# HDFS面试题整理

1. HDFS是什么

* hdfs（Hadoop Distribute File System）是分布式文件系统。
* 分布式文件系统(distributed file system)是指文件系统管理的物理存储资源不一定直接链接在本地节点上，而是通过计算机网络与节点相连，可让多机器上的多用户分享文件和存储空间。
* 说简单点就是将一个大文件分布式存储在不同的机器节点上

# HDFS架构

* HDFS为主从架构，由单个NameNode和多个DataNode组成，另外还包括SecondaryNameNode。
* NameNode主要负责管理dataNode和记录元数据。由FsImage和EditLog两部分组成。
  + FsImage用户维护文件目录和元数据
  + EditLog则记录所有对文件操作的操作。
* DataNode负责存储数据，同时定期与NameNode进行通信，汇报自己存储的信息。
* SecondaryNameNode负责合并日志，定期将新的FsImage拷贝回dataNode。

1. HDFS相关概念

* block：文件上传前需要分块，这个块就是block，一般为128MB，当然你可以去改，不顾不推荐。因为块太小：寻址时间占比过高。块太大：Map任务数太少，作业执行速度变慢。它是最大的一个单位。
* packet：packet是第二大的单位，它是client端向DataNode，或DataNode的PipLine之间传数据的基本单位，默认64KB。
* chunk：chunk是最小的单位，它是client向DataNode，或DataNode的PipLine之间进行数据校验的基本单位，默认512Byte，因为用作校验，故每个chunk需要带有4Byte的校验位。所以实际每个chunk写入packet的大小为516Byte。由此可见真实数据与校验值数据的比值约为128 : 1。（即64\*1024 / 512）

1. HDFS中小文件过多有什么坏处？

* 主要是NameNode内存会受限。由于NameNode会将文件系统的元数据都存储在内存中，因此所能存储的文件总量受限于NameNode的内存总容量。每个文件、目录和数据块大约占150个字节，过多的小文件会大量消耗NameNode的内存量
* 增加NameNode寻址的时间

1. 怎么处理小文件过多的问题？

* 数据未上传到HDFS前的处理：从根源上解决问题，每小时抽取一次数据改为每天抽取一次数据积累数据量
* 数据已上传到HDFS：通过程序将小文件和并成大文件

1. HDFS中NameNode内存总共有多大

* Hadoop 2.x系列，默认为2000m
* Hadoop 3.x系列，动态分配内存，最小值为1G，每增加100万个Block，增加1G内存

1. HDFS支持并发写入么？

* 不支持，客户端想写数据必须续约（许可证），同时只能有一个客户端写数据，直到这个客户端契约过期，下个客户端才可以获取契约并写入数据，详见写数据流程

1. HDFS的特性与优点

* 大数据分而治之，主要针对一次写入多次读取的场景
* 分布式，扩容方便
* 高可靠，稳定
* 成本低，只需要几台磁盘大点的服务器

1. NameNode 和 SecondaryNameNode的区别与联系

* NameNode负责管理元数据
* SecondaryNameNode见名字像是NameNode的备份，实则不是，这也是hdfs命名的缺陷，SecondaryNameNode主要负责 standby namenode中的checkporint动作，将内存中的fsimage合并到磁盘上的fsimage中

1. fsimage与editlogs是的作用？

* NameNode主要维护了两个文件，fsimage与editlogs
* fsimage：最近元数据的检查点，包括数据块描述信息、修改时间、访问时间、元数据的权限、副本个数等。相当于快照
* editlogs：所有元数据，写入、删除、移动的操作，都会记录在这里

1. HDFS有哪些设计不合理的地方

* 数据不能实时处理
* 不能并发写入
* 小文件太多会占用过多namenode中的内存，增加namenode的寻址地址的时间

1. 简述hadoop的二级排序原理

* 通过自定义分区实现，因为如果采用默认分区策略很有可能产生数据倾斜现象
* 哪个key分配到哪个分区这个过程自定义控制，将key分区好后，将key进行排序
* 再将每个key中的值进行排序

1. MapReduce跑的慢的原因

* key设计不合理 或 数据倾斜
* 小文件过多
* 数据太大导致spill和merge的次数过多
* Map和Reduce的个数设计的不合理

1. 什么是数据倾斜, 数据倾斜的原因

* key分布不均匀
* 业务数据本身的特性
  + 数据混乱，空值过多，可将空值的key变成一段字符串+随机数，把倾斜的数据分布到不同的reduce上，由于值是null关联不上( where 值 != null )，不会影响最终结果
* 建表时考虑不周
  + 表设计的不好，没有按照建模思想建表
* sql 语句本身就有数据倾斜
  + join
    - 做好列裁剪和数据过滤操作
    - map join ，大小表join ，让小表先进入到内存
  + group by
  + count distinct
    - 可用sum() + group by 方式替换

1. Hadoop Rpc 原理

* Rpc 是远程（不同进程）间过程（方法）的调用 的缩写形式，分为客户端和服务端，服务端会一直开启一个服务等待别人的调用，客户端获取服务端的代理，调用服务端的方法后，方法在服务端上执行
* Hadoop Rpc是通过构建者设计模式构建的

# hdfs故障检测机制

故障的类型主要有以下三种，针对这三种故障类型，HDFS提供了不同的故障检测机制：

* 针对DataNode失效问题， HDFS使用了心跳机制，DataNode定期向NameNode发送心跳信息，NameNode根据心跳信息判断DataNode是否存活；
* 针对网络故障而导致无法收发数据的问题，HDFS提供了ACK的机制，在发送端发送数据后，如果没有收到ACK并且经过多次重试后仍然如此，则认为网络故障；
* 针对数据损坏问题，所有DataNode会定期向NameNode发送自身存储的块清单，在传输数据的同时会发送总和校验码，NameNode依次来判断数据是否丢失或损坏。

1. **hdfs的文件上传流程**

* 首先客户端通过Distributed FileSystem模块向NameNode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
* NameNode返回是否可以上传。
* 如果文件大于128M则分块存储，客户端请求第一个 Block上传到哪几个DataNode服务器上。
* NameNode根据副本储存策略返回3个DataNode节点，假如为dn1、dn2、dn3。
* 客户端通过FSDataOutputStream模块请求dn1上传数据，dn1收到请求调用dn2，dn2调用dn3，建立通信管道完成,dn1、dn2、dn3逐级应答客户端。
* 客户端以Packet为单位往dn1上传第一个Block数据,dn1收到Packet就会传给dn2，dn2传给dn3；dn1,dn2,dn3每接收packet会放入一个待写队列等待写入数据，落盘。
* 当一个Block传输完成之后，客户端再次请求NameNode上传第二个Block的服务器，重复执行3-6步。

1. **文件的下载流程**

* 客户端通过Distributed FileSystem向NameNode请求下载文件，NameNode通过查询元数据，找到文件块所在的DataNode地址。
* 挑选一台DataNode（就近原则，然后随机）服务器，请求读取数据。
* DataNode开始传输数据给客户端（从磁盘里面读取数据输入流，以Packet为单位来做校验）。
* 客户端以Packet为单位接收，先在本地缓存，然后写入目标文件。

1. HDFS在上传文件的时候，如果其中一个DataNode突然挂掉了怎么办

* 客户端上传文件时与DataNode建立pipeline管道，管道的正方向是客户端向DataNode发送的数据包，管道反向是DataNode向客户端发送ack确认，也就是正确接收到数据包之后发送一个已确认接收到的应答。
* 当DataNode突然挂掉了，客户端接收不到这个DataNode发送的ack确认，客户端会通知NameNode，NameNode检查该块的副本与规定的不符，NameNode会通知DataNode去复制副本，并将挂掉的DataNode作下线处理，不再让它参与文件上传与下载。

1. HDFS的常用命令是什么?

* hdfs:查看hdfs命令所有参数
* hdfs dfsadmin -report //报告当前集群的情况
* hdfs dfsadmin -safemode //安全模式
* hdfs dfsamin -safemode leave // leave代表关闭,安全模式关闭才是读写正常的
* hdfs dfsadmin -safemode enter //进入安全模式.如正在做集群维护的时候,就可以手动进入安全模式,维护完就离开
* hdfs fasck //对hdfs做检查.如发现Target Replicas is 3 but 1 live replica(s) 代表有副本丢失.
* hdfs balancer //集群平衡. DN1与DN2间节点平衡.
* hdfs balancer -policy datanode -threshold 10 //让每个节点磁盘使用率减去平均磁盘使用率小于10%
* hdfs diskbalancer //磁盘平衡. DN1中,多块磁盘的平衡,前提条件:hdfs.disk.balancer.enabled = true;

# MapReduce面试题整理

1. **MapReduce核心思想** **分而治之，先分后和（只有一个模型）** 【将一个大的、复杂的工作或任务，拆分成多个小的任务，并行处理，最终进行合并。适用于大量复杂的、时效性不高的任务处理场景（大规模离线数据处理场景）。】 Map负责数据拆分 map: [k1,v1] → [(k2,v2)] Reduce负责数据合并 reduce: [k2, {v2,…}] → [k3, v3]

**2.偏移量** ：指的是每行行首字母移动到文办的最前面需要一定的字符。

**3.Suffle包含哪些步骤** partition(分区)-sort(排序)-combiner（局部聚合）=group(分组)

**4.MR从读取数据开始到将最终结果写入HDFS经过哪些步骤：**

**第一步：InputFormat** InputFormat 在HDFS文件系统中读取要进行计算的数据输出 给Split

**第二步：Split** Split 将数据进行逻辑切分，切分成多个任务。输出给RR

**第三步：RR( RecordReader)** RR 将切分后的数据转换成key value进行输出 key : 每 一行行首字母的偏移量 value: 每一行数据输出给Map

**第四步：Map** 接收一条一条的数据（有多少行数据Map运行多少次，输出的次数根据 实际业务需求而定）根域业务需求编写代码 Map的输出是 key value的 list 输出给 Shuffle（partition） ---------------------------------------Map------------------------------------------------------

**第五步： partition** partition: 按照一定的规则对 \*\*key value的 list进行分区输出给 Shuffle（sort）

**第六步：Sort** Sort :对每个分区内的数据进行排序。 输出给Shuffle（Combiner）

**第七步：Combiner** Combiner: 在Map端进行局部聚合（汇总） 目的是为了减少网络 带宽的开销输出给Shuffle（Group）

**第八步：Group** Group: 将相同key的key提取出来作为唯一的key 将相同key对应 的value提取出来组装成一个value 的List 输出给Shuffle（reduce） ------------------------------------Shuffle--------------------------------------------

**第九步：reduce reduce：** 根据业务需求对传入的数据进行汇总计算。输出给Shuffle （outputFormat）

**第十步：outputFormat** outputFormat:将最终的额结果写入HDFS

**5.如何设置ReduceTask的数量** reducetask的设置： job.setNumReduceTasks(2);

**6.combiner的作用** Map结束后，在Map端进行局部聚和。 作用：较少网络传入次数，降低了网络开销。

**7.combiner运行在MapReduce的哪一端？** map 每一个map都可能会产生大量的本地输出，Combiner的作用就是对map端的输出先做一次合并，以减少在map和reduce节点之间的数据传输量，以提高网络IO性能.

**8.Maptask的数量是可以人为设置的吗？** 不可以。

**9.Maptask的个数由什么决定？**一个job的map阶段MapTask并行度（个数），由客户端提交job时的切片个数决定。

**10.Shuffle阶段的Partition分区算法是什么** 计算逻辑：对map输出的key 取哈希值，用这个哈希值与reducetask的值取余。 余几，就将这个key，value放在对应的分区编号里（分区有多个编号）

**11.Split逻辑切分数据，节分大小是多大？** hdfs数据块大小是128 ，split 逻辑切分数据块大小是128M HDFS 128 是存储层面的概念， 是切分数据的分界点。 split 128 是一个逻辑切分。 这两个128没有关系。

**12内存角度介绍Map的输出到Reduce的输入的过程。** **1.Map 输出数据到内存:** map输出的数据写入环形缓冲区（内存），缓冲区的默认大小是100M（可修改）。当数据达到阈值（默认0.8-可修改）时，环形缓冲区进行flash, 环形缓冲区：数据在输出的同时，数据也可以写入空余的空间内。 当flash的数据个数达到一定的数量时（默认4个）。对数据进行合并（merge）。 **2:Reduce在Map拷贝数据** Map 输出的结果写入本地，reduce主动发出拷贝进程到Map端拷贝数据。 reduce获取[大数据](https://cloud.tencent.com/solution/bigdata?from=10680" \t "_blank)后，将数据写入内存，当数据达到阈值时进行flash. 当flash的个数达到一定的量时，进行合并，最终发送给reduce

**13.最优的Map效率是什么？** Map端的最高效率:尽量减少环形缓冲区flush的次数（减少磁盘IO的使用次数）

**14.最优的reduce是什么？** 1.尽量减少环形缓冲区flush的次数 2.尽量将所有的数据在内存计算。

**15.在MapReduce阶段，有哪些优化的点？（至少两个点）** 1、加大环形缓冲区的内存 2、增大缓冲区阈值的大小 （考虑剩余的空间是不是够系统使用） 3、对输出的进行压缩（压缩-解压的过程会消耗CPU）

**16.集群优化的核心思路是什么？** 在网络带宽，磁盘IO是瓶颈的前期下 能不使用IO网络就不使用，在必须使用的前提下，能少用就少用。 所有的只要能够减少网络带宽的开销，只要能够减少磁盘IO的使用的次数的配置项，都是集群调优的可选项。 （可选项包括：软件层面【系统软件和集群软件】，硬件层面，网络层面）

**17.如果没有定义partitioner，那数据在被送达reducer前是如何被分区的？**

如果没有自定义的 partitioning，则默认的 partition 算法，即根据每一条数据的 key 的 hashcode 值摸运算（%）reduce 的数量，得到的数字就是“分区号“。

**18.什么样的计算不能用mr来提速？**

1）数据量很小。2）繁杂的小文件。3）索引是更好的存取机制的时候。

4）事务处理。5）只有一台机器的时候。

**19.MapReduce Framework 中的分布式缓存是什么？请说明。**

答：分布式缓存是 MapReduce 框架的重要组成部分。它用于在执行期间跨操作缓存文件，并确保更快地执行任务。该框架使用分布式缓存来存储在该特定节点上执行任务经常需要的重要文件。

1. **如何减少Hadoop Map端到Reduce端的数据传输量**

1.使用combiner函数

2.减少map个数(修改参数)

3.减少reduce个数。

1. **简述Shuffle过程，如何对其IO进行优化**

Shuffle过程包括collect，sort，spill，merge，copy，Merge Sort的过程

Collect过程将Map的结果以kv的形式，放在一个环形数组kvbuffer中，以尽可能有效利用内存空间。

Sort将kvbuffer中的kv按照partition和key值进行升序排列，排序只移动索引。排序结果为kvmeta中的数据，按照partition为单位聚集，同一partition内按照key有序。

Spill创建磁盘文件，把kvmeta挨个partition输出到磁盘中。

Merge，如果map输出较大，可能有多个spill，merge要将这些out和index文件按照相同的key，merge成group。

Copy，reduce端通过http向map任务拖取数据，每个节点有一个httpserver，读取相应Map输出文件对应这个reduce的部分，通过网络流输出给reduce

Merge Sort，和map端类似，Map输出数据是有序的，merge sort进行一次归并排序。Reduce端一边进行copy，一边merge sort。

**Shuffle IO优化**：

Shuffle如果生成map reduce的数量的乘积个临时文件，将面临极大IO压力。

可以使用sort shuffle，map端输出M个数据文件和M个索引文件，reducer直接访问有序的数据，从而减少文件数目。

而在map和reduce数量不多时，sort shuffle会多此一举，可以退化为hash shuffle，让文件数和map数脱钩。

# HIVE面试题整理

Hive面试题整理：

1. hive中存的是什么？

表

存的是和hdfs的映射关系，hive是逻辑上的数据仓库，实际上操作的都是hdfs上的文件，hql就是用sql语法来写的MR程序

1. hive和关系型数据库的关系？

没有关系，hive是数据仓库，不能和数据库一样直接进行curb操作。

是一次性进行多次读取的操作，可以看成是ETL的工具。

1. hive和hbase的关系

Hbase是Hadoop database的简称，是一种NoSQL数据库，主要适用于海量明细数据（十亿、百亿）的随机实时查询，如日志明细、交易清单、轨迹行为等。

Hive是Hadoop数据仓库，通过SQL来处理和计算HDFS的数据，Hive会将SQL翻译为Mapreduce来处理数据，适用于离线的批量数据计算。

在大数据架构中，Hive和HBase是协作关系，在数据引入到数据存储上密切配合，共同完成任务，通过ETL工具将数据源抽取到HDFS存储；通过Hive清洗、处理和计算原始数据；HIve清洗处理后的结果，如果是面向海量数据随机查询场景的可存入Hbase；数据应用从HBase查询数据。

1. hive和hbase的区别

Hive是为了简化mapreduce操作而生的，hive是hadoop的一个数据仓库工具，可以将结构化工具映射成一张表数据库，并提供sql查询功能，hive适用于离线查询，因为查询时间较长

Hbase是为查询而生的，hbase是hadoop的数据库，是一个非关系型数据库，非常适合进行大数据的实时查询

1. 试说明hive中sort by, order by,cluster by,distribute by各代表什么意思

Order by：会对输入做全局排序，因此只有一个reducer（多个reducer无法保证全局有序），只有一个reducer会导致输入过大时，需要较长的计算时间。

Sort by：不是全局排序，其在数据进入reducer前完成排序，

Distribute by：根据指定字段对数据进行划分，输出到不同的reduce中

cluster by：除了Distribute by的功能外还具有Sort by的功能

1. hive底层和数据库交互的原理

Hive的查询功能是由hdfs+mapreduce结合起来实现的

Hive和mysql的关系：只是借助mysql来存储hive中的表的元数据的信息，成为metastore

1. hive内部表和外部表的区别

未被external修饰的是内部表，被external修饰的为外部表。

内部表数据由Hive自身管理，外部表数据由HDFS管理；

内部表数据存储的位置是hive.metastore.warehouse.dir（默认：/user/hive/warehouse）， 外部表数据的存储位置由自己制定（如果没有LOCATION，Hive将在HDFS上 的/user/hive/warehouse文件夹下以外部表的表名创建一个文件夹，并将属于这个表的数据存 放在这里）；

删除内部表会直接删除元数据（metadata）及存储数据；删除外部表仅仅会删除元数据，HDFS上的文件并不会被删除。

1. hive有索引吗

Hive支持索引（3.0版本之前），但是Hive的索引与关系型数据库中的索引并不相同。并且Hive索引提供的功能很有限，效率也并不高，因此Hive索引很少使用。

索引适用的场景：

适用于不更新的静态字段。以免总是重建索引数据。每次建立、更新数据后，都要重建索引以构建索引表。

1. 为什么要对数据仓库进行分层

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。如果不分层的话，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. hive小文件过多怎么办？

使用 hive 自带的 concatenate 命令，自动合并小文件

调整参数减少Map数量

减少Reduce的数量

使用hadoop的archive将小文件归档

1. hive的优化有哪些？

数据存储及压缩

通过调参优化

有效地减小数据集将大表拆分成子表；结合使用外部表和分区表

SQL优化

1. 写出Hive中split、coalesce及collect\_list函数的用法（可举例）？

split将字符串转化为数组，即：split('a,b,c,d' , ',') ==> ["a","b","c","d"]。

coalesce(T v1, T v2, …) 返回参数中的第一个非空值；如果所有值都为 NULL，那么返回NULL。

collect\_list列出该字段所有的值，不去重 => select collect\_list(id) from table。

1. 所有的Hive任务都会有MapReduce的执行吗？

不是，从Hive0.10.0版本开始，对于简单的不需要聚合的类似SELECT from LIMIT n语句，不需要起MapReduce job，直接通过Fetch task获取数据。

Fetch抓取是指，Hive中对某些情况的查询可以不必使用MapReduce计算。例如：SELECT \* FROM employees;在这种情况下，Hive可以简单地读取employee对应的存储目录下的文件，然后输出查询结果到控制台。

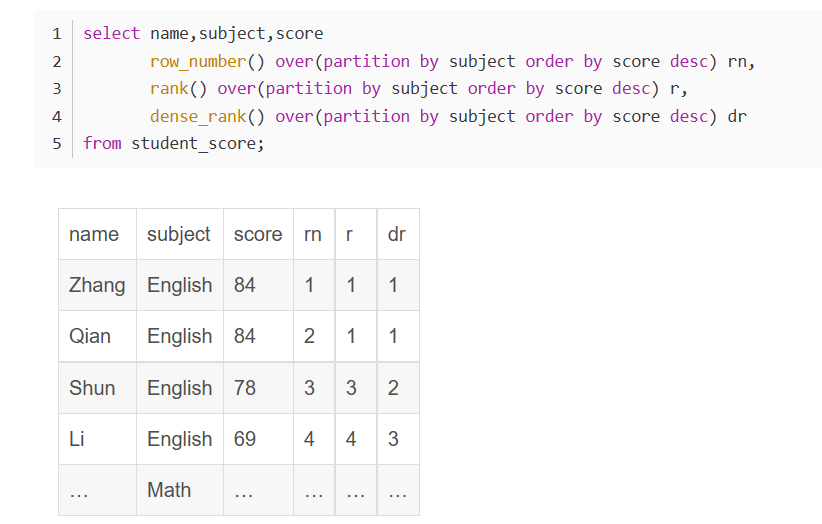
1. row\_number()，rank()和dense\_rank()的区别

都有对数据进行排序的功能

row\_number()：根据查询结果的顺序计算排序，多用于分页查询

rank()：排序相同时序号重复，总序数不变

dense\_rank()：排序相同时序号重复时，总序数减少



1. Hive导入数据的五种方式

1. Load方式，可以从本地或HDFS上导入，本地是copy，HDFS是移动本地：load data local inpath ‘/root/student.txt’ into table student;

HDFS：load data inpath ‘/user/hive/data/student.txt’ into table student;

2. Insert方式，往表里插入

insert into table student values(1,’zhanshan’);

3. As select方式，根据查询结果创建表并插入数据

create table if not exists stu1 as select id,name from student;

4. Location方式，创建表并指定数据的路径

Create external if not exists stu2 like student location '/user/hive/warehouse/student/student.txt';

5. Import方式，先从hive上使用export导出在导入

import table stu3 from ‘/user/export/student’;

16、Hive 有哪些方式保存元数据，各有哪些优缺点。

1)存储于 derby 数据库，此方法只能开启一个hive客户端，不推荐使用

2)存储于mysql数据库中，可以多客户端连接，推荐使用。

17、数据倾斜造成的原因有哪些？解决方案是什么？

1、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。

解决方案：把数据类型转换成字符串类型

1. 在生产环境经常会用大量空值数据进入到一个reduce中去，导致数据倾斜

解决方案：自定义分区，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。

注意：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

18、Union与Union all区别

union会将联合的结果集去重，效果较union all差

union all 不会对结果集去重，所以效率高

1. Hive分区分桶的区别

Hive分区：是指按照数据表的某列或者某些列分为多个区，区从形式上可以理解为文件夹，比如我们要收集某个大型网站的日志数据，一个网站每天的日志数据存在同一张表上，由于每天会生成大量的日志，导致数据表的内容巨大，在查询时进行全表扫描耗费的资源非常多。那其实这个情况下，我们可以按照日期对数据表进行分区，不同日期的数据存放在不同的分区，在查询时只要指定分区字段的值就可以直接从该分区查找。

Hive分桶：分桶是相对分区进行更细粒度的划分。分桶将整个数据内容安装某列属性值得hash值进行区分，如要安装name属性分为3个桶，就是对name属性值的hash值对3取摸，按照取模结果对数据分桶。如取模结果为0的数据记录存放到一个文件，取模为1的数据存放到一个文件，取模为2的数据存放到一个文件。

1. Hive的执行顺序

from --> join on --> where --> group by --> 聚合函数 -->having -->select -->distinct--> order by

1. Hive文件存储格式

支持 Apache Hadoop 中使用的几种熟悉的文件格式，比如 TextFile、SequenceFile、RCFile、Avro、ORC、ParquetFile等。

TEXTFILE 和 SEQUENCEFILE 的存储格式都是基于行存储的，ORC 和 PARQUET 是基于列式存储的。

**第一种：TextFile**

1、存储方式：行存储。默认格式，如果建表时不指定默认为此格式。

2、每一行都是一条记录，每行都以换行符"\n"结尾。默认是不压缩，但可以采用多种压缩方式，但是部分压缩算法压缩数据后生成的文件是不支持split。

3、可结合Gzip、Bzip2等压缩方式一起使用（系统会自动检查，查询时会自动解压）,推荐选用可切分的压缩算法。

4、该类型的格式可以识别在hdfs上的普通文件格式（如txt、csv），因此该模式常用语仓库数据接入和导出层；

5、无法区分数据类型，各个字段都被认为是文本，但需要制定列分隔符和行分隔符。

**第二种：Sequence File**

1、SequenceFile是一种二进制文件，以<key,value>的形式序列化到文件中。存储方式：行存储；

2、支持三种压缩选择：NONE、RECORD、BLOCK。RECORD压缩率低，一般建议使用BLOCK压缩。

3、优势是文件和Hadoop API的MapFile是相互兼容的

4、缺点是由于该种模式是在textfile基础上加了些其他信息，故该类格式的大小要大于textfile，现阶段基本上不用。

**第三种：RC File**

1、存储方式：数据按行分块，每块按照列存储 。

A、首先，将数据按行分块，保证同一个record在一个块上，避免读一个记录需要读取多个block。

B、其次，块数据列式存储，有利于数据压缩和快速的列存取。

2、相对来说，RCFile对于提升任务执行性能提升不大，但是能节省一些存储空间。可以使用升级版的ORC格式。

**第四种：ORC File**

1、存储方式：数据按行分块，每块按照列存储

2、Hive提供的新格式，属于RCFile的升级版，性能有大幅度提升，而且数据可以压缩存储，压缩快，快速列存取。

3、ORC File会基于列创建索引，当查询的时候会很快，现阶段主要使用的文件格式。

**第五种：Parquet File**

1、存储方式：列式存储。

2、Parquet对于大型查询的类型是高效的。对于扫描特定表格中的特定列查询，Parquet特别有用。

Parquet一般使用Snappy、Gzip压缩，默认是Snappy。

1. Hive连接优化
2. MapReduce任务优化：

当对3个或者更多个表进行 JOIN 连接时，如果每个 ON 子句都使用相同的连接键的话，那么只会产生一个 Map Reduce job。

即Hive通过一个优化可以在同一个Map Reduce job 中连接多个表（需要每个子句中有相同的连接键）。

1. JOIN连接时表的顺序（内连接）

Hive假定查询中最后一个表是最大的那个表，在对每行记录进行连接操作时，它会尝试将其他表缓存起来，然后扫描最后那个表进行计算。因此用户需要保证连续查询中表的大小从左往右是依次增加的。

用户也可以人为显示告诉查询优化器哪张表是大表，即使它在查询中不是位于最后面的

1. 连接前通过WHERE过滤数据加快查询数据，最直接的方式是使用嵌套SELECT语句：
2. map-side JOIN 优化含有小表的连接：

如果所有表中只有一张表是小表，则可以在最大的表通过mapper的时候将小表完全放到内存中。Hive可以在map端执行连接过程（称为mao-side JOIN），这是因为

Hive可以和内存中的小表逐一匹配，从而省略常规连接操作需要的reduce过程。不仅减少了reduce过程，有时也可以同时减少map过程的执行步骤。

1. Bucket Map Join

Bucket map join 需要待连接的两个表在连接字段上进行分桶（每个分桶对应 hdfs上的一个文件），而且小表的桶数需要时大表桶数的倍数。

1. 设置属性hive.optimize.bucketmapjoin= true控制 hive 执行 bucket map join；
2. 对小表的每个分桶文件建立一个 hashtable，并分发到所有做连接的 map 端；
3. map 端接受了 N（N 为小表分桶的个数）个小表的 hashtable，做连接 操作的时候，只需要将小表的一个 hashtable 放入内存即可，然后将大表的对应的 split 拿出来进行连接，所以其内存限制为小表中最大的那个 hashtable 的大小
4. Sort Merge Bucket Map Join

对于 bucket map join 中的两个表，如果每个桶内分区字段也是有序的，则还可以进行 sort merge bucket map join。

这样一来当两边 bucket 要做局部 join 的时候，只需要用类似 merge sort 算法中的 merge 操作一样把两个 bucket 顺序遍历一遍即可完成，这样甚至都不用把一个 bucket 完整的加载成 hashtable，而且可以做全连接操作。

操作时需要设置Hive的部分属性，从而使其进行Sort Merge Bucket Map Join

# 分布式一致性

**1. 为什么要做数据冗余？保障数据一致性方法？**

防止数据丢失、出错，进行容灾、读写加速。

* 2PC
* 3PC
* paxos
* raft
* zookeeper

**2. 什么是脑裂？zookeeper怎么解决脑裂？**

脑裂：一个整体内的节点互相联系着，一切运转正常。当网络问题导致心跳检测误判原master不在了，主备切换产生新的master。而后续被判定“死”的master由于网络恢复复活，此时网络出现两个主master，出现争抢共享资源造成系统混乱、数据损坏的现象，即：脑裂。

ZooKeeper默认采⽤了Quorums(法定⼈数)的⽅式: 只有获得超过半数节点的投票, 才能选举出leader，这种⽅式可以确保要么选出唯⼀的leader，要么选举失败。

**3. Zookeeper锁机制，Zookeeper读写锁实现？**

客户端尝试创建一个 znode 节点，比如 /lock 。那么第一个客户端就创建成功了，相当于拿到了锁；而其它的 客户端会创建失败（znode 已存在），获取锁失败。znode 应该被创建成 ephemeral 的。这是znode的一个特性，它保证如果创建 znode 的那个客户端崩溃了，那么相应的 znode 会被自动删除。这保证了锁一定会被释放。这个特性避免了设置锁的过期时间。

如果是写锁：

获取/lock 下的子节点

按照顺序号排序

检查此写锁之前是否还有其他锁，若有先注册对该写锁前一个锁的监听，然后阻塞该写锁获取，若监听到该写锁前一个锁已释放，则该写锁打开阻塞。

如果是读锁：

获取/lock 下的子节点

按照顺序号排序

检查此读锁之前是否有写锁，若有先注册对该读锁的前一个写锁的监听，然后阻塞该读锁的获取。若监听到该读锁前一个写锁已释放，则该读锁打开阻塞

**4. 什么是“羊群效应”，在Zookeeper中怎么优化？**

持有锁的客户端访问共享资源完成后，将 znode 删掉，这样其它客户端接下来就能来获取锁了。如上所述的基于ZooKeeper 的分布式锁的实现，并不是最优的,它会引发 “herd effect”（羊群效应），降低获取锁的性能。可以设置锁节点为顺序临时节点，后面的节点 watch 前面的节点，当前面的节点删除时唤醒后面的节点从而避免羊群效应。

**5. Zookeeper为什么是CP架构？AP结构用在哪里？**

CAP理论：在一个分布式系统中， Consistency（一致性）、 Availability（可用性）、Partition tolerance（分区容错性），三者不可得兼。

强调的是一致性（C），在事务消息写入时，为尽最大努力保证数据视图一致性

ZooKeeper为了保证在集群脑裂分区（P）情况下数据一致性（C），其放弃了可用性（A），所以在集群部分节点或者网络发生故障，master离线的时候，集群将进行崩溃恢复选注，在选主期间，整个集群对外是不可用的，一直持续到选主完成，主从数据同步成功为止

明显AP结构选择了高可用和分区容错性，此时，那个失去联系的节点依然可以向系统提供服务，不过它的数据就不能保证是同步的了（失去了C属性）。Eureka就是一个AP架构的例子，当Eureka客户端心跳消失的时候，那Eureka服务端就会启动自我保护机制，不会剔除该EurekaClient客户端的服务，依然可以提供需求；

**6. 什么是BASE理论，什么是最终一致性？**

是在实践过程中基于CAP理论演化而来的，基本可用，软状态，最终一致性

基本可用：是指分布式系统在出现不可预知故障的时候，允许损失部分可用性。即保证核心可用

软状态：指系统中的数据存在中间状态，并认为该中间状态的存在不会影响系统的整体可用性，即允许系统在不同节点的数据副本之间进行数据同步的过程存在延迟

最终一致性：所有数据副本经过一定时间后，最终能够达到一致的状态

核心思想：既然无法做到强一致性，但每个应用都可以跟据自身业务特点，采用适当的方式来使系统达到最终一致性，也就是牺牲数据的一致性来满足系统的高可用性，系统中一部分数据不可用或不一致时，仍需要保持系统整体基本可用。

**7. Zookeeper的ZAB协议和Paxos有什么区别？**

Paxos算法中，一个新的选举产生的主进程会进行两个阶段的工作

1、读阶段，新的主进程会通过和所有其他进程进行通信的方式来搜集上一个主进程提出的提案，并将它们提交。

2、写阶段，当前主进程开始提出它自己的提案。

ZAB在Paxos基础上额外添加一个同步阶段。同步阶段之前，ZAB协议存在一个和Paxos读阶段类似的发现（Discovery）阶段

同步阶段中，新的Leader会确保存在过半的Follower已经提交了之前Leader周期中的所有事务Proposal

发现阶段的存在，确保所有进程都已经完成对之前所有事物Proposal的提交

ZAB协议主要用于构建一个高可用的分布式数据主备系统，例如ZooKeeper

Paxos算法则是用于构建一个分布式的一致性状态机系统

**8. ZAB协议和Raft有什么区别？**







**9. Zookeeper的选举算法：**

Zookeeper 在配置文件中并没有指定 Master 和 Slave。但是，Zookeeper 工作时， 是有一个节点为 Leader，其他则为 Follower，而这个 Leader 是通过内部的选举机制临时产生的。

每个 Server 首先都提议自己是 Leader，并为自己投票，然后将投票结果与其他 Server 的选票进行对比，权重大的胜出，使用权重较大的选票更新自身的投票箱，我们介绍下服务器启动时期的 Leader 选举。

1）每一个 Server 都会发出一个投票

在集群初次启动时，每个 Server 都会推荐自己为 Leader，然后各自将这个投票发给集群中其他 Server。

2）接收来自各个 Server 的投票

每个 Server 在接收到其他 Server 的投票后，首先会判断该票的有效性，包括检查是否本轮投票，是否来自 Looking 状态的 Server。（Looking 状态表示当前集群正处于选举状态）

3）处理投票

针对每一个投票，Server 都会将别人的投票和自己的投票进行 PK，计算出 Zxid 最大的 Server，并将该 Server 设置成下一次投票推荐的 Server。

4）统计投票

每次投票结束之后，都会统计所有投票，获取投票最多的 Server 将成为获胜者，如果获胜者的票数超过集群个数的一半，则该 Server 将为推选为 Leader。否则继续投票，直至 Leader 被选举出来。

5）改变服务器状态

一旦 Leader 确定后，Leader 会通知其他 Follower 集群已经成为 Uptodate 状态，Follower 在收到 Uptodate 消息后，接收 Client 的请求并开始对外提供服务。

**10. 什么是Paxos协议，其与拜占庭将军协议应用场景存在什么区别？**

在常见的分布式系统中，总会发生诸如机器宕机或网络异常（包括消息的延迟，丢失，重复，乱序，网络分区）等情况。Paxos算法需要解决的问题就是如何在一个可能发生上述异常的分布式系统中，快速且正确的在集群内布对某个数据的值达成一致，并且保证不论发生以上任何异常，都不会破坏整个系统的一致性。

proposer、Acceptor、Learner

是否存在拜占庭错误。人为恶意篡改包括“精神分裂式投票”，中继篡改上一个办事处的投票信息。对应到分布式系统面临的问题就是：消息被伪造、系统安全攻击等等。发生的人为恶意篡改的过程就可以称之为系统发生了拜占庭错误

**11. raft选举过程，prevote**

节点启动时都是follower状态；在一段时间内如果没有收到来自leader的心跳，从follower切换到candidate，发起选举；如果收到majority的造成票（含自己的一票）则切换到leader状态；如果发现其他节点比自己更新，则主动切换到follower。

增加节点本地的 current term ，切换到candidate状态

投自己一票

并行给其他节点发送 RequestVote RPCs

等待其他节点的回复

为了防止网络分区后，被隔离的几点始终自增任期，再log落后的情况下成为leader，需要进行预选举。在预选举，不增加任期，如果预选举获胜，则增加任期参加选举

**12. raft怎么解决脑裂**

网络分区导致老的leader在少数区，那么，由于其发现自己无法联系大多数节点，因此不会进行数据写。多数区重新选举出新leader

老leader在多数区，少数区没法选举，问题不大

**13. raft怎么保证数据一致性，raft提案过程描述**

整个系统只有一个领导者，所有的外部请求只会由领导者处理，即使是跟随者收到请求，也会转发给领导者处理，这是数据一致性的保障之一。

整个系统对外可用需要达到多数节点存活的情况下才行，也就是说领导者是由多数节点达成一致选举出来的，这是数据一致性的又一个保障。

Raft的数据持久化分为快照和事务日志。跟随者会始终保持跟领导者的数据同步，这是数据一致性的又一个保障。

**14. 什么是分布式事务？分布式事务有哪些解决方案？**

一次大的操作由不同的小操作组成，这些小的操作分布在不同的服务器上，且属于不同的应用，分布式事务需要保证这些小操作要么全部成功，要么全部失败。本质上来说，分布式事务就是为了保证不同数据库的数据一致性。

2PC，3PC，TCC

**15. 两阶段提交协议和三阶段提交协议的区别，三阶段提交协议解决了什么问题？还存在什么问题？**

与二阶段相比，三阶段：

1. 引入超时机制：同时在协调者和参与者中都引入超时机制。

2. 在第一阶段和第二阶段中插入一个准备阶段：保证了在最后提交阶段之前各参与节点的状态是一致的，即三阶段提交就有 CanCommit（无锁状态）、PreCommit（无锁状态）、DoCommit 三个阶段。

存在问题：

1. 三阶段的超级机制，解决了阻塞问题。

2. CanCommit的预先铺垫 过渡到 PreCommit 的预备阶段，相当于让我们有理由相信 DoCommit 成功提交的几率很大，但是由于网络原因导致的数据不一致问题依然存在。

**16. 什么是TCC，TCC存在什么问题？**

TCC方案分为Try Confirm Cancel三个阶段，属于补偿性分布式事务。

Try：尝试待执行的业务

这个过程并未执行业务，只是完成所有业务的一致性检查，并预留好执行所需的全部资源

Confirm：执行业务

这个过程真正开始执行业务，由于Try阶段已经完成了一致性检查，因此本过程直接执行，而不做任何检查。并且在执行的过程中，会使用到Try阶段预留的业务资源。

Cancel：取消执行的业务

若业务执行失败，则进入Cancel阶段，它会释放所有占用的业务资源，并回滚Confirm阶段执行的操作。

TCC需要编写大量的补偿事务的代码，比较冗余

**第二小组：Spark生态圈面试题**

1.Spark中的数据本地性有哪三种：

 PROCESS\_LOCAL是指读取缓存在本地节点的数据

NODE\_LOCAL是指读取本地节点硬盘数据

 ANY是指读取非本地节点数据

### 2.[过滤器有什么用途：](https://link.zhihu.com/?target=https://gitee.com/souyunku/DevBooks/blob/master/docs/%E5%A4%A7%E6%95%B0%E6%8D%AE/%E5%A4%A7%E6%95%B0%E6%8D%AE%E6%9C%80%E6%96%B02021%E5%B9%B4%E9%9D%A2%E8%AF%95%E9%A2%98%E9%99%84%E7%AD%94%E6%A1%88%E8%A7%A3%E6%9E%90%EF%BC%8C%E5%A4%A7%E6%B1%87%E6%80%BB.md%231%E8%BF%87%E6%BB%A4%E5%99%A8%E6%9C%89%E4%BB%80%E4%B9%88%E7%94%A8%E9%80%94%EF%BC%9A)

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

reduce之后数据的输出量有多大（结合具体场景，比如pi）

Sca阶段的增强日志（1.5T—2T）

过滤性质的mr程序，输出比输入少

解析性质的mr程序，输出比输入多（找共同朋友）

### [3、RDD缓存](https://gitee.com/souyunku/DevBooks/blob/master/docs/%E5%A4%A7%E6%95%B0%E6%8D%AE/%E5%A4%A7%E6%95%B0%E6%8D%AE%E6%9C%80%E6%96%B02021%E5%B9%B4%E9%9D%A2%E8%AF%95%E9%A2%98%E9%99%84%E7%AD%94%E6%A1%88%E8%A7%A3%E6%9E%90%EF%BC%8C%E5%A4%A7%E6%B1%87%E6%80%BB.md#2rdd%E7%BC%93%E5%AD%98)类型

rdd.cache

rdd.persist

### 4.[什么是spark](https://link.zhihu.com/?target=https://gitee.com/souyunku/DevBooks/blob/master/docs/%E5%A4%A7%E6%95%B0%E6%8D%AE/%E5%A4%A7%E6%95%B0%E6%8D%AE%E6%9C%80%E6%96%B02021%E5%B9%B4%E9%9D%A2%E8%AF%95%E9%A2%98%E9%99%84%E7%AD%94%E6%A1%88%E8%A7%A3%E6%9E%90%EF%BC%8C%E5%A4%A7%E6%B1%87%E6%80%BB.md%233%E4%BB%80%E4%B9%88%E6%98%AFspark)

基于内存计算发数据分析引擎，提高在大数据环境下数处理的实时性，spark仅涉及数据计算

### 5.[RDD 是什么](https://link.zhihu.com/?target=https://gitee.com/souyunku/DevBooks/blob/master/docs/%E5%A4%A7%E6%95%B0%E6%8D%AE/%E5%A4%A7%E6%95%B0%E6%8D%AE%E6%9C%80%E6%96%B02021%E5%B9%B4%E9%9D%A2%E8%AF%95%E9%A2%98%E9%99%84%E7%AD%94%E6%A1%88%E8%A7%A3%E6%9E%90%EF%BC%8C%E5%A4%A7%E6%B1%87%E6%80%BB.md%238rdd-%E6%98%AF%E4%BB%80%E4%B9%88)

弹性分布式数据集，是spark中最基本的数据抽象，可以存于内存中或者磁盘中，分布式存储可用于分布式计算

6.什么是窄依赖

窄依赖指的是每一个父RDD的partition最多被子RDD的一个artition使用一对一

1. 什么是宽依赖

是多个子RDD的partition会依赖于同一个父RDD的partition多对一

# 简述RDD的血缘(Lineage)

RDD血缘（RDD Lineage），也可以叫：RDD依赖关系图。当我们计算一个RDD时，会依赖一个或多个父RDD的数据，而这些父RDD又会依赖它自身的父RDD，这样RDD之间的依赖关系就形成了一个有向无环图（也叫DAG图），这些依赖关系被记录在一个图中，这就是RDD的血缘（也叫RDD Lineage）。

1. RDD之间的依赖关系是如何产生的呢？

这些依赖关系的产生是由于对RDD进行的多个转换操作。由于RDD是懒评价的，一般情况下在没有action操作时，不会执行转换操作。也就是说在执行计算任务之前，执行的操作基本上都是转换操作，而我们知道每个转换操作的输出是一个新的RDD，这样通过多个转换操作就会得到多个新的RDD，而新的RDD之间就形成了RDD 的血缘关系（RDD Lineage）。

10.1.什么时候该使用持久化（缓存）

在[Spark](https://so.csdn.net/so/search?q=Spark&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_43161811/article/details/_blank)应用开发中，有时我们希望能多次使用同一个RDD。如果简单地对RDD进行进行调用，每执行一次action操作，Spark每次都会重算RDD以及它的所有依赖，这在迭代算法中消耗格外的大。

11.简述RDD CheckPoint 检查点

所谓的检查点其实就是通过将 RDD 中间结果写入磁盘由于血缘依赖过长会造成容错成本过高，这样就不如在中间阶段做检查点容错，如果检查点之后有节点出现问题，可以从检查点开始重做血缘，减少了开销。对 RDD 进行 checkpoint 操作并不会马上被执行，必须执行 Action 操作才能触发。

12.cache & persist & checkpoint 的特点和区别

特点

cache：

将数据临时存储在内存中进行数据重用

会在血缘关系中添加新的依赖，一旦出现问题，可以重头读取数据

persist：

将数据临时存储在磁盘文件中进行数据重用

因为涉及到磁盘IO，性能较低，但是数据安全

如果作业执行完毕，临时保存的数据文件会丢失

会在血缘关系中添加新的依赖，一旦出现问题，可以重头读取数据

checkpoint：

将数据长久的保存在磁盘文件中进行数据重用

因为涉及到磁盘IO，性能较低，但是数据安全

为了保证数据安全，所以一般情况下，会独立执行作业，

即调用检查点的rdd以前的流程都会重新执行一遍，所以效率比较低

为了能够提高效率，一般情况下，是需要和cache联合使用的

执行过程中，会切断血缘关系，重新建立新的血缘关系，

相当于将一个作业的数据源，由原文件切到检查点落盘的文件

下次重读数据时直接从checkpoint文件中读取

checkpoint等同于改变数据源

区别

cache/persist缓存只是将数据保存起来，不切断血缘依赖，只是加一条缓存的依赖关系。而checkpoint检查点会切断血缘依赖，checkpoint下游的RDD直接从checkpoint落盘的文件中读取数据，checkpoint等同于改变了数据源。（如下图）

cache/persist缓存的数据通常存储在磁盘、内存等地方，可靠性低。Checkpoint的数据通常存储在HDFS等容错性强、高可用的文件系统中，可靠性高。

建议对checkpoint的RDD使用cache缓存，这样checkpoint的job只需从cache缓存中读取数据即可，否则需要从头计算一次RDD。

13.简述Spark的几种持久化级别：

1.MEMORY\_ONLY

使用未序列化的Java对象格式，将数据保存在内存中。如果内存不够存放所有的数据，Spark会自动利用最近最少用(LRU)的缓存策略把最老的分区从内存中移除。下一次要用到已经被移除的分区数据时，这些分区需要从源头处重新计算一遍。这是默认的持久化策略，使用cache()方法时，实际就是使用的这种持久化策略。

2.MEMORY\_AND\_DISK

使用未序列化的Java对象格式，优先尝试将数据保存在内存中。如果内存不够存放所有的数据，Spark会自动利用最近最少用(LRU)的缓存策略把最老的分区从内存中移除，并将数据溢写到磁盘文件中，下次对这个RDD执行算子时，持久化在磁盘文件中的数据会被读取出来使用。

3.MEMORY\_ONLY\_SER

基本含义同MEMORY\_ONLY。唯一的区别是，会将RDD中的数据进行序列化，RDD的每个partition会被序列化成一个字节数组。这种方式会使缓存过程变慢，因为序列化对象也会消耗一些代价，不过这更加节省内存，从而可以避免持久化的数据占用过多内存导致频繁GC，同时也可以显著减少JVM的GC时间。

4.MEMORY\_AND\_DISK\_SER

基本含义同MEMORY\_AND\_DISK。唯一的区别是，会将RDD中的数据进行序列化，RDD的每个partition会被序列化成一个字节数组。这种方式会使缓存过程变慢，因为序列化对象也会消耗一些代价，不过这更加节省内存，从而可以避免持久化的数据占用过多内存导致频繁GC，同时也可以显著减少JVM的GC时间。

5.DISK\_ONLY

使用未序列化的Java对象格式，将数据全部写入磁盘文件中。

1. Spark SQL简介：是Spark用来处理结构化数据的一个模块，它提供了一个编程抽象叫做DataFrame并且作为分布式SQL查询引擎的作用。
2. Spark SQL的特点：1.容易整合（集成）；2.统一的数据访问方式；3.兼容Hive；44.标准的数据连接
3. Spark SQL允许用户提交SQL文本，支持哪些手段编写sql文本：1. spark 代码2. spark-sql的shell3. thriftserver
4. 什么是DataFrame：DataFrame是组织成命名列的数据集。它在概念上等同于关系数据库中的表，但在底层具有更丰富的优化。DataFrames可以从各种来源构建，例如：结构化数据文件(JSON)、外部数据库或现有RDDs。DataFrame API支持的语言有Scala，Java，Python和R。
5. 什么是DataSet：Dataset是数据的分布式集合。Dataset是在Spark 1.6中添加的一个新接口，是DataFrame之上更高一级的抽象。它提供了RDD的优点（强类型化）以及Spark SQL优化后的执行引擎的优点。一个Dataset 可以从JVM对象构造，然后使用函数转换（map， flatMap，filter等）去操作。 Dataset API 支持Scala和Java。 Python不支持Dataset API。

### Spark-SQL连接Hive 的五种方法：1）内嵌的 HIVE。2）外部的 HIVE。3）运行 Spark beeline。4）运行Spark-SQL CLI。5）代码操作Hive

1. 简述 GraphX应用背景： Spark GraphX是一个分布式图处理框架，它是基于Spark平台提供对图计算和图挖掘简洁易用的而丰富的接口，极大的方便了对分布式图处理的需求。社交网络中人与人之间有很多关系链，例如Twitter、Facebook、微博和微信等，这些都是大数据产生的地方都需要图计算，现在的图处理基本都是分布式的图处理，而并非单机处理。Spark GraphX由于底层是基于Spark来处理的，所以天然就是一个分布式的图处理系统。图的分布式或者并行处理其实是把图拆分成很多的子图，然后分别对这些子图进行计算，计算的时候可以分别迭代进行分阶段的计算，即对图进行并行计算。
2. 什么是边分割（Edge-Cut）：每个顶点都存储一次，但有的边会被打断分到两台机器上。这样做的好处是节省存储空间；坏处是对图进行基于边的计算时，对于一条两个顶点被分到不同机器上的边来说，要跨机器通信传输数据，内网通信流量大。
3. 什么是点分割（Vertex-Cut）：每条边只存储一次，都只会出现在一台机器上。邻居多的点会被复制到多台机器上，增加了存储开销，同时会引发数据同步问题。好处是可以大幅减少内网通信量。
4. GraphX存储模式：Graphx借鉴PowerGraph，使用的是Vertex-Cut(点分割)方式存储图，用三个RDD存储图数据信息：lVertexTable(id, data)：id为Vertex id，data为Edge data
5. 图计算模式： 目前基于图的并行计算框架已经有很多，比如来自Google的Pregel、来自Apache开源的图计算框架Giraph/HAMA以及最为著名的GraphLab，其中Pregel、HAMA和Giraph都是非常类似的，都是基于BSP（Bulk Synchronous Parallell）模式。

25.图数据库neo4j和spark下面的graphx有什么区别？

neo4j是native graph database，也就是有自己的数据库存储。它的长处在于支持交互式查询，属于oltp系统，很多人说不支持分片存储使其无法应付海量数据，本人觉得恰恰相反，可以说neo4j的存储方式是教科书式的以空间换时间，每台服务器配备ssd磁盘阵列虽然贵，但是可以大幅减少分片存储的带宽占用和通信时间开销，保证oltp的效率。 neo4j很容易上手，特有的cypher查询语言以画草图的方式查询和建模数据，很直观。适当构建查询计划的情况下，neo4j的查询效率很高，能够迅速从整网中找出符合特定模式的子网，供随后分析之用。 此外，neo4j实现了tinkerpop接口，tinkerpop是刚刚毕业的一个阿帕奇项目，有望建立图数据库的一套标准用户接口。同样实现tinkerpop的还有titan，orient等主流图数据库。再来看graphX ： graphX是spark的系统组件，存储是基于spark rdd的，有节点和边两种rdd。熟悉spark的朋友对rdd该不会陌生，spark通过缓存rdd的操作节省了大量计算和io开支，因此spark特别适合对海量数据进行运算，此理同样适用于graphX。因此，graphX自设计之初就是奔着图计算的目标去的，属于olap系统，而非oltp系统。 graphX有丰富的函数库，能完成很多经典图算法，如pagerank、三角计数、社群发现、最短路径计算等等。此外，图存储和计算的方式不禁让人想到神经网络算法，如果将隐层用节点rdd表示，隐层之间的边用边rdd表示，运用graphX的计算优势搭建起一套多层神经网络的想法很美妙，这应该就是MLlab相应算法模块的工作原理。 因此跟graphx相关的概念集中在图计算，而非图存储和查询领域。所以经常浏览db-engines的朋友们不难发现，图数据库列表里就没有graphx这一项。在比较图存储和图查询性能时，比较集合多是neo4j、orientdb、titan、arangodb等图数据库系统。而比较图计算时，比较集合多是graphlab、giraph、graphX。 简言之，图数据库系统和图计算系统不是一回事：前者是为了存储完整数据，并根据需求从中查询数据子集供分析展示之用；后者的任务是拿到一个图结构的数据集，从中计算一些有用的东西。

1. 图（GraphX）的基本概念：图是由顶点集合(vertex)及顶点间的关系集合（边edge）组成的一种数据结构。这里的图并非指代数中的图。图可以对事物以及事物之间的关系建模，图可以用来表示自然发生的连接数据，如：社交网络、互联网web页面。常用的应用有：在地图应用中找到最短路径、基于与他人的相似度图，推荐产品、服务、人际关系或媒体。
2. 有向图和无向图在有向图中，一条边的两个顶点一般扮演者不同的角色，比如父子关系、页面A连接向页面B；在一个无向图中，边没有方向，即关系都是对等的，比如qq中的好友。GraphX中有一个重要概念，所有的边都有一个方向，那么图就是有向图，如果忽略边的方向，就是无向图。
3. 有环图和无环图：有环图是包含循环的，一系列顶点连接成一个环。无环图没有环。在有环图中，如果不关心终止条件，算法可能永远在环上执行，无法退出。
4. 超步：图进行迭代计算时，每一轮的迭代叫做一个超步
5. 图数据分析：Google Pregel、Spark GraphX、GraphLab等图计算软件。传统的数据分析方法侧重于事物本身，即实体，例如银行交易、资产注册等等。而图数据不仅关注事物，还关注事物之间的联系。例如，如果在通话记录中发现张三曾打电话给李四，就可以将张三和李四关联起来，这种关联关系提供了与两者相关的有价值的信息，这样的信息是不可能仅从两者单纯的个体数据中获取的。
6. 图数据可视化：OLTP风格的图数据库或者OLAP风格的图数据分析系统（或称为图计算软件），都可以应用图数据库可视化技术。需要注意的是，图可视化与关系数据可视化之间有很大的差异，关系数据可视化的目标是对数据取得直观的了解，而图数据可视化的目标在于对数据或算法进行调试。

31.GraphX实现架构大致分为哪3个层：

• 实现层：该层定义了 GraphX由最基本的数据结构，即顶点、边和边点三元组；实现了GraphX不同的顶点切分策略以及数据存储方式以及图计算过程使用的数据存储结构，如路由表、重复顶点视图等。

• 操作层：主要包括抽象类GraphX及其实现类GraphImpl,在这两个类中定义了构建图操作、转换操作、结构操作、聚合操作和缓存操作等。另外，在GraphOps类中也实现了图基本属性操作和连接操作等。

• 算法层：GraphX根据实现层和操作层实现了常用的算法，如PageRank、三角关系统计、最短路径等。

32.Spark有哪些适用场景：

Spark是基于内存的迭代计算框架，适用于需要多次操作特定数据集的应用场合。需要反复操作的次数越多，所需读取的数据量越大，受益越大，数据量小但是计算密集度较大的场合，受益就相对较小。

33.Spark有哪些不适用场景：

由于RDD的特性，Spark不适用那种异步细粒度更新状态的应用，例如web服务的存储或者是增量的web爬虫和索引。就是对于那种增量修改的应用模型不适合。数据量不是特别大，但是要求近实时统计分析需求· 内存hold不住的场景，在内存不足的情况下，Spark会下放到磁盘，会降低应有的性能 ；高实时性要求的流式计算业务，例如实时性要求毫秒级；由于RDD设计上的只读特点，所以Spark对于待分析数据频繁变动的情景很难做（并不是不可以），比如题主例子里的搜索，假设你的数据集在频繁变化（不停增删改），而且又需要结果具有很强的一致性（不一致时间窗口很小），那么就不合适了。

### **34.描述spark计算模型：**

RDD可以看做是对各种数据计算模型的统一抽象，Spark的计算过程主要是RDD的迭代计算过程。RDD的迭代计算过程非常类似于管道。分区数量取决于partition数量的设定，每个分区的数据只会在一个Task中计算。所有分区可以在多个机器节点的Executor上并行执行。

35.叙述spark 运行流程

· 构建Spark Application的运行环境，启动SparkContext

· SparkContext向资源管理器（可以是Standalone，Mesos，Yarn）申请运行Executor资源，并启动StandaloneExecutorbackend，

· Executor向SparkContext申请Task

· SparkContext将应用程序分发给Executor

· SparkContext构建成DAG图，将DAG图分解成Stage、将Taskset发送给Task Scheduler，最后由Task Scheduler将Task发送给Executor运行

· Task在Executor上运行，运行完释放所有资源

36.叙述spark 运行特点：

每个Application获取专属的executor进程，该进程在Application期间一直驻留，并以多线程方式运行Task。这种Application隔离机制是有优势的，无论是从调度角度看（每个Driver调度他自己的任务），还是从运行角度看（来自不同Application的Task运行在不同JVM中），当然这样意味着Spark Application不能跨应用程序共享数据，除非将数据写入外部存储系统

Spark与资源管理器无关，只要能够获取executor进程，并能保持相互通信就可以了

提交SparkContext的Client应该靠近Worker节点（运行Executor的节点），最好是在同一个Rack里，因为Spark Application运行过程中SparkContext和Executor之间有大量的信息交换Task采用了数据本地性和推测执行的优化机制

37.MLlib提供了哪些工具：

1.机器学习算法：常规机器学习算法包括分类、回归、聚类和协同过滤。

2.特征工程：特征提取、特征转换、特征选择以及降维。

3.管道：构造、评估和调整的管道的工具。

4.存储：保存和加载算法、模型及管道

5.实用工具：线性代数，统计，数据处理等。

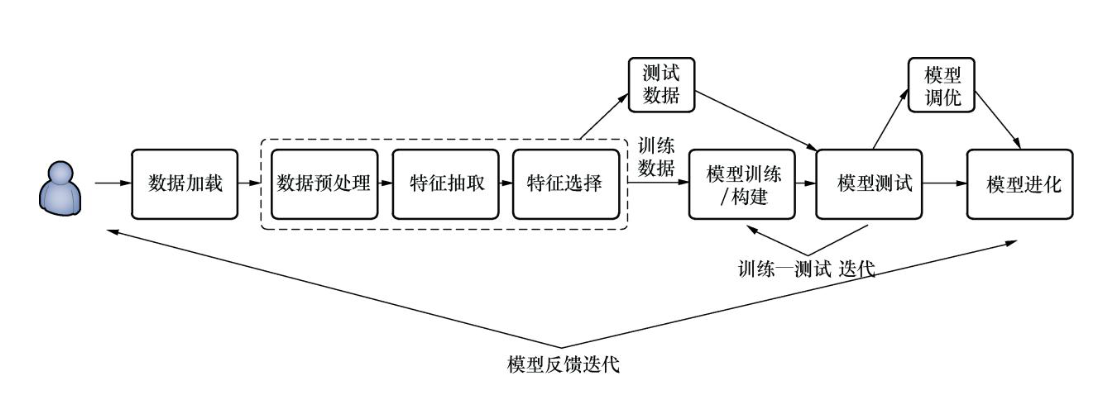
38.为什么MLlib转向数据框接口？

1.数据框可以提供比RDD更容易掌握使用的接口。数据框的主要优点包括Spark数据源来源、结构化查询语言的数据框查询、各编程语言之间统一的接口。

2.基于数据框的MLlib接口为多种机器学习算法与编程语言提供统一的接口。

3.数据框有助于实现机器学习管道，特别是特征转换。管道指南中可查看详细信息。

## 39.请画出典型的机器学习工作流



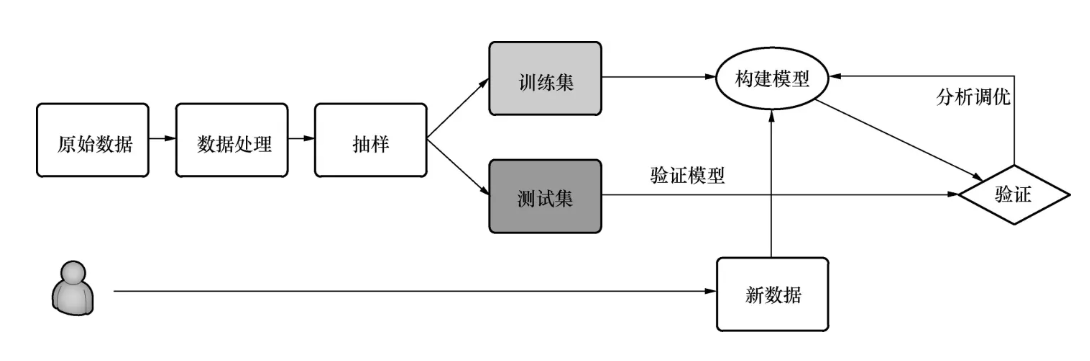
1. 根据学习系统学习反馈的本质，机器学习任务通常被分为哪几类

3类，即监督学习、无监督学习以及增强学习

## 41.什么是监督学习

监督学习的目标是学习将输入映射到与现实世界相一致的输出的一般规则。例如，垃圾邮件过滤数据集通常包含垃圾邮件以及非垃圾邮件。因此，能够知道训练集中的数据是垃圾邮件还是正常邮件。我们有机会利用这些信息来训练模型，以便对新来的邮件进行分类。如图 6-3所示，该图为监督学习的示意图。算法找到所需的模式后，可以使用这些模式对未标记的测试数据进行预测。这是最常见的机器学习任务类型，MLlib 也不例外，其中大部分算法都是监督学习，如朴素贝叶斯、逻辑回归、随机森林等。

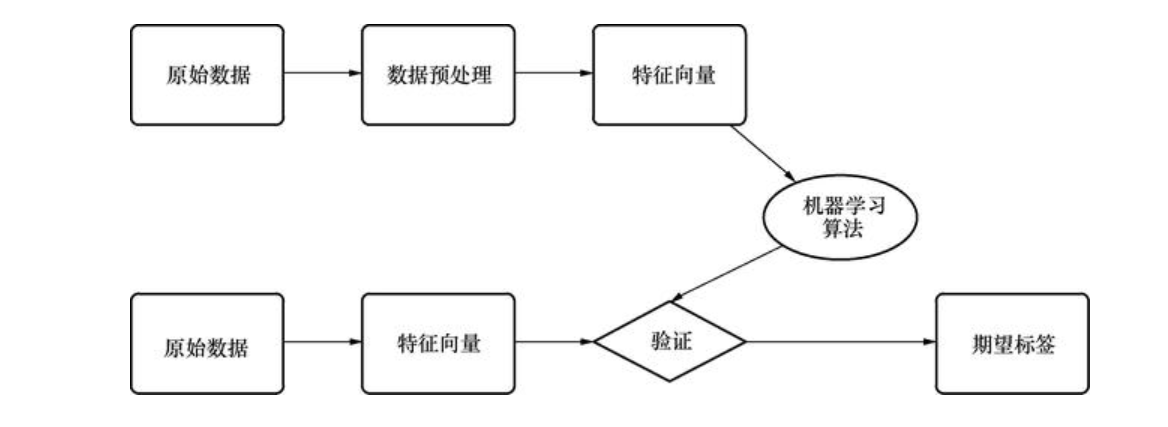
1. 画出监督学习的数据处理流程图



## 什么是无监督学习

在无监督学习中，数据没有相关的标签，也就是说无法区分训练集与测试集。因此，我们需要用算法上加上标签。因此，标签必须从数据集中推断出来，这意味着无监督学习算法的目标是通过描述结构，以某种结构化的方式对数据进行预处理。

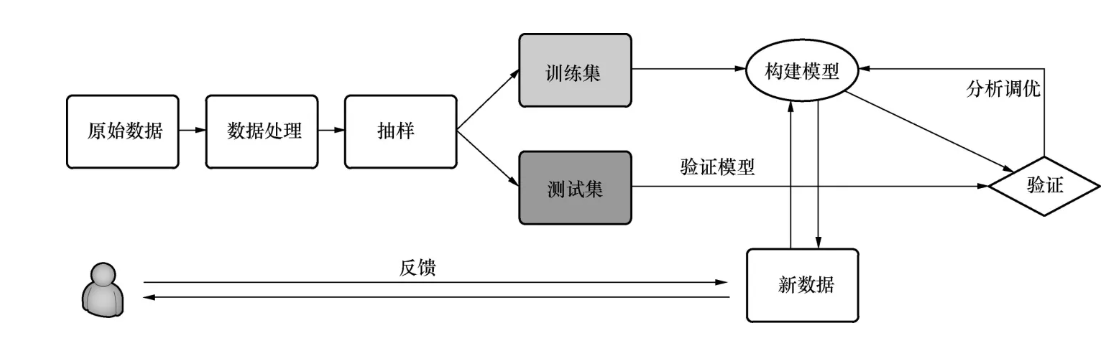
1. .画出无监督学习的数据处理流程图



## 44.什么是增强学习

增强学习是一种模型本身从一系列行为中学习的技术。数据集或样本的复杂性对于需要算法成功学习目标函数的增强学习非常重要。此外，为了达到最终目标，每条数据都需要做到一点，即在保证与外部环境相互作用的同时，应确保奖励函数的最大化。

1. 请画出**增强学习流程图**



## 45.试比较park MLlib与Spark ML异同

在Spark MLlib模块中，可以看到它的源码主要分为两个包：spark.ml与spark.mllib，我们将前者称为Spark ML API，后者称为Spark MLlib API，有些算法在两个包中都可以找到，如协同过滤，有些算法只有MLlib有，如SVD。除此以外，它们还有一些区别。

一言以蔽之，MLlib与ML之间最大的区别在于，ML基于DataFrame，而MLlib API基于RDD，这与GraphX和GraphFrame之间的关系类似。在Spark 2.0后，基于RDD的API，也就是MLlib API，就已经进入了维护状态，而Spark MLlib首要的API为ML API。虽然如此，Spark MLlib仍然会以修复Bug的方式支持MLlib API，但不会增加新特性了。在Spark 2.x的版本中，ML API会逐渐变得与MLlib API一样，在完成这个过程后，MLlib API会被弃用。在Spark 3.0中，MLlib API会被彻底舍弃。

## 46.什么是数据预处理，为什么要进行数据预处理

在机器学习实践中，数据科学家拿到的数据通常是不尽如人意的，例如存在大量的缺失值、特征的值是不同的量纲、有一些无关的特征、特征的值需要再次处理等情况，这样的数据无法直接训练，因此我们需要对这些数据进行预处理。预处理在机器学习中是非常重要的步骤，如果没有按照正确的方法对数据进行预处理，往往会得到错误的训练结果。

1. 请介绍集中介绍几种常见的预处理方法

## 数据标准化 缺失值处理 特征抽取 特征选择

## 介绍机器学习中常见的分类算法应用

决策树：决策树是一种机器学习的方法，本质上是通过一种树形结构对样本进行分类，每个非叶子结点是一次判断，每个叶子结点代表了分类结果。决策树是一种典型的监督学习，需要一定量的样本来学习的一个树形结构，常见的决策树构造树算法有C4.5与ID3。

随机森林：在[机器学习](https://baike.baidu.com/item/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%9A%8F%E6%9C%BA%E6%A3%AE%E6%9E%97/_blank)中，随机森林是一个包含多个[决策树](https://baike.baidu.com/item/%E5%86%B3%E7%AD%96%E6%A0%91/10377049?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%9A%8F%E6%9C%BA%E6%A3%AE%E6%9E%97/_blank)的[分类器](https://baike.baidu.com/item/%E5%88%86%E7%B1%BB%E5%99%A8?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%9A%8F%E6%9C%BA%E6%A3%AE%E6%9E%97/_blank)， 并且其输出的类别是由个别树输出的类别的[众数](https://baike.baidu.com/item/%E4%BC%97%E6%95%B0/44796?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%9A%8F%E6%9C%BA%E6%A3%AE%E6%9E%97/_blank)而定。 Leo Breiman和Adele Cutler发展出推论出随机森林的算法。

49.简述Spark MLlib架构

从架构图可以看出MLlib主要包含三个部分：底层基础：包括Spark的运行库、矩阵库和向量库； 算法库：包含广义线性模型、推荐系统、聚类、决策树和评估的算法；实用程序：包括测试数据的生成、外部数据的读入等功能。

50.简述回归算法：属于监督式学习，每个个体都有一个与之相关联的实数标签，并且我们希望在给出用于表示这些实体的数值特征后，所预测出的标签值可以尽可能接近实际值。MLlib 目前支持回归算法有：线性回归、岭回归、Lasso和决策树。

51.简述聚类算法：聚类算法属于非监督式学习，通常被用于探索性的分析，是根据“物以类聚”的原理，将本身没有类别的样本聚集成不同的组，这样的一组数据对象的集合叫做簇，并且对每一个这样的簇进行描述的过程。它的目的是使得属于同一簇的样本之间应该彼此相似，而不同簇的样本应该足够不相似，常见的典型应用场景有客户细分、客户研究、市场细分、价值评估。MLlib 目前支持广泛使用的KMmeans聚类算法。

52.什么是协同过滤

协同过滤常被应用于推荐系统，这些技术旨在补充用户-商品关联矩阵中所缺失的部分。MLlib当前支持基于模型的协同过滤，其中用户和商品通过一小组隐语义因子进行表达，并且这些因子也用于预测缺失的元素。

## 什么是Spark Streaming？

数据流是连续到达的无穷序列。流处理将不断流动的输入数据分成独立的单元进行处理。流处理是对流数据的低延迟处理和分析。Spark Streaming是Spark API核心的扩展，可实现实时数据的快速扩展，高吞吐量，高容错处理。Spark Streaming适用于大量数据的快速处理。

54.**Spark Streaming**实时处理用例包括哪些场景：

* 网站监控，网络监控
* 欺诈识别
* 网页点击
* 广告
* 物联网传感器

## 55.Spark Straming如何工作

Spark Streaming将数据流每X秒分作一个集合，称为Dstreams，它在内部是一系列RDD。您的Spark应用程序使用Spark API处理RDD，并且批量返回RDD操作的结果。

1. 简谈Spark Streaming的实时计算整合

基于Spark通用计算平台，可以很好地扩展各种计算类型的应用，尤其是Spark提供了内建的计算库支持，像Spark Streaming、Spark SQL、MLlib、GraphX，这些内建库都提供了高级抽象，可以用非常简洁的代码实现复杂的计算逻辑、这也得益于Scala编程语言的简洁性。这里，我们基于1.3.0版本的Spark搭建了计算平台，实现基于Spark Streaming的实时计算。我们的应用场景是分析用户使用手机App的行为。手机客户端会收集用户的行为事件（我们以点击事件为例），将数据发送到[数据服务](https://cloud.tencent.com/solution/data-collect-and-label-service?from=10680" \t "https://cloud.tencent.com/developer/article/_blank)器，我们假设这里直接进入到Kafka[消息队列](https://cloud.tencent.com/product/cmq?from=10680)。后端的实时服务会从Kafka消费数据，将数据读出来并进行实时分析，这里选择Spark Streaming，因为Spark Streaming提供了与Kafka整合的内置支持，经过Spark Streaming实时计算程序分析，将结果写入[Redis](https://cloud.tencent.com/product/crs?from=10680" \t "https://cloud.tencent.com/developer/article/_blank)，可以实时获取用户的行为数据，并可以导出进行离线综合统计分析。Spark Streaming提供了一个叫做DStream（Discretized Stream）的高级抽象，DStream表示一个持续不断输入的数据流，可以基于Kafka、TCP Socket、Flume等输入数据流创建。在内部，一个DStream实际上是由一个RDD序列组成的。Sparking Streaming是基于Spark平台的，也就继承了Spark平台的各种特性，如容错（Fault-tolerant）、可扩展（Scalable）、高吞吐（High-throughput）等。在Spark Streaming中，每个DStream包含了一个时间间隔之内的数据项的集合，我们可以理解为指定时间间隔之内的一个batch，每一个batch就构成一个RDD数据集，所以DStream就是一个个batch的有序序列，时间是连续的，按照时间间隔将数据流分割成一个个离散的RDD数据集。

### Spark Straming与Storm的区别

Storm是纯实时的流式处理框架，SparkStreaming是准实时的处理框架（微批处理）。因为微批处理，SparkStreaming的吞吐量比Storm要高。Storm 的事务机制要比SparkStreaming的要完善。Storm支持动态资源调度。(spark1.2开始和之后也支持)

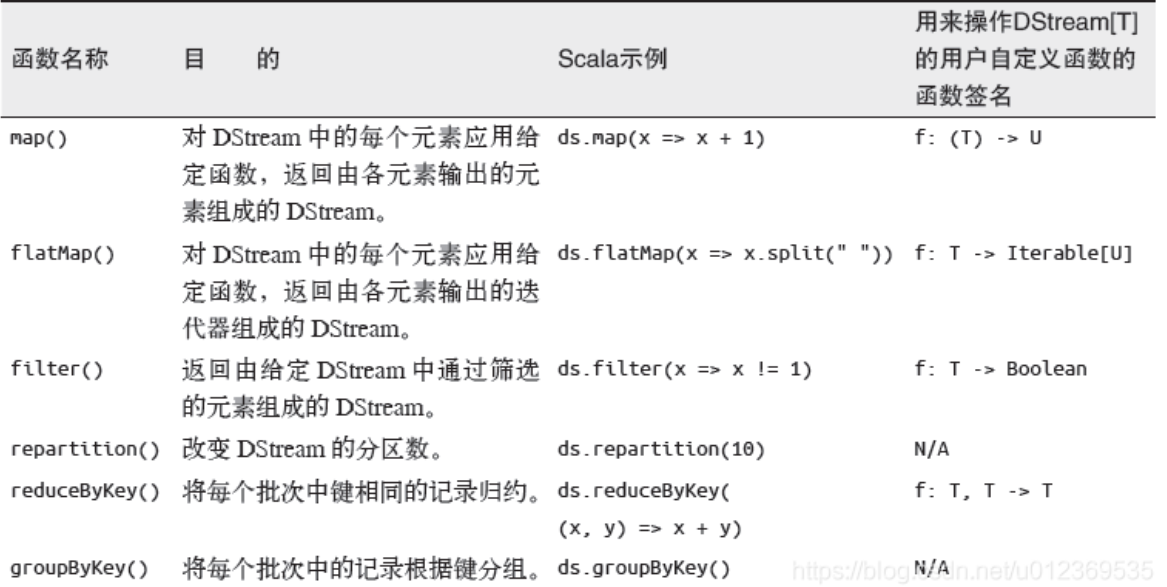
SparkStreaming擅长**复杂的业务处理**，Storm不擅长复杂的业务处理，擅长简单的汇总型计算。

### 列举SparkStreaming算子

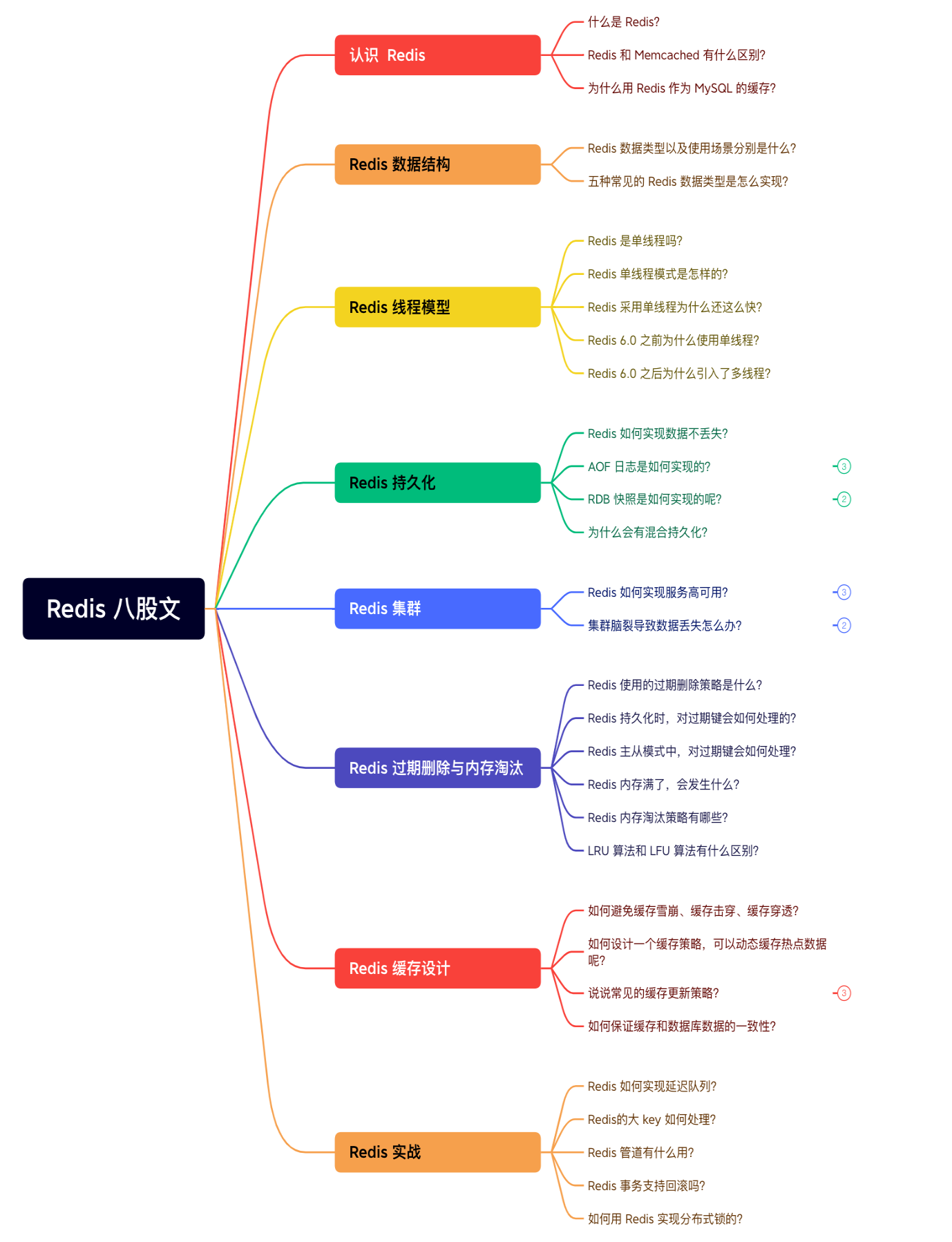
foreachRDD属于output operation算子； transform属于transformation算子； updateStateByKey

59.描述Spark Streamin运行架构

总体来说，Spark Streaming是将流式计算分解成一系列短小的批处理作业。这里的批处理引擎是Spark Core，也就是Spark Streaming将输入数据按照batch interval（如5秒）分成一段一段的数据（Discretized Stream），每一段数据都转换成Spark中的RDD（Resilient Distributed Dataset），然后将Spark Streaming中对DStream的Transformation操作变为针对DSteam内各个RDD的Transformation操作，将RDD经过操作变成中间结果保存在内存中。整个流式计算根据业务的需求可以对中间的结果进行叠加或者存储到外部设备.

60.举例说明DStream的转化操作

# 一、Redis



**一、什么是Redis？**

Redis 是一种基于内存的数据库，对数据的读写操作都是在内存中完成，因此**读写速度非常快**，常用于**缓存，消息队列、分布式锁等场景**。

Redis 提供了多种数据类型来支持不同的业务场景，比如 String(字符串)、Hash(哈希)、 List (列表)、Set(集合)、Zset(有序集合)、Bitmaps（位图）、HyperLogLog（基数统计）、GEO（地理信息）、Stream（流），并且对数据类型的操作都是**原子性**的，因为执行命令由单线程负责的，不存在并发竞争的问题。

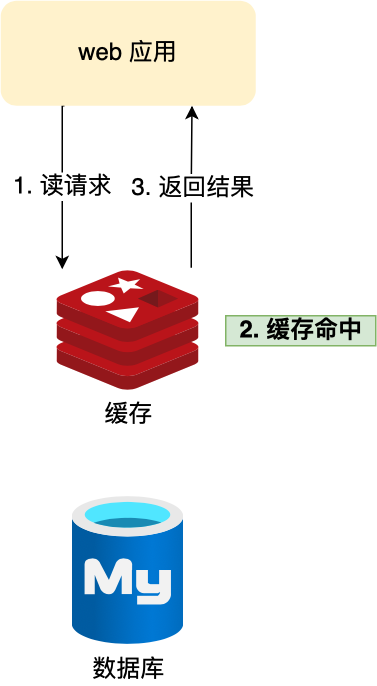
除此之外，Redis 还支持**事务 、持久化、Lua 脚本、多种集群方案（主从复制模式、哨兵模式、切片机群模式）、发布/订阅模式，内存淘汰机制、过期删除机制**等等。

**二、为什么用 Redis 作为 MySQL 的缓存？**

主要是因为 **Redis 具备「高性能」和「高并发」两种特性**。

***1、Redis 具备高性能***

假如用户第一次访问 MySQL 中的某些数据。这个过程会比较慢，因为是从硬盘上读取的。将该用户访问的数据缓存在 Redis 中，这样下一次再访问这些数据的时候就可以直接从缓存中获取了，操作 Redis 缓存就是直接操作内存，所以速度相当快。



如果 MySQL 中的对应数据改变的之后，同步改变 Redis 缓存中相应的数据即可，不过这里会有 Redis 和 MySQL 双写一致性的问题，后面我们会提到。

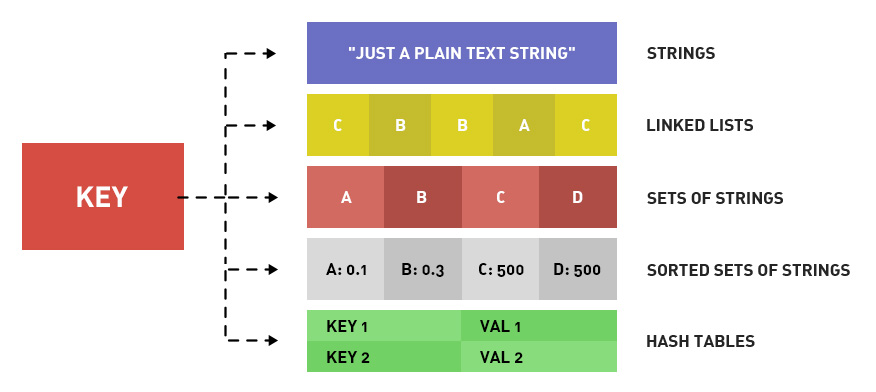
***2、 Redis 具备高并发***

单台设备的 Redis 的 QPS（Query Per Second，每秒钟处理完请求的次数） 是 MySQL 的 10 倍，Redis 单机的 QPS 能轻松破 10w，而 MySQL 单机的 QPS 很难破 1w。

所以，直接访问 Redis 能够承受的请求是远远大于直接访问 MySQL 的，所以我们可以考虑把数据库中的部分数据转移到缓存中去，这样用户的一部分请求会直接到缓存这里而不用经过数据库。

**三、Redis 数据类型以及使用场景分别是什么？**

Redis 提供了丰富的数据类型，常见的有五种数据类型：**String（字符串），Hash（哈希），List（列表），Set（集合）、Zset（有序集合）**。





* String 类型的应用场景：缓存对象、常规计数、分布式锁、共享 session 信息等。
* List 类型的应用场景：消息队列（但是有两个问题：1. 生产者需要自行实现全局唯一 ID；2. 不能以消费组形式消费数据）等。
* Hash 类型：缓存对象、购物车等。
* Set 类型：聚合计算（并集、交集、差集）场景，比如点赞、共同关注、抽奖活动等。
* Zset 类型：排序场景，比如排行榜、电话和姓名排序等。

**四、Redis 如何实现数据不丢失？**

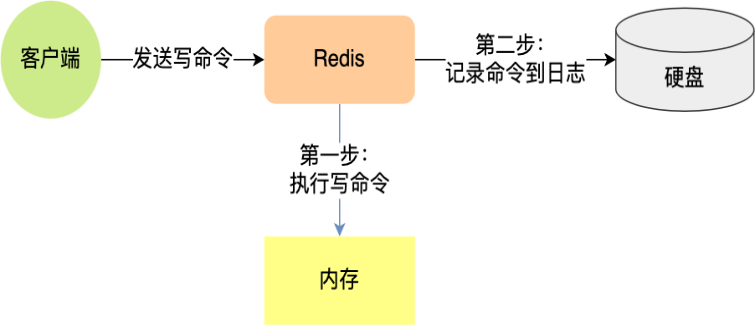
Redis 的读写操作都是在内存中，所以 Redis 性能才会高，但是当 Redis 重启后，内存中的数据就会丢失，那为了保证内存中的数据不会丢失，Redis 实现了数据持久化的机制，这个机制会把数据存储到磁盘，这样在 Redis 重启就能够从磁盘中恢复原有的数据。

Redis 共有三种数据持久化的方式：

* **AOF 日志**：每执行一条写操作命令，就把该命令以追加的方式写入到一个文件里；
* **RDB 快照**：将某一时刻的内存数据，以二进制的方式写入磁盘；
* **混合持久化方式**：Redis 4.0 新增的方式，集成了 AOF 和 RBD 的优点

**五、AOF 日志是如何实现的？**

Redis 在执行完一条写操作命令后，就会把该命令以追加的方式写入到一个文件里，然后 Redis 重启时，会读取该文件记录的命令，然后逐一执行命令的方式来进行数据恢复。



**六、什么是集群脑裂？**

在 Redis 主从架构中，部署方式一般是「一主多从」，主节点提供写操作，从节点提供读操作。 如果主节点的网络突然发生了问题，它与所有的从节点都失联了，但是此时的主节点和客户端的网络是正常的，这个客户端并不知道 Redis 内部已经出现了问题，还在照样的向这个失联的主节点写数据（过程A），此时这些数据被旧主节点缓存到了缓冲区里，因为主从节点之间的网络问题，这些数据都是无法同步给从节点的。

这时，哨兵也发现主节点失联了，它就认为主节点挂了（但实际上主节点正常运行，只是网络出问题了），于是哨兵就会在「从节点」中选举出一个 leader 作为主节点，这时集群就有两个主节点了 —— **脑裂出现了**。

然后，网络突然好了，哨兵因为之前已经选举出一个新主节点了，它就会把旧主节点降级为从节点（A），然后从节点（A）会向新主节点请求数据同步，**因为第一次同步是全量同步的方式，此时的从节点（A）会清空掉自己本地的数据，然后再做全量同步。所以，之前客户端在过程 A 写入的数据就会丢失了，也就是集群产生脑裂数据丢失的问题**。

总结一句话就是：由于网络问题，集群节点之间失去联系。主从数据不同步；重新平衡选举，产生两个主服务。等网络恢复，旧主节点会降级为从节点，再与新主节点进行同步复制的时候，由于会从节点会清空自己的缓冲区，所以导致之前客户端写入的数据丢失了。

**七、什么是缓存雪崩？如何解决？**

当**大量缓存数据在同一时间过期（失效）时，如果此时有大量的用户请求，都无法在 Redis 中处理，于是全部请求都直接访问数据库，从而导致数据库的压力骤增，严重的会造成数据库宕机，从而形成一系列连锁反应，造成整个系统崩溃，这就是缓存雪崩**的问题。

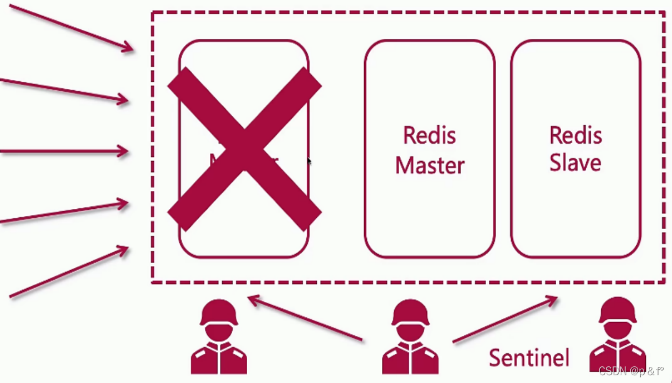
对于缓存雪崩问题，我们可以采用两种方案解决。

* **将缓存失效时间随机打散：** 我们可以在原有的失效时间基础上增加一个随机值（比如 1 到 10 分钟）这样每个缓存的过期时间都不重复了，也就降低了缓存集体失效的概率。
* **设置缓存不过期：** 我们可以通过后台服务来更新缓存数据，从而避免因为缓存失效造成的缓存雪崩，也可以在一定程度上避免缓存并发问题。

**八、什么是哨兵**

[Sentinel](https://so.csdn.net/so/search?q=Sentinel&spm=1001.2101.3001.7020)(哨兵)是用于监控Redis集群中Master状态的工具，是Redis高可用解决方案，哨兵可以监视一个或者多个redis **master服务**，以及这些master服务的所有**从服务**。 某个master服务宕机后，会把这个master下的某个从服务**升级**为master来替代已宕机的master继续工作。

**哨兵模式原理图**



# 二、Hbase

**1. Hbase是什么**

Apache HBase 是 Hadoop 数据库，一个高可靠、高性能、面向列、可伸缩的 分布式数据库，存储非结构化和半结构化数据

2. Hbase和HDFS的区别与联系

HBase**是在**HDFS**的基础之上构建的，**HDFS**是**[分布式文件系统](https://www.zhihu.com/search?q=%E5%88%86%E5%B8%83%E5%BC%8F%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22145551967%22%7D" \t "_blank)。

HDFS是文件系统，而HBase是数据库。

HDFS面向批量访问模式，不是随机访问模式

HBase在HDFS之上提供了**高并发的**[随机写](https://www.zhihu.com/search?q=%E9%9A%8F%E6%9C%BA%E5%86%99&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22145551967%22%7D" \t "_blank)**和支持实时查询**，这是HDFS不具备的。

3. Hbase的作用/功能

以**低成本**来**存储海量**的数据并且支持高并发随机写和实时查询。

4.Hbase支持的检索方式

通过单个Rowkey访问，即按照某个Rowkey键值进行get操作，这样获取唯一一条记录

通过Rowkey的range进行scan，即通过设置startRowKey和endRowKey，在这个范围内进行扫描

全表扫描，即直接扫描整张表中所有行记录

5.需要准确定位一条数据，需要什么条件？

RowKey+Column+时间戳

6. 行键设计原则

Rowkey唯一原则:必须在设计上保证其唯一性

rowkey长度原则:短小精悍,通过rowkey能定位到具体数据，越短越好不必要的数据不要存在rowkey中

Rowkey作为索引原则:Rowkey是Hbase里面唯一的索引，某些查询频繁的限定条件需要把其内容存放在rowkey里面

rowkey顺序原则:精确&必选在前、尽量缩小定位范围>过滤能力越强的字段，越放到前面

7.HBase 的特点是什么？

1）大：一个表可以有数十亿行，上百万列；

2）无模式：每行都有一个可排序的主键和任意多的列，列可以根据需要动态的增加，同一

张表中不同的行可以有截然不同的列；

3）面向列：面向列（族）的存储和权限控制，列（族）独立检索；

4）稀疏：空（null）列并不占用存储空间，表可以设计的非常稀疏；

5）数据多版本：每个单元中的数据可以有多个版本，默认情况下版本号自动分配，是单元

格插入时的时间戳；

6）数据类型单一：Hbase 中的数据都是字符串，没有类型。

8.HBase 和 Hive 的区别？

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。当然，这两种工具是可以同时使用的。就像用 Google 来搜索，用 FaceBook 进行社交一样，Hive 可以用来进行统计查询，HBase 可以用来进行实时查询，数据也可以从 Hive 写到 Hbase，设置再从 Hbase 写回 Hive。

9.HBase 适用于怎样的情景？

① 半结构化或非结构化数据

② 记录非常稀疏

③ 多版本数据

④ 超大数据量

10.描述 HBase 中 scan 和 get 的功能以及实现的异同？

HBase 的查询实现只提供两种方式：

1）按指定 RowKey 获取唯一一条记录，get 方法（org.apache.hadoop.hbase.client.Get）Get 的方法处理分两种 : 设置了 ClosestRowBefore 和没有设置 ClosestRowBefore 的rowlock。主要是用来保证行的事务性，即每个 get 是以一个 row 来标记的。一个 row 中可以有很多 family 和 column。

2）按指定的条件获取一批记录，scan 方法(org.apache.Hadoop.hbase.client.Scan）实现条件查询功能使用的就是 scan 方式。

11. 简述 HBase 中 compact 用途是什么，什么时候触发，分为哪两种，有什么区别，有哪些相关配置参数？

在 hbase 中每当有 memstore 数据 flush 到磁盘之后，就形成一个 storefile，当 storeFile的数量达到一定程度后，就需要将 storefile 文件来进行 compaction 操作。

Compact 的作用：

① 合并文件

② 清除过期，多余版本的数据

③ 提高读写数据的效率

HBase 中实现了两种 compaction 的方式：minor and major. 这两种 compaction 方式的

区别是：

1、Minor 操作只用来做部分文件的合并操作以及包括 minVersion=0 并且设置 ttl 的过

期版本清理，不做任何删除数据、多版本数据的清理工作。

2、Major 操作是对 Region 下的 HStore 下的所有 StoreFile 执行合并操作，最终的结果是整理合并出一个文件。

12.HBase 优化？

（1）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。

（2）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 .

（3）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。接下来我们就谈一谈 rowkey 常用的设计方案

（4）7.4 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。

13.Region 如何预建分区？

预分区的目的主要是在创建表的时候指定分区数，提前规划表有多个分区，以及每个分区的区间范围，这样在存储的时候 rowkey 按照分区的区间存储，可以避免 region 热点问题。

通常有两种方案：

方案 1:shell 方法

create ‘tb\_splits’, {NAME => ‘cf’,VERSIONS=> 3},{SPLITS => [‘10’,‘20’,‘30’]}

方案 2: JAVA 程序控制

· 取样，先随机生成一定数量的 rowkey,将取样数据按升序排序放到一个集合里；

· 根据预分区的 region 个数，对整个集合平均分割，即是相关的 splitKeys；

· HBaseAdmin.createTable(HTableDescriptor tableDescriptor,byte[][]splitkeys)可以指定预分区的 splitKey，即是指定 region 间的 rowkey 临界值。

14、HRegionServer 宕机如何处理？

1）ZooKeeper 会监控 HRegionServer 的上下线情况，当 ZK 发现某个 HRegionServer 宕机之后会通知 HMaster 进行失效备援；

2）该 HRegionServer 会停止对外提供服务，就是它所负责的 region 暂时停止对外提供服务；

3）HMaster 会将该 HRegionServer 所负责的 region 转移到其他 HRegionServer 上，并且会对 HRegionServer 上存在 memstore 中还未持久化到磁盘中的数据进行恢复；

4）这个恢复的工作是由 WAL 重播来完成，这个过程如下：

· wal 实际上就是一个文件，存在/hbase/WAL/对应 RegionServer 路径下。

· 宕机发生时，读取该 RegionServer 所对应的路径下的 wal 文件，然后根据不同的region 切分成不同的临时文件 recover.edits。

· 当 region 被分配到新的 RegionServer 中，RegionServer 读取 region 时会进行是否存在 recover.edits，如果有则进行恢复。

15.HBase 读写流程？

读：

① HRegionServer 保存着 meta 表以及表数据，要访问表数据，首先 Client 先去访问zookeeper，从 zookeeper 里面获取 meta 表所在的位置信息，即找到这个 meta 表在哪个HRegionServer 上保存着。

② 接着 Client 通过刚才获取到的 HRegionServer 的 IP 来访问 Meta 表所在的HRegionServer，从而读取到 Meta，进而获取到 Meta 表中存放的元数据。

③ Client 通过元数据中存储的信息，访问对应的 HRegionServer，然后扫描所在HRegionServer 的 Memstore 和 Storefile 来查询数据。

④ 最后 HRegionServer 把查询到的数据响应给 Client。

写：

① Client 先访问 zookeeper，找到 Meta 表，并获取 Meta 表元数据。

② 确定当前将要写入的数据所对应的 HRegion 和 HRegionServer 服务器。

③ Client 向该 HRegionServer 服务器发起写入数据请求，然后 HRegionServer 收到请求

并响应。

④ Client 先把数据写入到 HLog，以防止数据丢失。

⑤ 然后将数据写入到 Memstore。

⑥ 如果 HLog 和 Memstore 均写入成功，则这条数据写入成功

⑦ 如果 Memstore 达到阈值，会把 Memstore 中的数据 flush 到 Storefile 中。

⑧ 当 Storefile 越来越多，会触发 Compact 合并操作，把过多的 Storefile 合并成一个大

的 Storefile。

⑨ 当 Storefile 越来越大，Region 也会越来越大，达到阈值后，会触发 Split 操作，将

Region 一分为二。

16.如何提高 HBase 客户端的读写性能？请举例说明

1 开启 bloomfilter 过滤器，开启 bloomfilter 比没开启要快 3、4 倍

2 Hbase 对于内存有特别的需求，在硬件允许的情况下配足够多的内存给它

3 通过修改 hbase-env.sh 中的

export HBASE\_HEAPSIZE=3000 #这里默认为 1000m

4 增大 RPC 数量

通过修改 hbase-site.xml 中的 hbase.regionserver.handler.count 属性，可以适当的放大RPC 数量，默认值为 10 有点小。

17.直接将时间戳作为行健，在写入单个 region 时候会发生热点问题，为什么呢？

region 中的 rowkey 是有序存储，若时间比较集中。就会存储到一个 region 中，这样一个 region 的数据变多，其它的 region 数据很少，加载数据就会很慢，直到 region 分裂，此问题才会得到缓解。

18.请描述如何解决 HBase 中 region 太小和 region 太大带来的冲突？

Region 过大会发生多次compaction，将数据读一遍并重写一遍到 hdfs 上，占用io，region过小会造成多次 split，region 会下线，影响访问服务，最佳的解决方法是调整 hbase.hregion.max.filesize 为 256m。

19.hbase如何导入数据？

通过HBase API进行批量写入数据；

使用Sqoop工具批量导数到HBase集群；

使用MapReduce批量导入；

HBase BulkLoad的方式。

20.hbase 的存储结构？

Hbase 中的每张表都通过行键 (rowkey) 按照一定的范围被分割成多个子表（HRegion），默认一个 HRegion 超过 256M 就要被分割成两个，由 HRegionServer 管理，管理哪些 HRegion 由 Hmaster 分配。 HRegion 存取一个子表时，会创建一个 HRegion 对象，然后对表的每个列族 （Column Family） 创建一个 store 实例， 每个 store 都会有 0个或多个 StoreFile 与之对应，每个 StoreFile 都会对应一个 HFile ， HFile 就是实际的存储文件，因此，一个 HRegion 还拥有一个 MemStore 实例。

21.解释下 hbase 实时查询的原理

实时查询，可以认为是从内存中查询，一般响应时间在 1 秒内。HBase 的机制是数据先写入到内存中，当数据量达到一定的量（如 128M），再写入磁盘中， 在内存中，是不进行数据的更新或合并操作的，只增加数据，这使得用户的写操作只要进入内存中就可以立即返回，保证了 HBase I/O 的高性能。

22.为什么不建议在 HBase 中使用过多的列族

在 Hbase 的表中，每个列族对应 Region 中的一个Store，Region的大小达到阈值时会分裂，因此如果表中有多个列族，则可能出现以下现象：

一个Region中有多个Store，如果每个CF的数据量分布不均匀时，比如CF1为100万，CF2为1万，则Region分裂时导致CF2在每个Region中的数据量太少，查询CF2时会横跨多个Region导致效率降低。

如果每个CF的数据分布均匀，比如CF1有50万，CF2有50万，CF3有50万，则Region分裂时导致每个CF在Region的数据量偏少，查询某个CF时会导致横跨多个Region的概率增大。

多个CF代表有多个Store，也就是说有多个MemStore(2MB)，也就导致内存的消耗量增大，使用效率下降。

Region 中的 缓存刷新 和 压缩 是基本操作，即一个CF出现缓存刷新或压缩操作，其它CF也会同时做一样的操作，当列族太多时就会导致IO频繁的问题。

# 三、Flink

1.Flink是什么

Flink是一个**框架**和**分布式**处理引擎，用于在**无边界**和**有边界**数据流上进行**有状态**的计算。Flink 能在所有常见集群环境中运行，并能以内存速度和任意规模进行计算。

2.Flink的特点

Apache Flink 功能强大，支持开发和运行多种不同种类的应用程序。它的主要特性包括：批流一体化、精密的状态管理、事件时间支持以及精确一次的状态一致性保障等。Flink 不仅可以运行在包括 YARN、 Mesos、Kubernetes 在内的多种资源管理框架上，还支持在裸机集群上独立部署。在启用高可用选项的情况下，它不存在单点失效问题。事实证明，Flink 已经可以扩展到数千核心，其状态可以达到 TB 级别，且仍能保持高吞吐、低延迟的特性。世界各地有很多要求严苛的流处理应用都运行在 Flink 之上。

3.Flink提供了哪些更高抽象层的API以方便用户编写分布式任务，分别可以处理何种操作？

1. DataSet API, 对静态数据进行批处理操作，将静态数据抽象成分布式的数据集，用户可以方便的采用Flink提供的各种操作符对分布式数据集进行各种操作，支持Java，Scala和Python。

2. DataStream API，对数据流进行流处理操作，将流式的数据抽象成分布式的数据流，用户可以方便的采用Flink提供的各种操作符对分布式数据流进行各种操作，支持Java和Scala。

3. Table API，对结构化数据进行查询操作，将结构化数据抽象成关系表，并通过Flink提供的类SQL的DSL对关系表进行各种查询操作，支持Java和Scala。

4. Flink的容错机制

Flink基于分布式快照与可部分重发的数据源实现了容错，用户可自定义对整个Job进行快照的时间间隔，当出现任务失败时，Flink将整个Job恢复到最近一次快照的状态，并从数据源重发快照之后的数据。

5.相对于其他流处理系统的容错方案，Flink的特点/优点？

1. 低延迟。由于操作符状态的存储可以是异步的，所以进行快照的过程基本上不会阻塞消息的处理，对消息的延迟不会产生负面的影响。

2. 高吞吐量。当操作符状态较少时，对吞吐量基本没有影响。当操作符状态较多时，相对于其他的容错机制，分布式快照的时间间隔是用户自定义的，所以用户可以权衡错误恢复时间和吞吐量的要求，调整分布式快照的时间间隔。

3. 与业务逻辑的隔离。Flink的分布式快照机制与用户的业务逻辑是完全隔离的，用户的业务逻辑不会依赖或是对分布式快照产生任何影响。

4. 错误恢复代价。分布式快照的时间间隔越短，错误恢复的时间越少，与吞吐量负相关。

6. Flink支持哪些类型的时间窗口，分别适用于何种要求？

1. Operator Time。根据Task所在节点的本地时钟来进行切分的时间窗口。

2. Event Time。消息自带时间戳，根据消息的时间戳进行处理，确保时间戳在同一个时间窗口的所有消息一定会被正确处理。由于消息可能是乱序流入Task的，所以Task需要缓存当前时间窗口消息处理的状态，直到确认属于该时间窗口的所有消息都被处理后，才可以释放其状态。如果乱序的消息延迟很高的话，会影响分布式系统的吞吐量和延迟。

3. Ingress Time。有时消息本身并不带有时间戳信息，但用户依然希望按照消息而不是节点时钟划分时间窗口(例如，避免上面提到的第二个问题)。此时可以在消息源流入Flink流处理系统时，自动生成增量的时间戳赋予消息，之后处理的流程与Event Time相同。Ingress Time可以看成是Event Time的一个特例，由于其在消息源处时间戳一定是有序的，所以在流处理系统中，相对于Event Time，其乱序的消息延迟不会很高，因此对Flink分布式系统的吞吐量和延迟的影响也会更小。

7. Flink如何实现基于时间戳的排序

在流处理系统中，由于流入的消息是无限的，所以对消息进行排序基本上被认为是不可行的。但是在Flink流处理系统中，基于WaterMark，Flink实现了基于时间戳的全局排序。

Flink基于时间戳进行排序的实现思路如下：排序操作符缓存所有流入的消息，当其接收到WaterMark时，对时间戳小于该WaterMark的消息进行排序，并发送到下一个节点，在此排序操作符中释放所有时间戳小于该WaterMark的消息，继续缓存流入的消息，等待下一个WaterMark触发下一次排序。由于WaterMark保证了其之后不会出现时间戳比它小的消息，所以可以保证排序的正确性。需要注意的是，如果排序操作符有多个节点，只能保证每个节点的流出消息是有序的，节点之间的消息不能保证有序，要实现全局有序，则只能有一个排序操作符节点。

8.spark与flink的异同

spark中分布式RDD缓存是一个非常强大的功能，比如在实时计算过程中还需要一些离线大数据与之关联，就可以用spark。

spark实时计算本来就是**微批**处理，所以批处理能做的事情流处理都能做，代码也是批流高度统一

flink重在它的**高实时性**，是真正的实时计算，在状态数据和checkpoint容错上做的比较好，能够做到exactly once，对实时性要求高肯定用flink

9.Flink的应用场景

实时智能推荐

利用Flink流计算帮助用户构建更加实时的智能推荐系统，对用户行为指标进行实时计算，对模型进行实时更新，对用户指标进行实时预测，并将预测的信息推送给Web/App端，帮助用户获取想要的商品信息，另一方面也帮助企业提高销售额，创造更大的商业价值。

复杂事件处理

例如工业领域的复杂事件处理，这些业务类型的数据量非常大，且对数据的时效性要求较高。我们可以使用Flink提供的CEP（复杂事件处理）进行事件模式的抽取，同时应用Flink的SQL进行事件数据的转换，在流式系统中构建实时规则引擎。

实时欺诈检测

在金融领域的业务中，常常出现各种类型的欺诈行为。运用Flink流式计算技术能够在毫秒内就完成对欺诈判断行为指标的计算，然后实时对交易流水进行规则判断或者模型预测，这样一旦检测出交易中存在欺诈嫌疑，则直接对交易进行实时拦截，避免因为处理不及时而导致的经济损失

实时数仓与ETL

结合离线数仓，通过利用流计算等诸多优势和SQL灵活的加工能力，对流式数据进行实时清洗、归并、结构化处理，为离线数仓进行补充和优化。另一方面结合实时数据ETL处理能力，利用有状态流式计算技术，可以尽可能降低企业由于在离线数据计算过程中调度逻辑的复杂度，高效快速地处理企业需要的统计结果，帮助企业更好的应用实时数据所分析出来的结果。

流数据分析

实时计算各类数据指标，并利用实时结果及时调整在线系统相关策略，在各类投放、无线智能推送领域有大量的应用。流式计算技术将数据分析场景实时化，帮助企业做到实时化分析Web应用或者App应用的各种指标。

实时报表分析

实时报表分析说近年来很多公司采用的报表统计方案之一，其中最主要的应用便是实时大屏展示。利用流式计算实时得出的结果直接被推送到前段应用，实时显示出重要的指标变换，最典型的案例就是淘宝的双十一实时战报。

10.Flink和sparkstreaming对比

架构模型上：Spark Streaming 在运行时的主要角色包括：Master、Worker、Driver、Executor，Flink 在运行时主要包含：Jobmanager、Taskmanager和Slot。

Spark Streaming 是微批处理，运行的时候需要指定批处理的时间，每次运行 job 时处理一个批次的数据；Flink 是基于事件驱动的，事件可以理解为消息。事件驱动的应用程序是一种状态应用程序，它会从一个或者多个流中注入事件，通过触发计算更新状态，或外部动作对注入的事件作出反应。

任务调度上：Spark Streaming 的调度分为构建 DGA 图，划分 stage，生成 taskset，调度 task 等步骤，而 Flink 首先会生成 StreamGraph，接着生成 JobGraph，然后将 jobGraph 提交 给 Jobmanager 由它完成 jobGraph 到 ExecutionGraph 的转变，最后由 jobManager 调度执行。

时间机制上：Spark Streaming 只支持处理时间，Structured streaming 则支持了事件时 间和 watermark 机制。flink 支持三种时间机制事件时间，注入时间，处理时间，同时支持watermark 机制处理滞后数据。

容错机制上：二者保证 exactly-once 的方式不同。spark streaming 通过保存 offset 和事务的方式；Flink 则使用两阶段提交协议来解决这个问题。

11.Flink集群有哪些角色？各自有什么作用？

TaskManager，JobManager，Client三种。

JobManager：Master角色。集群的协调者，负责接收Flink Job，协调检查点，Failover 故障恢复等，同时管理Flink集群中从节点TaskManager。

TaskManger：实际负责计算的worker。在其上执行Flink Job的一组Task，每个TaskManager负责管理其所在节点上的资源信息，包括磁盘、网络等。在启动的时候将资源的状态向JobManager汇报。Flink 运行时至少会存在一个 worker 处理器。

Client是Flink程序提交的客户端，当用户提交一个Flink程序时，会首先创建一个Client，该Client首先会对用户提交的Flink程序进行预处理，并建立到 JobManager 的连接，提交到Flink集群中处理。

12.Flink并行度

Flink中的任务被分成多个并行任务，每个并行的实例处理一部分数据，这些并行实例的数量被称为并行度。可以从四个不同层面设置并行度：

操作算子、执行环境、客户端、系统层面。优先级依次降低。

13.Flink三种时间语义？

event time：事件创建

ingestion time：数据进入flink的时间

processing time：进入具体算子的时间。

14.Flink支持哪几种重启策略？分别如何配置？ 重启策略种类

固定延迟重启策略（Fixed Delay Restart Strategy）：尝试一个给定的次数来重启 Job，如果超过了最大的重启次数，Job 最终将失败。在连续的两次重启尝试之间，重启策略会等待一个固定的时间。

故障率重启策略（Failure Rate Restart Strategy）：在 Job 失败后会重启，但是超过失败率后，Job 会最终被认定失败。在两个连续的重启尝试之间，重启策略会等待一个固定的时间。

无重启策略（No Restart Strategy）：Job 直接失败，不会尝试进行重启。

Fallback 重启策略（Fallback Restart Strategy）

15.Flink 在使用 Window 时出现数据倾斜，有什么解决办法？

window 产生的数据倾斜指的是不同的窗口内积攒的数据量不同，主要是由源头 数据的产生速度导致的差异。核心思路：1.重新设计 key 2.在窗口计算前做预聚合

16.Flink任务 delay极高 怎么调优

首先要确定问题产生的原因，找到最耗时的点，确定性能瓶颈点。比如任务频繁反压，找到 反压点。主要通过：资源调优、作业参数调优。

资源调优即是对作业中的 Operator 的并发数 （parallelism）、CPU（core）、堆内存（heap\_memory）等参数进行调优。作业参数调优包括：并行度的设置，State 的设置，checkpoint 的设置。

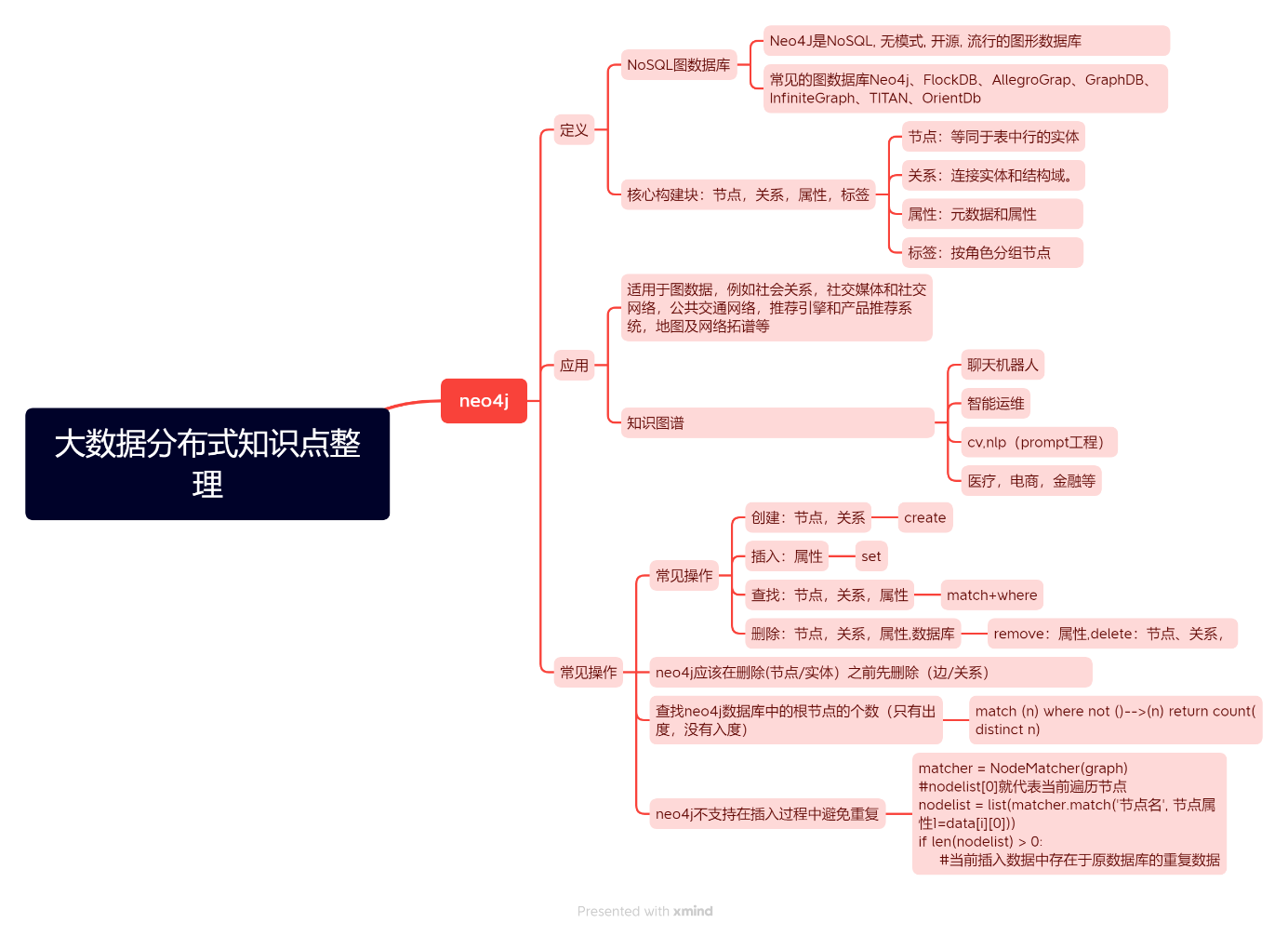
17.Flink内存管理

Flink 并不是将大量对象存在堆上，而是将对象都序列化到一个预分配的内存块上。此外，Flink 大量的使用了堆外内存。如果需要处理的数据超出了内存限制，则会将部分数据存储到硬盘上。Flink 为了直接操作二进制数据实现了自己的序列化框架。

18.Flink如何支持批流一体

在执行引擎这一层，流处理系统与批处理系统最大的不同在于节点间数据传输的方式。对于一个流处理系统，其节点间数据传输的标准模型是：当一条数据被处理完成后，序列化到缓存中，然后立刻通过网络传输到下一个节点，由下一个节点继续处理。而对于一个批处理系统，其节点间数据传输的标准模型是：当一条数据被处理完成后，序列化到缓存中，并不会立刻通过网络传输到下一个节点，当缓存写满，就持久化到本地硬盘上，当所有数据都被处理完成后，才开始将处理后的数据通过网络传输到下一个节点。这两种数据传输模式是两个极端，对应的是流处理系统对低延迟的要求和批处理系统对高吞吐量的要求。Flink的执行引擎采用了一种十分灵活的方式，同时支持了这两种数据传输模型。Flink以固定的缓存块为单位进行网络数据传输，用户可以通过缓存块超时值指定缓存块的传输时机。如果缓存块的超时值为0，则Flink的数据传输方式类似上面提到的流处理系统的标准模型，此时系统可以获得最低的处理延迟。如果缓存块的超时值为无限大，则Flink的数据传输方式类似上面提到的批处理系统的标准模型，此时系统可以获得最高的处理吞吐量。同时缓存块的超时值也可以设置为0到无限大之间的任意值。缓存块的超时阈值越小，则Flink流处理执行引擎的数据处理延迟越低，但吞吐量也会越低，缓存块的超时阈值越大时，则反之。通过调整缓存块的超时阈值，用户可根据自己的需要灵活的权衡Flink的延迟和吞吐量。

# 四、Neo4j



（1)举例neo4j通常的应用场景？

适用于图数据，例如社会关系，社交媒体和社交网络，公共交通网络，推荐引擎和产品推荐系统，地图及网络拓谱等

（2）对比neo4j和mysql说明neo4j的优点

Neo4j是NoSql数据库，专门用于网络图的存储，而mysql是关系型数据库，存储的数据是表。



neo4j允许更快的数据库操作，例如一跳或者多跳查询。在MySql中查询则需要join表。

neo4j数据更直观，使用Cypher语言，

neo4j更灵活。不管有什么新的数据需要存储，都是一律的节点和边，只需要考虑节点属性和边属性。而MySql中即意味着新的表，还要考虑和其他表的关系。

（3）neo4j节点, 关系, 属性和标签等构建块的作用是什么？

节点：等同于表中行的实体。关系：连接实体和结构域。属性：元数据和属性

标签：按角色分组节点

（4）neo4j中SET字句的用途？

SET子句用于向现有节点或关系添加新属性。

（5）neo4j应该在删除(节点/实体）之前先删除（边/关系）

（6）Neo4j中删除节点和关系的属性的命令是？

remove

例如：MATCH (n:book) where n.name=bigdata REMOVE n.price RETURN n

(7）查找neo4j数据库中的根节点的个数（只有出度，没有入度）

match (n) where not ()-->(n) return count(distinct n)

(8) 什么是知识图谱？

知识图谱是结构化的语义知识库，其基本组成单位是『实体-关系-实体』三元组，以及实体及其相关属性-值对，实体之间通过关系相互联结，构成网状的知识结构。本质上是语义网络，是一种基于图的数据结构，由节点和边组成。

（9）知识图谱常见的应用场景？

最常见的如智能运维，聊天机器人等，常依赖于neo4j等图数据库实现。

（10）neo4j在创建节点时，是否支持自动过滤重复节点？如不支持，写出防止节点与数据库中数据重复的创建语句

# 使用py2neo库函数操作neo4j

#其中data是是需要插入的数据，可以以dataframe格式存储

for i in range(1,len(data)):

matcher = NodeMatcher(graph)

#nodelist[0]就代表当前遍历节点

nodelist = list(matcher.match('节点名', 节点属性1=data[i][0]))

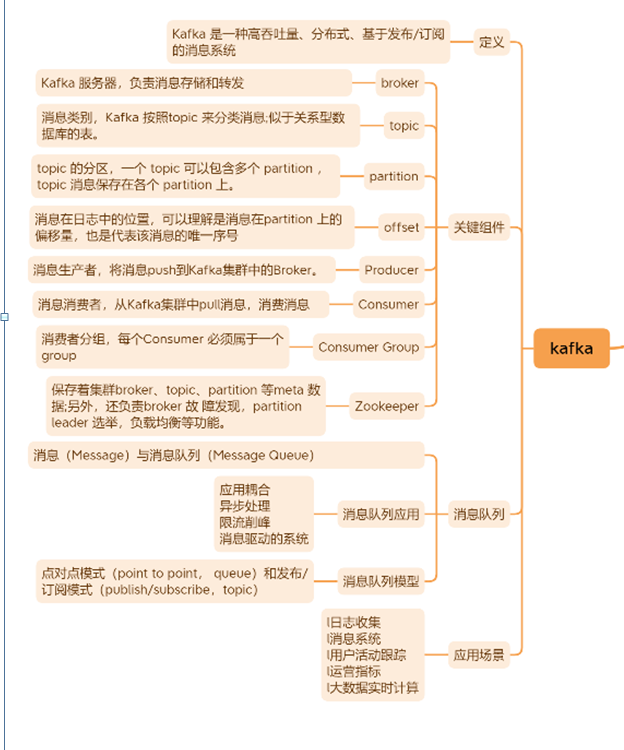
if len(nodelist) > 0:

print(“已经存在”）  
else:

node = Node('节点名', 节点属性=data[i][1])

graph.create(node)

# 五、Kafka



1. Kafka 是什么?

Kafka 是一款分布式流处理框架，用于实时构建流处理应用

1. 简单介绍kakfa的消费组？

消费者组是 Kafka 提供的可扩展且具有容错性的消费者机制。

在 Kafka 中，消费者组是一个由多个消费者实例 构成的组。多个实例共同订阅若干个主题，实现共同消费。同一个组下的每个实例都配置有相同的组 ID，被分配不同的订阅分区。当某个实例挂掉的时候，其他实例会自动地承担起它负责消费的分区。

1. kafka中的broker 的作用？

broker 是消息的代理，Producers往Brokers里面的指定Topic中写消息，Consumers从Brokers里面拉取指定Topic的消息，然后进行业务处理，broker在中间起到一个代理保存消息的中转站

（4) kafka的follower如何与leader同步数据?

Kafka的复制机制既不是完全的同步复制，也不是单纯的异步复制。完全同步复制要求All Alive Follower都复制完，这条消息才会被认为commit，这种复制方式极大的影响了吞吐率。而异步复制方式下，Follower异步的从Leader复制数据，数据只要被Leader写入log就被认为已经commit，这种情况下，如果leader挂掉，会丢失数据，kafka使用ISR的方式很好的均衡了确保数据不丢失以及吞吐率。Follower可以批量的从Leader复制数据，而且Leader充分利用磁盘顺序读以及send file(zero copy)机制，这样极大的提高复制性能，内部批量写磁盘，大幅减少了Follower与Leader的消息量差。

（5）什么情况下一个 broker 会从被 isr中移除？

leader会维护一个与其基本保持同步的Replica列表，该列表称为ISR(in-sync Replica)，每个Partition都会有一个ISR，而且是由leader动态维护 ，如果一个follower比一个leader落后太多，或者超过一定时间未发起数据复制请求，则leader将其重ISR中移除。

（6）Kafka 分区的目的？

对于kafka集群来说,分区可以做到负载均衡,对于消费者来说,可以提高并发度,提高读取效率

（7）Kafka 都有哪些特点？

高吞吐量、低延迟：kafka每秒可以处理几十万条消息，它的延迟最低只有几毫秒，每个topic可以分多个partition, consumer group 对partition进行consume操作。

可扩展性：kafka集群支持热扩展

持久性、可靠性：消息被持久化到本地磁盘，并且支持数据备份防止数据丢失

容错性：允许集群中节点失败（若副本数量为n,则允许n-1个节点失败）

高并发：支持数千个客户端同时读写

（8） Kafka 是如何做到消息的有序性？

kafka 中的每个 partition 中的消息在写入时都是有序的，而且单独一个 partition 只能由一个消费者去消费，可以在里面保证消息的顺序性。但是分区之间的消息是不保证有序的

（9）消费者读取数据的两种方式

pull：消费者主动拉取；数据消费者决定自己何时请求并接收数据,数据持有者只能被动地响应请求

push：生产者主动推送；数据生产者决定何时向消费者推送数据。数据消费者不知道何时会收到数据更新

（10）kafka的应用场景？

日志收集：可以用Kafka收集各种服务的log，通过kafka以统一接口服务的方式开放给各种consumer。

l消息系统：解耦生产者和消费者、缓存消息等。

l用户活动跟踪：kafka经常被用来记录web用户或者app用户的各种活动，如浏览网页、搜索、点击等活动，这些活动信息被各个服务器发布到kafka的topic中，然后消费者通过订阅这些topic来做实时的监控分析，亦可保存到数据库。

l运营指标：kafka也经常用来记录运营监控数据。包括收集各种分布式应用的数据，生产各种操作的集中反馈，比如报警和报告。

l大数据实时计算：kafka被应用到大数据处理，如与spark、storm等整合。