 Exactly-once 语义，而且该方法可以同时适用于 Map-only 以及包含汇聚操作的计算流程。

5 spark的运行流程



6 Bulk Load的步骤

1、准备数据文件

Bulk Load的第一步，会运行一个Mapreduce作业，其中使用到了HFileOutputFormat输出HBase数据文件：StoreFile。HFileOutputFormat的作用在于使得输出的HFile文件可以适应单个region，使用TotalOrderPartitioner类将map输出结果分区到各个不同的key区间中，每个key区间都对应着HBase表的region。

2、导入HBase表

第二步使用completebulkload工具将第一步的结果文件依次交给负责文件对应region的RegionServer，并将文件move到region在HDFS上的存储目录中，一旦完成，将数据开放给clients。

如果在bulk load准备导入或在准备导入与完成导入的临界点上发现region的边界已经改变，completebulkload工具会自动split数据文件到新的边界上，但是这个过程并不是最佳实践，所以用户在使用时需要最小化准备导入与导入集群间的延时，特别是当其他client在同时使用其他工具向同一张表导入数据。

7 kafka的分区分配策略

<https://blog.csdn.net/qq_39907763/article/details/82697211>

8 HIVE中的UDF UDAF UDTF

1. UDF：用户定义（普通）函数，只对单行数值产生作用；
2. UDAF：User- Defined Aggregation Funcation；用户定义聚合函数，可对多行数据产生作用；等同与SQL中常用的SUM()，AVG()，也是聚合函数；
3. UDTF：User-Defined Table-Generating Functions，用户定义表生成函数，用来解决输入一行输出多行；

9 kafka中数据储存结构

<https://blog.csdn.net/u012501054/article/details/80180370>

10 数据库的三范式

**第一范式（1NF）**

所谓第一范式（1NF）是指在[关系模型](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E6%A8%A1%E5%9E%8B" \t "https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E8%8C%83%E5%BC%8F/_blank)中，对于添加的一个规范要求，所有的域都应该是原子性的，即数据库表的**每一列都是不可分割的原子数据项**，而不能是集合，数组，记录等非原子数据项。即实体中的某个属性有多个值时，必须拆分为不同的属性。在符合第一范式（1NF）表中的每个域值只能是实体的一个属性或一个属性的一部分。简而言之，第一范式就是无重复的域。

**第二范式（2NF）**

在1NF的基础上，非码属性必须完全依赖于候选码（在1NF基础上消除非主属性对主码的部分函数依赖）

第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。第二范式（2NF）要求数据库表中的每个实例或记录必须可以被唯一地区分。简而言之，第二范式就是在第一范式的基础上属性完全依赖于主键。

**第三范式（3NF）**

在2NF基础上，**任何非主[属性](https://baike.baidu.com/item/%E5%B1%9E%E6%80%A7" \t "https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E8%8C%83%E5%BC%8F/_blank)不依赖于其它非主属性（在2NF基础上消除传递依赖）**

第三范式（3NF）是第二范式（2NF）的一个子集，即满足第三范式（3NF）必须满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个关系中不包含已在其它关系已包含的非主关键字信息。第三范式就是属性不依赖于其它非主属性，也就是在满足2NF的基础上，任何非主属性不得传递依赖于主属性。

11 范式建模及维度建模

一般常规的数据仓库层级结构可分为：ods、dw(默认为汇总数据层，也可在细分为dwd(明细)与dw(汇总)两层)、dm共三层：

ods层：称为接口层或近源数据层，表结构与源系统表结构高度相似，通常在ods层主要会做字段的筛选，枚举值转换，编码统一，异常&缺失数据处理等操作

dw层：称为中间层，按主题建模（域->主题）的明细数据层，数据粒度与ods层一致。

dm层：称为数据集市层，集市层是按照业务主题、分主题构建出来的、面向特定部门或人员的数据集合。

维度建模源于Kimball提出的总线式的自下而上（下和上指数据的下游和上游）（DM-DW）的数据仓库架构。

特点：

1.模型结构简单，星型模型为主；

2.开发周期短，能够快速迭代；

3.维护成本较高；

范式建模源于Inmon提出的集线器的自上而下（EDW-DM）的数据仓库架构。

特点：

1.同一份数据只存放在一个地方，因此只能从一个地方获取，没有数据冗余，保证了数据一致性；

2.解耦（系统级与业务级），方便维护；

3.开发周期较长，开发成本较高；

当下的数据仓库模型架构设计中，dw层通常会采用范式建模，并且可以根据实际情况允许存在一些冗余。dm层通常会采用维度建模，因为采用维度建模构建出来的数据模型更加符合普通人的认知、易于被普通人所理解，从而有利于数据的推广使用。

12 OLAP 和OLTP



13 Mysql中的事务

1. 事务：

A（atomicity）原子性。一个事务的执行被视为一个不可分割的最小单元。事务里面的操作，要么全部成功执行，要么全部失败回滚，不可以只执行其中的一部分。

C（consistency）一致性。一个事务的执行不应该破坏数据库的完整性约束。如果上述例子中第2个操作执行后系统崩溃，保证A和B的金钱总计是不会变的。

I（isolation）隔离性。事务之间相互独立，互不干挠。

D（durability）持久性。事务提交之后，需要将提交的事务持久化到磁盘。即使系统崩溃，提交的数据也不应该丢失。

1. 实现方式：
   * 隔离性：通过锁来实现
   * 原子性和持久性：通过redo log 来实现
   * 一致性：通过undo来实现
2. redo 和 undo 比较：

     　　 都是恢复操作：

* 1. redo：恢复提交事务修改的页操作
  2. undo: 回滚行记录到某个特定版本

     　　记录内容不同：

* 1. redo: 是物理日志，记录的是物理的修改操作
  2. undo: 是逻辑日志，根据每行记录进行记录

     　　读取方式不同：

* 1. redo : 在数据库运行时，不需要读取操作(注：数据库恢复时，才用redo)
  2. undo ： 在数据库运行时，需要随机读取(注：回滚时用)

## **1，脏读**

　　脏读是指在一个事务处理过程里读取了另一个未提交的事务中的数据。

　　当一个事务正在多次修改某个数据，而在这个事务中这多次的修改都还未提交，这时一个并发的事务来访问该数据，就会造成两个事务得到的数据不一致。例如：用户A向用户B转账100元，对应SQL命令如下

update account set money=money+**100** where name=’B’; (此时A通知B)

update account set money=money - **100** where name=’A’;

　　当只执行第一条SQL时，A通知B查看账户，B发现确实钱已到账（此时即发生了脏读），而之后无论第二条SQL是否执行，只要该事务不提交，则所有操作都将回滚，那么当B以后再次查看账户时就会发现钱其实并没有转。

## **2，不可重复读**

　　不可重复读是指在对于数据库中的某个数据，一个事务范围内多次查询却返回了不同的数据值，这是由于在查询间隔，被另一个事务修改并提交了。

　　例如事务T1在读取某一数据，而事务T2立马修改了这个数据并且提交事务给数据库，事务T1再次读取该数据就得到了不同的结果，发送了不可重复读。

　　不可重复读和脏读的区别是，脏读是某一事务读取了另一个事务未提交的脏数据，而不可重复读则是读取了前一事务提交的数据。

　　在某些情况下，不可重复读并不是问题，比如我们多次查询某个数据当然以最后查询得到的结果为主。但在另一些情况下就有可能发生问题，例如对于同一个数据A和B依次查询就可能不同，A和B就可能打起来了……

## **3，虚读(幻读)**

　　幻读是事务非独立执行时发生的一种现象。例如事务T1对一个表中所有的行的某个数据项做了从“1”修改为“2”的操作，这时事务T2又对这个表中插入了一行数据项，而这个数据项的数值还是为“1”并且提交给数据库。而操作事务T1的用户如果再查看刚刚修改的数据，会发现还有一行没有修改，其实这行是从事务T2中添加的，就好像产生幻觉一样，这就是发生了幻读。

　　幻读和不可重复读都是读取了另一条已经提交的事务（这点就脏读不同），所不同的是不可重复读查询的都是同一个数据项，而幻读针对的是一批数据整体（比如数据的个数）。

　　现在来看看MySQL数据库为我们提供的四种隔离级别：

　　① Serializable (串行化)：可避免脏读、不可重复读、幻读的发生。

　　② Repeatable read (可重复读)：可避免脏读、不可重复读的发生。

　　③ Read committed (读已提交)：可避免脏读的发生。

　　④ Read uncommitted (读未提交)：最低级别，任何情况都无法保证。

　　以上四种隔离级别最高的是Serializable级别，最低的是Read uncommitted级别，当然级别越高，执行效率就越低。像Serializable这样的级别，就是以锁表的方式(类似于Java多线程中的锁)使得其他的线程只能在锁外等待，所以平时选用何种隔离级别应该根据实际情况。在MySQL数据库中默认的隔离级别为Repeatable read (可重复读)。

14 操作系统的进程调度

一、先来先去服务

　　先来先去服务调度算法是一种最简单的调度算法，也称为先进先出或严格排队方案。当每个进程就绪后，它加入就绪队列。当前正运行的进程停止执行，选择在就绪队列中存在时间最长的进程运行。该算法既可以用于作业调度，也可以用于进程调度。先来先去服务比较适合于常作业（进程），而不利于段作业（进程）。

二、时间片轮转法

　　轮转法是基于适中的抢占策略的，以一个周期性间隔产生时钟中断，当中断发生后，当前正在运行的进程被置于就绪队列中，然后基于先来先去服务策略选择下一个就绪作业的运行。这种技术也称为时间片，因为每个进程再被抢占之前都给定一片时间。

三、最短进程优先

　　最短进程优先是一个非抢占策略，他的原则是下一次选择预计处理时间最短的进程，因此短进程将会越过长作业，跳至队列头。该算法即可用于作业调度，也可用于进程调度。但是他对长作业不利，不能保证紧迫性作业（进程）被及时处理，作业的长短只是被估算出来的。

四、最短剩余时间优先

　　最短剩余时间是针对最短进程优先增加了抢占机制的版本。在这种情况下，进程调度总是选择预期剩余时间最短的进程。当一个进程加入到就绪队列时，他可能比当前运行的进程具有更短的剩余时间，因此只要新进程就绪，调度程序就能可能抢占当前正在运行的进程。像最短进程优先一样，调度程序正在执行选择函数是必须有关于处理时间的估计，并且存在长进程饥饿的危险。

五、最高响应比优先

根据比率：R=(w+s)/s （R为响应比，w为等待处理的时间，s为预计的服务时间）

　　如果该进程被立即调用，则R值等于归一化周转时间（周转时间和服务时间的比率）。R最小值为1.0，只有第一个进入系统的进程才能达到该值。调度规则为：当前进程完成或被阻塞时，选择R值最大的就绪进程，它说明了进程的年龄。当偏向短作业时，长进程由于得不到服务，等待时间不断增加，从而增加比值，最终在竞争中赢了短进程。

　　和最短进程优先、最短剩余时间优先一样，使用最高响应比策略需要估计预计服务时间。

大数据面试题：

1. **spark运行流程、源码架构**

Spark-yarn模式：

1.client向ResouceManager申请启动ApplicationMaster，同时在SparkContext初始化中创建DAGScheduler和TaskScheduler

2.ResouceManager收到请求后，在一台NodeManager中启动第一个Container运行ApplicationMaster

3.Dirver中的SparkContext初始化完成后与ApplicationMaster建立通讯，ApplicationMaster向ResourceManager申请Application的资源

4.一旦ApplicationMaster申请到资源，便与之对应的NodeManager通讯，启动Executor，并把Executor信息反向注册给Dirver

5.Dirver分发task，并监控Executor的运行状态，负责重试失败的task

6.运行完成后，Client的SparkContext向ResourceManager申请注销并关闭自己。

DAGScheduler 把一个Spark作业转换成Stage的DAG（Directed Acyclic Graph有向无环图），根据RDD和Stage之间的关系找出开销最小的调度方法，然后把Stage以TaskSet的形式提交给 TaskScheduler

DAGScheduler 决定了运行Task的理想位置，并把这些信息传递给下层的TaskScheduler。此外，DAGScheduler还处理由于Shuffle数据丢失 导致的失败，这有可能需要重新提交运行之前的Stage（非Shuffle数据丢失导致的Task失败由TaskScheduler处理）。   
TaskScheduler维护所有TaskSet，当Executor向Driver发送心跳时，TaskScheduler会根据其资源剩余情况分配 相应的Task。另外TaskScheduler还维护着所有Task的运行状态，重试失败的Task。下图展示了TaskScheduler的作用：

1. Hbase主键设计、hbase为啥比mysql快、为什么项目选用hbase

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，比如超过100字节，1000w行数据，光rowkey就要占用100\*1000w=10亿个字节，将近1G数据，这样会极大影响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。

目前操作系统都是64位系统，内存8字节对齐，控制在16个字节，8字节的整数倍利用了操作系统的最佳特性。

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，这样在数据检索的时候负载会集中在个别的RegionServer上，造成热点问题，会降低查询效率。

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。

HBase能提供实时计算服务主要原因是由其架构和底层的数据结构决定的，即由LSM-Tree(Log-Structured Merge-Tree) + HTable(region分区) + Cache决定——客户端可以直接定位到要查数据所在的HRegion server[服务器](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/keda8997110/article/details/_blank)，然后直接在服务器的一个region上查找要匹配的数据，并且这些数据部分是经过cache缓存的。

1. **Hbase读写流程，数据compact流程**

https://blog.csdn.net/whdxjbw/article/details/81107285

1. **Spark stage的切分、task资源分配、任务调度、master计算资源分配**

Spark任务会根据RDD之间的依赖关系，形成一个DAG有向无环图，DAG会提交给DAGScheduler，DAGScheduler会把DAG划分相互依赖的多个stage，划分stage的依据就是RDD之间的宽窄依赖。遇到宽依赖就划分stage,每个stage包含一个或多个task任务。然后将这些task以taskSet的形式提交给TaskScheduler运行。stage是由一组并行的task组成。

切割规则：****从后往前****，****遇到宽依赖就切割stage。****

****Task资源分配过程：****

**<https://www.jianshu.com/p/17a61ff4d65c>**

****Master计算资源分配：****

****Master是通过schedule方法进行资源调度，告知worker启动executor等。****

****一schedule方法****

****1判断master状态，只有alive状态的master才可以进行资源调度，standby是不能够调度的****

****2将可用的worker节点打乱，这样有利于driver的均衡****

****3进行driver资源调度，遍历处于等待状态的driver队列，发起driver****

1. **HDFS的在HA机制下的checkpoint**

**<https://blog.csdn.net/amber_amber/article/details/47003589>**

1. **Sparkshuffle和hadoopshuffle原理、对比**

<https://blog.csdn.net/WYpersist/article/details/79982627>

**Maprecue 的shuffle过程**

**<https://www.cnblogs.com/felixzh/p/4680808.html>**

**Spark shuffle过程**

**[https://www.cnblogs.com/itboys/p/9201750.html](https://www.cnblogs.com/itboys/p/9201750.html‘’)**

**（6）hive sql怎么转化为MapReduce的**  
Antlr定义SQL的语法规则，完成SQL词法，语法解析，将SQL转化为抽象语法树AST Tree

遍历AST Tree，抽象出查询的基本组成单元QueryBlock

遍历QueryBlock，翻译为执行操作树OperatorTree

逻辑层优化器进行OperatorTree变换，合并不必要的ReduceSinkOperator，减少shuffle数据量

遍历OperatorTree，翻译为MapReduce任务

物理层优化器进行MapReduce任务的变换，生成最终的执行计划

**（7）spark的内存管理机制**

**<https://www.cnblogs.com/yangsy0915/p/6058215.html>**

1. **spark rdd、dataframe、dataset区别**

这里主要对比Dataset和DataFrame，因为Dataset和DataFrame拥有完全相同的成员函数，区别只是每一行的数据类型不同

DataFrame也可以叫Dataset[Row],每一行的类型是Row，不解析，每一行究竟有哪些字段，各个字段又是什么类型都无从得知，只能用上面提到的getAS方法或者共性中的第七条提到的模式匹配拿出特定字段

而Dataset中，每一行是什么类型是不一定的，在自定义了case class之后可以很自由的获得每一行的信息

1. **hashpartitioner与rangePartitioner的实现**

**<https://www.cnblogs.com/liuming1992/p/6377540.html>**

1. **spark有哪几种join，join用什么算子可以替代**

**<https://www.cnblogs.com/kangoroo/p/7778962.html>**

1. **[HBase原理–所有Region切分的细节都在这里了](https://www.cnblogs.com/163yun/p/9014762.html)**

**<https://www.cnblogs.com/163yun/p/9014762.html>**

# **Kafka 高吞吐率的实现，0拷贝实现原理**

**<https://blog.csdn.net/u010039929/article/details/77934910>**

1. **zookeeper怎么保证原子性，怎么实现分布式锁**

**<https://blog.csdn.net/qiangcuo6087/article/details/79067136>**

1. **Zookeeper脑裂问题**

**<https://blog.csdn.net/kenan2012/article/details/80781594>**

1. **scala隐式转换**

**<https://www.cnblogs.com/MOBIN/p/5351900.html>**

1. **scala柯理化**

**<https://www.jianshu.com/p/5eb84e76ca6f>**

1. **spark和MR的区别：**

1、spark把运算的中间数据存放在内存，迭代计算效率更高；mapreduce的中间结果需要落地，需要保存到磁盘，

这样必然会有磁盘io操做，影响性能

2、spark容错性高，它通过弹性分布式数据集RDD来实现高效容错，RDD是一组分布式的存储在节点内存中的

只读性质的数据集，这些集合是弹性的，某一部分丢失或者出错，可以通过整个数据集的计算流程的血缘关系

来实现重建；mapreduce的话容错可能只能重新计算了，成本较高。

3、spark更加通用，spark提供了transformation和action这两大类的多个功能api，另外还有流式处理

sparkstreaming模块、图计算GraphX等等；mapreduce只提供了map和reduce两种操作，流计算以及其他

模块的支持比较缺乏。

4、spark框架和生态更为复杂，首先有RDD、血缘lineage、执行时的有向无环图DAG、stage划分等等，

很多时候spark作业都需要根据不同业务场景的需要进行调优已达到性能要求；mapreduce框架及其生态

相对较为简单，对性能的要求也相对较弱，但是运行较为稳定，适合长期后台运行。

1. **七层网络结构总结**

<https://www.cnblogs.com/wxgblogs/p/5641643.html>

## **（19）进程和线程通信方式？实际开发中，在哪些地方用上了，请说出具体场景？**

每个进程有自己的地址空间，进程间的通信一般通过操作系统的公共区进行。

线程可以看作是轻量级的进程，同一进程中的线程属于同一地址空间，所以可以直接通信，而且可以共享全局变量和内存。总之，线程之间共享数据特别容易，这是它的优点，也由此带来了同步、异步、互斥的问题。

**总结：**

线程通信，共享地址空间和数据空间，不必通过操作系统。

进程通信，需要通过操作系统，分为单机版和网络通信（通过socket）。

**进程通信方式：**

管道（pipe）：

管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。亲缘关系一般指父子进程关系。

有名管道（namedpipe）：

允许无亲缘关系的进程的通信。

信号量（semophore）：

信号量是一个计数器，用来控制多个进程对共享资源的访问。类比锁机制。

消息队列（messagequeue）：

消息队列是有消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

信号（sinal）：

用于通知接收进程某个事件已经发生。

共享内存（shared memory）：

共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的IPC方式，它是针对其他进程间通信方式运    行效率低而专门设计的。它往往与其他通信机制，如信号量配合使用，来实现进程间的同步和通信。

套接字（socket）：

用于不同机器间的进程通信。

**线程通信方式：**

锁机制：包括互斥锁、条件变量、读写锁。

互斥锁提供了以排他方式防止数据结构被并发修改的方法。

读写锁允许多个线程同时读共享数据，而对写操作是互斥的。

条件变量可以以原子的方式阻塞进程，直到某个特定条件为真为止。对条件的测试是    在互斥锁的保护下进行的。条件变量始终与互斥锁一起使用。

信号量机制(Semaphore)：

包括无名线程信号量和命名线程信号量

信号机制(Signal)：

类似进程间的信号处理。

线程间的通信目的主要是用于线程同步，所以线程没有像进程通信中的用于数据交换的通信机制。

**（20）MR怎么实现全局排序的**

Hadoop提供TotalOrderPartitioner类用于实现全局排序的功能，并且解决了OOM和数据倾斜的问题。

TotalOrderPartitioner类提供了数据采样器，对key值进行部分采样，然后按照采样结果寻找key值的最佳分割点，将key值均匀的分配到不同的分区中。

TotalOrderPartitioner类提供了三个采样器，分别是：

· SplitSampler分片采样器，从数据分片中采样数据，该采样器不适合已经排好序的数据

· RandomSampler随机采样器，按照设置好的采样率从一个数据集中采样

· IntervalSampler间隔采样机，以固定的间隔从分片中采样数据，对于已经排好序的数据效果非常好。

三个采样器都实现了K[] getSample(InputFormat<K,V> inf, Job job)方法，该方法返回的是K[]数组，数组中存放的是根据采样结果返回的key值，即分隔点，MapRdeuce就是根据K[]数组的长度N生成N-1个分区partition数量，然后按照分割点的范围将对应的数据发送到对应的分区中。

**（21）FLUME必须了解**

<https://www.cnblogs.com/zhangyinhua/p/7803486.html>

1. **KAFKA保持数据的一致性**

**Kafka的消息传递语义（重要，若问Kafka基本必问），换种问法，Kafka怎么保持数据的一致性（怎么保证数据0丢失）？**

1.幂等写入（idempotent writes）

需要设置好唯一主键等，比如用redis、mysql

再比如每次往一个目录覆盖写数据，这样主键不容易获取。

一次语义：幂等写入

当获取到数据后，先写到mysql，再保存offset，如果在写到mysql数据后，在保存offset之前宕机，重启作业后也不会影响一次语义，因为会在mysql重复更新。

注：在软件开发领域，幂等写入即为同样的请求被执行一次与连续执行多次的效果是一样的，服务器的状态也是一样的，实际上就是接口的可重复调用（包括时间和空间上两个维度）。

2.事务控制

保证数据和offset在同一个事务里面，比如用mysql，这样需要事务存储的支持。

3.自己实现Exactly-once，offset和数据绑定保存等。

**（22）sqoop应用与原理**

**<https://blog.csdn.net/lusijie1/article/details/78729448>**

1. **Nosql**

**<https://baike.baidu.com/item/NoSQL/8828247?fr=aladdin>**

# **(24)**String StringBuffer StringBuilder的区别以及对String不变性的理解

都是final类，不可以被继承

String是字符串常量；StringBuffer,StringBuilder是字符串变量

String长度是不可变的；StringBuffer,StringBuilder长度是可变的

StringBuffer是线程安全的；StringBuilder不是线程安全的。StringBuilder比StringBuffer快

String不变性的理解：

String类是被final修饰的，不能被继承

在用+号连接字符串时会创建一个新的字符串

String s = new String("helloworld"); 可能创建两个对象，也可能创建一个对象。如果静态区有“helloworld”字符串常量对象的话，则只在堆里创建一个对象；如果静态区没有这个对象，那个在堆和静态区都会创建对象

**(25)如果重写了equals()不重写hashCode()会发生什么**

在存储散列集合时(如Set类)，如果原对象.equals(新对象)，但没有对hashCode重写，即两个对象拥有不同的hashcode，则在集合中将会存储两个值相同的对象，从而导致混淆。因此在重写equals方法时，必须重写hashcode方法。

1. **MySQL binlog格式**

binlog的格式也有三种：STATEMENT、ROW、MIXED 。

1、STATMENT模式：基于SQL的复制(statement-based replication, SBR)，每一条会修改数据的sql语句会记录到binlog中。

优点：不需要记录每一条SQL语句与每行的数据变化，这样子binlog的日志也会比较少，减少了磁盘IO，提高性能。

缺点：在某些情况下会导致master-slave中的数据不一致(如sleep()函数， last\_insert\_id()，以及user-defined functions(udf)等会出现问题)

2、基于行的复制(row-based replication, RBR)：不记录每一条SQL语句的上下文信息，仅需记录哪条数据被修改了，修改成了什么样子了。

优点：不会出现某些特定情况下的存储过程、或function、或trigger的调用和触发无法被正确复制的问题。

缺点：会产生大量的日志，尤其是alter table的时候会让日志暴涨。

3、混合模式复制(mixed-based replication, MBR)：以上两种模式的混合使用，一般的复制使用STATEMENT模式保存binlog，对于STATEMENT模式无法复制的操作使用ROW模式保存binlog，MySQL会根据执行的SQL语句选择日志保存方式。

1. **星型模型与雪花模型**

**<https://blog.csdn.net/nisjlvhudy/article/details/7889422>**

1. **数据库索引 B树与B+树**

**<https://www.cnblogs.com/vincently/p/4526560.html>**

1. **ConcurrentHashMap实现原理**

<http://www.importnew.com/28263.html>

https://www.jianshu.com/p/95a9a82d7a1c

1. 线程死锁的原因及避免

产生死锁的四个必要条件

互斥条件：

进程要求对所分配的资源（如打印机）进行排他性控制，即在一段时间内某资源仅为一个进程所占有。此时若有其他进程请求该资源，则请求进程只能等待。

不可剥夺条件:

进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走，即只能由获得该资源的进程自己来释放（只能是主动释放)。

请求与保持条件：

进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源已被其他进程占有，此时请求进程被阻塞，但对自己已获得的资源保持不放。

循环等待条件:

存在一种进程资源的循环等待链，链中每一个进程已获得的资源同时被 链中下一个进程所请求。即存在一个处于等待状态的进程集合{Pl, P2, …, pn}，其中Pi等 待的资源被P(i+1)占有（i=0, 1, …, n-1)，Pn等待的资源被P0占有，如图2-15所示。

直观上看，循环等待条件似乎和死锁的定义一样，其实不然。按死锁定义构成等待环所 要求的条件更严，它要求Pi等待的资源必须由P(i+1)来满足，而循环等待条件则无此限制。 例如，系统中有两台输出设备，P0占有一台，PK占有另一台，且K不属于集合{0, 1, …, n}。

预防死锁

破坏“互斥”条件:

就是在系统里取消互斥。若资源不被一个进程独占使用，那么死锁是肯定不会发生的。但一般来说在所列的四个条件中，“互斥”条件是无法破坏的。因此，在死锁预防里主要是破坏其他几个必要条件，而不去涉及破坏“互斥”条件。

注意：互斥条件不能被破坏，否则会造成结果的不可再现性。

破坏“占有并等待”条件:

破坏“占有并等待”条件，就是在系统中不允许进程在已获得某种资源的情况下，申请其他资源。即要想出一个办法，阻止进程在持有资源的同时申请其他资源。

方法一：创建进程时，要求它申请所需的全部资源，系统或满足其所有要求，或什么也不给它。这是所谓的 “ 一次性分配”方案。

方法二：要求每个进程提出新的资源申请前，释放它所占有的资源。这样，一个进程在需要资源S时，须先把它先前占有的资源R释放掉，然后才能提出对S的申请，即使它可能很快又要用到资源R。

破坏“不可抢占”条件：

破坏“不可抢占”条件就是允许对资源实行抢夺。

方法一：如果占有某些资源的一个进程进行进一步资源请求被拒绝，则该进程必须释放它最初占有的资源，如果有必要，可再次请求这些资源和另外的资源。

方法二：如果一个进程请求当前被另一个进程占有的一个资源，则操作系统可以抢占另一个进程，要求它释放资源。只有在任意两个进程的优先级都不相同的条件下，方法二才能预防死锁。

破坏“循环等待”条件：

破坏“循环等待”条件的一种方法，是将系统中的所有资源统一编号，进程可在任何时刻提出资源申请，但所有申请必须按照资源的编号顺序（升序）提出。这样做就能保证系统不出现死锁。

1. **数据库引擎 myisam和innoDB**

<https://www.cnblogs.com/y-rong/p/8110596.html>

1. mysql索引为什么选择B+树

<https://blog.csdn.net/majiawenzzz/article/details/81098870>（32）接口和抽象类的区别

<http://www.importnew.com/12399.html>

<https://blog.csdn.net/gongxiao1993/article/details/82055007>

(33)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

思考过程

（1）参见我的其他大数据面试题博文。此处1G文件远远大于1M内存，分治法，先hash映射把大文件分成很多个小文件，具体操作如下：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f0,f1,...,f4999)中，这样每个文件大概是200k左右（每个相同的词一定被映射到了同一文件中）

（2）对于每个文件fi，都用hash\_map做词和出现频率的统计，取出频率大的前100个词（怎么取？topK问题，建立一个100个节点的最小堆），把这100个词和出现频率再单独存入一个文件

（3）根据上述处理，我们又得到了5000个文件，归并文件取出top100（Top K 问题，比较最大的前100个频数）

堆排序找Top K

借助堆这个数据结构，找出Top K，时间复杂度为N‘logK。即借助堆结构，我们可以在log量级的时间内查找和调整/移动。因此，维护一个K(该题目中是10)大小的小根堆，然后遍历300万的Query，分别和根元素进行对比。所以，我们最终的时间复杂度是：O（N） + N' \* O（logK），（N为1000万，N’为300万）。

总结：

要解决该问题首先要进行分类，把重复出现的IP都放到一个文件里面，一共分成100份，这可以通过把IP对100取模得到，具体方法如把IP中的点转化为整型long型变量，这样取模为0,1,2...99的IP都分到一个文件了，但是要考虑一个问题，如果某个文件的IP取余之后还是特别多无法放入内存中，可以再对这一类IP做一次取模，直到每个小文件足够载入内存为止。这个分类很关键，如果是随便分成100份，相同的IP被分在了不同的文件中，接下来再对每个文件统计次数并做归并，这个思路就没有意义了，起不到“大而化小，各个击破，缩小规模，逐个解决”的效果了。

接下来把每个小文件载入内存，建立哈希表将每个IP作为关键字映射为出现次数，这个哈希表建好之后也得先写入硬盘，因为内存就那么多，一共要统计100个文件。

在统计完100个文件之后，我再建立一个小顶堆，大小为100，把建立好并存在硬盘哈希表载入内存，逐个对出现次数排序，挑出出现次数最多的100个，由于次数直接和IP是对应的，找出最多的次数也就找出了相应的IP。

1. CAS机制

<https://blog.csdn.net/qq_32998153/article/details/79529704>

1. 三次握手四次挥手

<https://baijiahao.baidu.com/s?id=1593714120815701015&wfr=spider&for=pc>

1. spark算子总结

<https://blog.csdn.net/fortuna_i/article/details/81170565>

1. 聚簇索引和非聚簇索引

<https://blog.csdn.net/u010727189/article/details/79399384>

1. **HIVE数据倾斜及解决**

**<https://www.cnblogs.com/qingyunzong/p/8847597.html>**

1. **redis持久化**

**<https://blog.csdn.net/bible_reader/article/details/84138665>**

1. **elasticsearch 倒排索引和原理**

**<https://www.cnblogs.com/dreamroute/p/8484457.html>**

**<https://blog.csdn.net/andy_wcl/article/details/81631609>**

1. **java线程池及参数详解**

**<https://www.cnblogs.com/CarpenterLee/p/9558026.html>**