**Spark vs MapReduce**

1.spark基于内存处理数据，MapReduce基于磁盘处理数据

[MapReduce](https://so.csdn.net/so/search?q=MapReduce&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/u013963379/article/details/_blank)是将中间结果保存到磁盘中，减少了内存占用，牺牲了计算性能。

 Spark是将计算的中间结果保存到[内存](https://so.csdn.net/so/search?q=%E5%86%85%E5%AD%98&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/u013963379/article/details/_blank)中,可以反复利用，提高了处理数据的性能

2.Spark在处理数据时构建了DAG有向无环图，减少了shuffle和数据落地磁盘的次数

Spark 计算比 MapReduce 快的根本原因在于 DAG 计算模型。DAG 比MapReduce 在大多数情况下可以减少 shuffle 次数。

Spark 的 DAGScheduler 相当于一个改进版的 MapReduce，

如果计算不涉及与其他节点进行数据交换，Spark 可以在内存中一次性完成这些操作，也就是中间结果无须落盘，减少了磁盘 IO 的操作。

但是，如果计算过程中涉及数据交换，Spark 也是会把 shuffle 的数据写磁盘的有一个误区，**Spark 是基于内存的计算，所以快，这不是主要原因，要对数据做计算，必然得加载到内存，Hadoop 也是如此，只不过 Spark 支持将需要反复用到的数据给 Cache 到内存中，减少数据加载耗时**，所以 Spark 跑机器学习算法比较在行（需要对数据进行反复迭代）

3.Spark是粗粒度资源申请，而MapReduce是细粒度资源申请

粗粒度申请资源指的是在提交资源时，spark会提前向资源管理器（yarn，mess）将资源申请完毕，如果申请不到资源就等待，如果申请到就运行task任务，而不需要task再去申请资源。

MapReduce是细粒度申请资源，提交任务，task自己申请资源自己运行程序，自己释放资源，虽然资源能够充分利用，但是这样任务运行的很慢。

**MapReduce**

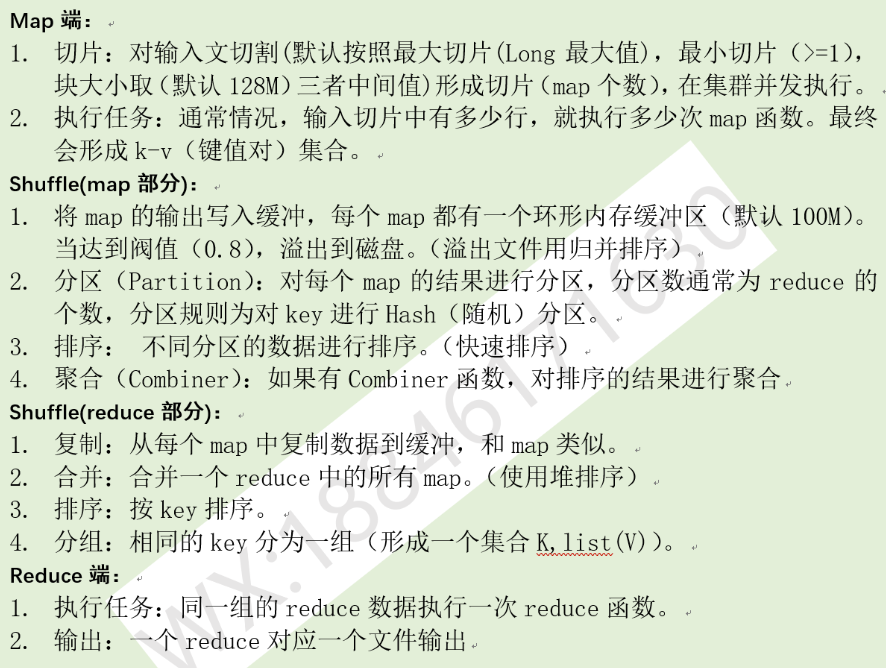
是一种编程模型，用于大规模数据集（大于1TB）的并行运算。

概念"Map（映射）"和"Reduce（归约）"是主要思想

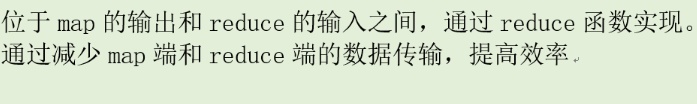
方便在分布式系统运行程序，用Map和Reduce两个函数编程实现基本的并行计算任务

实现是指定一个Map（映射）函数，用来把一组键值对映射成一组新的键值对，指定并发的Reduce（归约）函数，用来保证所有映射的键值对中的每一个共享相同的键组

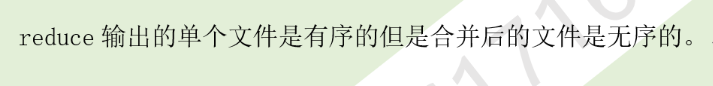
**工作流程**

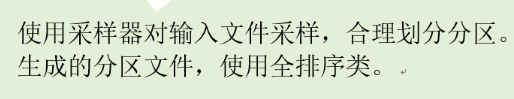


**combiner函数**



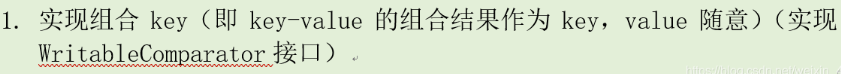
**全排序的原因**

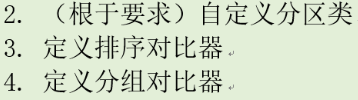


**全排序的流程**

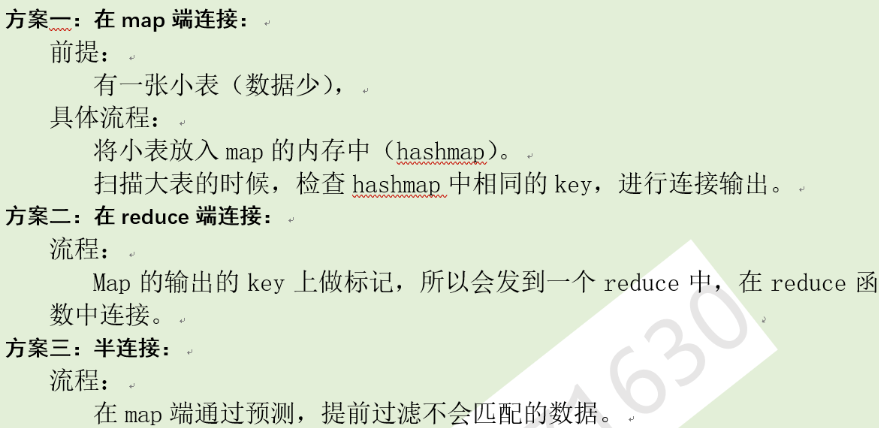
**二次排序目的：对value进行排序**

**二次排序流程：**

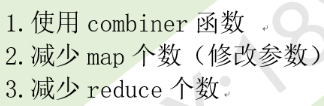
****

****

**MapReduce的join操作**

****

**减少hadoop map端到reduce端的数据传输量**

****

**reduce个数的选择：目标reduce保持在每个运行5分钟左右，而且至少产生一个HDFS块的输出**

**数据倾斜：Key分布不均匀**

**解决reduce端数据倾斜：**

****

**Spark计算引擎**

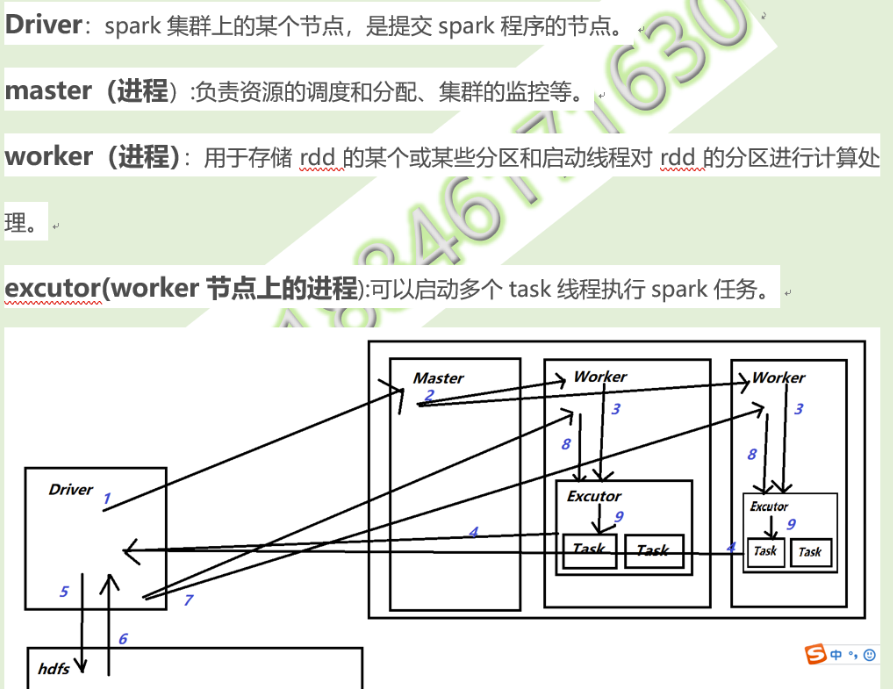
**特点：**分布式，基于内存，迭代式计算

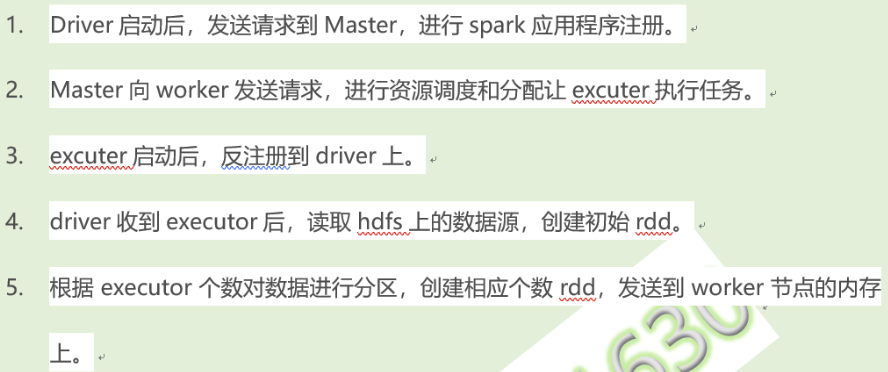
**RDD：**从hdfs上面读取数据形成数据集合rdd，经过分区(设置并行度)，发送到集群上的不同节点的内存里。每个节点的数据集合为一个新的rdd，之后多个rdd在集群里并行运算

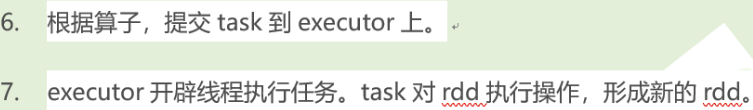
**弹性：**rdd存储在内存中，满了以后会溢出到磁盘

**容错性：**发现节点故障后，会根据来源数据重新计算结果

**提交任务(job)流程(standalone)**

****

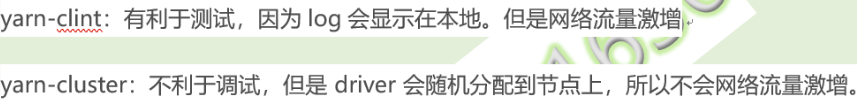
****

****

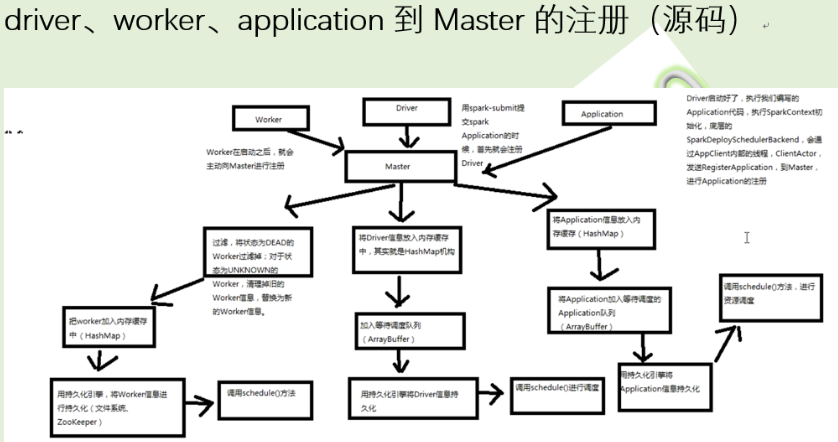
**Master节点宕机、主备切换**

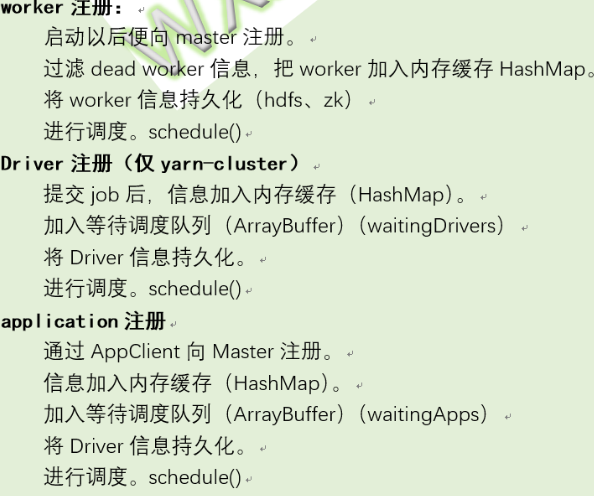
****

**2种提交模式**

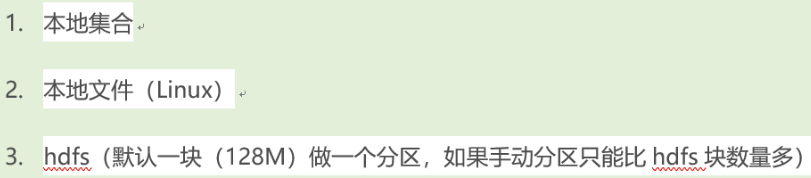
****

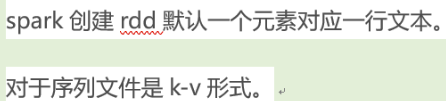
**Spark工作流程**

****

****

**创建RDD的方式：**

****

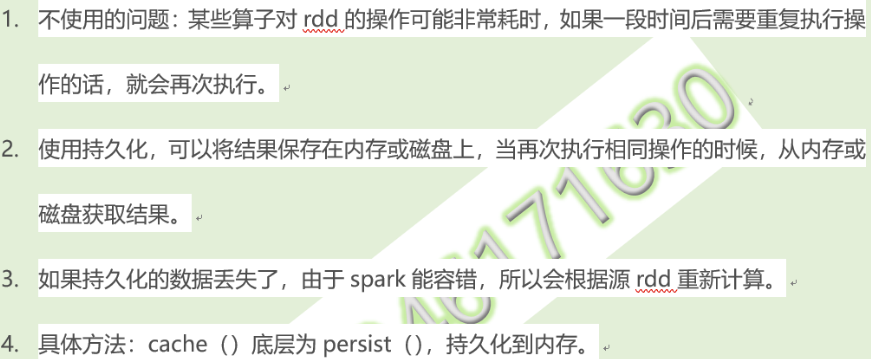
****

**RDD的操作**

transformation：把现有的rdd转换成新rdd，结果为rdd

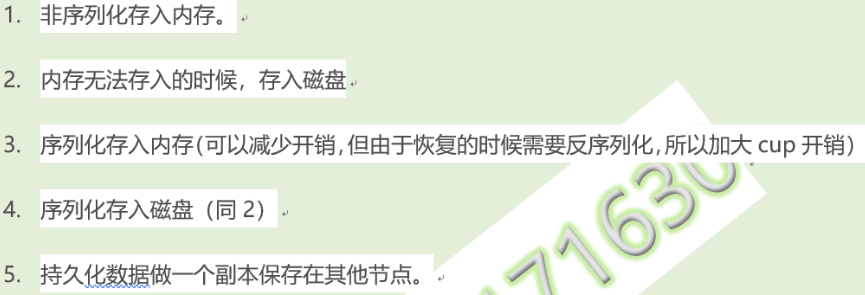
action：将rdd计算出最后结果，返回给driver

**RDD持久化**

****

**1651575738(1)**

**RDD持久化策略**

****

**Spark vs Flink**

Spark是用微批来模拟流的计算,数据流以时间为单位被切分为一个个批次,通过分布式数据集RDD进行批量处理,是一种伪实时,批处理所以吞吐量大但有秒级的延迟

但Flink是基于事件驱动的，它是一个面向流的处理框架, Flink基于每个事件一行一行地流式处理做流式计算. 但他也可以基于流来模拟批来计算来实现批处理，所以能有高吞吐而且延迟也低

和kafka的结合的时候也有不同

其实和Kafka结合的区别还是跟他们的设计理念有关,SparkStreaming是基于微批处理的,所以他采用DirectDstream的方式根据计算出的每个partition要取数据的Offset范围,拉取一批数据形成Rdd进行批量处理,而且该Rdd和kafka的分区是一一对应的;

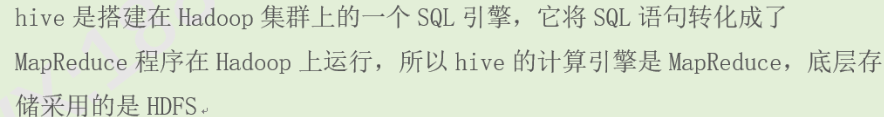
Flink是真正的流处理,他是基于事件触发机制进行处理,在KafkaConsumer拉取一批数据以后,Flink将其经过处理之后变成,逐个Record发送的事件触发式的流处理

另外,Flink支持动态发现新增topic或者新增partition,而SparkStreaming和0.8版本的kafka结合是不支持的,后来跟0.10版本的kafka结合的时候,支持了,看了源码;

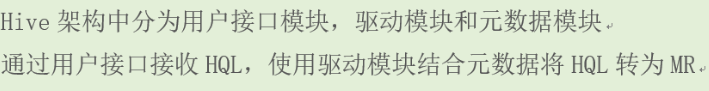
**Hive**

**基于Hadoop的数据仓库**

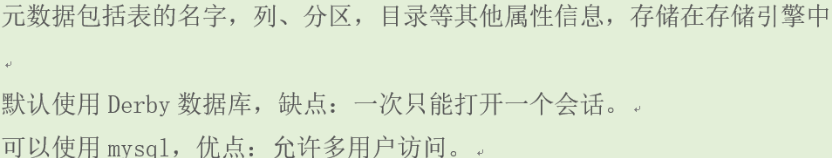
**hive的计算通过什么实现：**

****

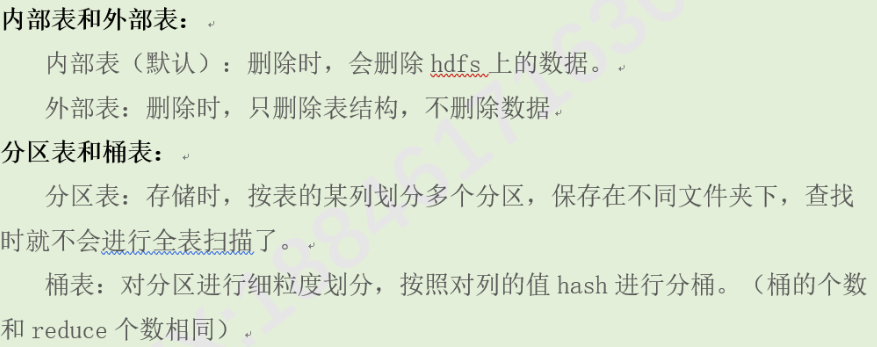
**怎么把HQL转化为MapReduce**

****

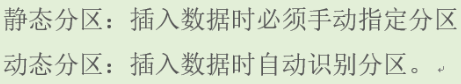
**元数据和存储引擎**

****

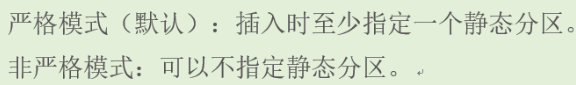
**各种表**

****

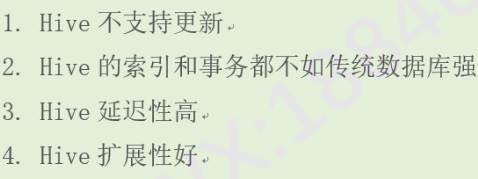
**静态分区和动态分区**

****

**动态分区两种模式：**

****

**Hive和传统数据库的对比**

****

**HDFS**

**Hadoop[分布式文件系统](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F/1250388" \t "https://baike.baidu.com/item/hdfs/_blank)**

**将一个大文件分布式存储在不同的机器节点上**

**HDFS架构**

1.主从架构，由单个NameNode和多个DataNode组成,另外还包括 SecondaryNameNode

2.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。

3.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。  
4.SecondaryNameNode负责合并日志，定期将新的FsImage拷贝回dataNode。

**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）

**写文件过程**

1.客户端向NameNode发送写数据请求

2.NameNode检查是否已存在文件、检查权限。若通过检查，直接先将操作写入 EditLog

3.客户端切分文件为多个block。

4.客户端请求第一个 Block上传到哪几个DataNode服务器上

5.NameNode返回多个DataNode节点，例如返回三个分别为dn1、dn2、dn3。 表示这一个block的多个副本会存储在这些节点上。

6.客户端通过FSDataOutputStream模块请求dn1上传数据，dn1收到请求会继续通信dn2，然后dn2通信dn3，将这个通信管道建立完成。

7. dn1、dn2、dn3逐级应答客户端。

8. 客户端开始往dn1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，dn1收到一个Packet就会传给dn2，dn2传给dn3；dn1每传一个packet会放入一个应答队列等待应答。  
8）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4-8步）

**读文件过程**

1.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。  
2.挑选一台DataNode（就近原则，然后随机）服务器，请求读取数据。  
3.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）。  
4.客户端以Packet为单位接收，先在本地缓存，然后写入目标文件。  
5.然后就是读取下一个块的过程