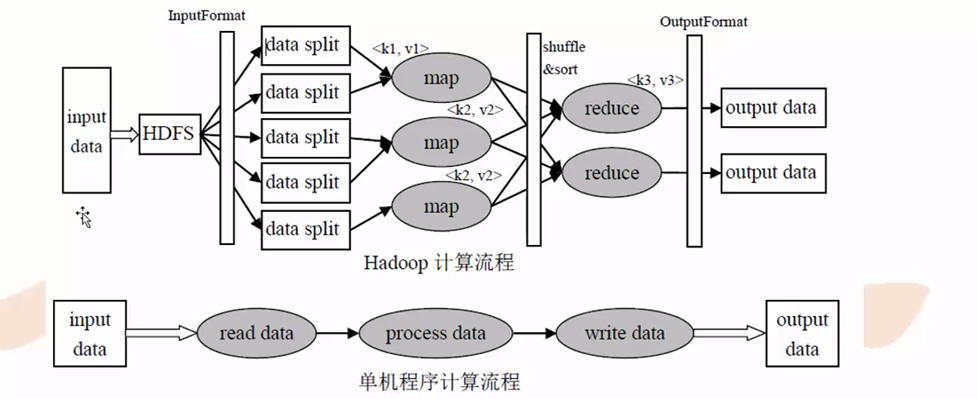
MapReduce初步2



1.刚刚通过一个非常通俗易懂的案例，了解到了一个MapReduce最核心的一个思想，那这个核心思想就是分而治之的思想，MapReduce相当于就是解决数据源，将这个很大的数据进行切割，然后切割成多个小的一个子集或者是从大数据变成一个小的数据，然后再进行切割后进行一个并行计算这么一个应用过程，所以在MapReduce里面刚刚说了两个阶段，第一个阶段是如果面对一个大数据第一个想到的走法就是把这个数据链给分解掉。所以第一个是分解，把这个很大的数据进行划分多个小份进行分解，然后其次对这个分解后的数据进行求解然后再进行合并，所以整个的过程相对来说就这两个阶段，这两个阶段就分别对应了一个Map阶段和一个Reduce阶段。

那也就是说Map就是来负责分，Reduce来负责合，那也就是说map把复杂的问题分解为若干简单的问题，第一就是相当于把这个一个是数据和计算规模的相对于原来任务来说都大大减少了，还有一个原则，刚才数钱的场景，其实里面隐含了一个特点或者是说隐含了一个原则，这个原则的名字叫做经计算原则，就是刚才举得桌子这个例子，让100个人把这个桌子围住，然后每个人把自己面前的那些就是能够的着的钞票都整理到自己的面前，也就是说你只能整理你够的着的数据，但是你够不着的数据那就不属于你的，你只负责处理你面前能够的着的数据，比如说这100个人代表着100不同的计算，然后这个数据相当于是在你面前的，也就是说计算移动，数据不移动，本来数据就是这么分布的，而100个人就是100个可以自由活动的人，然后这100个人你可以去桌子的另外一头，可以去桌子的任何一个角落，但是你一旦去了某一个角落之后，那你就一定要固定下来把这个面前这个数据整在你的面前，就像当于这100个人里面的个人你是一个可移动的计算，但是数据是不动的，然后把这个大的数据变成一个小的任务然后就可以并发执行了，然后这个Reduce负责汇总，那么具体要多少个Reduce，这个可以通过用户具体的规定来决定。

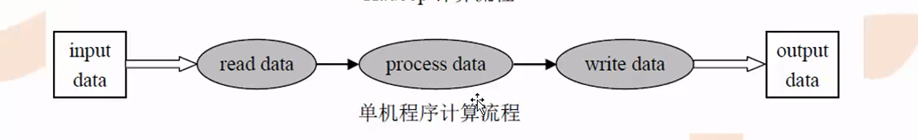
比如说当前的桌子上就有三种面额的钞票，100元50元10元，这时候你不找3个人来进行Reduce计算，却找5个人来进行Reduce计算，这个时候你就会发现一个问题，有两个人是分不了任何钞票的，那相当于他们的计算是空的，因为找的人的个数已经超过了该数钞票任务所需要的人，那这时候你会发现最后你计算出来的结果肯定有两个人最后没有任何的数据产出。



1. 这上面的这张图非常重要，要好好的理解，并且这张图不仅仅要看懂要理解最好自己还要划出来。

我们来看一下这上面这张图里面有两张图，那我们先看非常简单第二张图（如下图1所示）

图1：



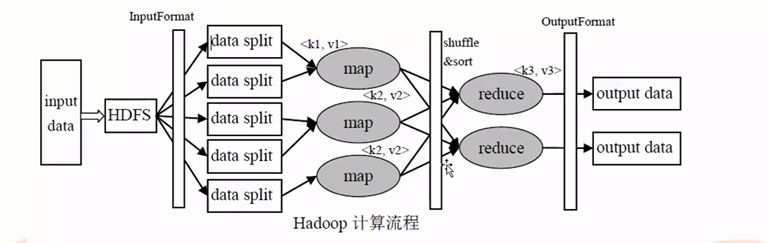
这图1这张图是什么意思呢？是单机程序计算流程，也就是说我们的数据存在了某一台机器上，然后我们就开始写一个程序（input data），这程序就开始对数据开始读取(read data)，读完之后开始对数据进行逻辑处理(process data),比如说把这些个数据把大写变成小写或者是从小写变成大写或者是对这个数据做一些加减法操作等，任何的逻辑处理都可以在process data里面真正处理的环节，处理完之后得到的这个数据肯定是要输出，那输出之前

还需要一个write data输出这么一个模块来对数据进行输出，然后最后把这个数据固化到本地的磁盘上，相当于原始数据来自你当前机器的一个磁盘上的某一个文件，输出也是一个最终在这个机器上生成一个新的文件，那整个的里面的过程都是一些个系列的处理的流程

那在这里面其实对于作为开发者来说仅仅需要关注的点是图中灰色的地方。

这个灰色的部分是我们需要开发代码用的，而白色的地方是因为这个数据它就有了就不需要改了，所以我们只关心灰色需要我们开发的那部分，那有了这个单机的处理流程作为铺垫的话，再去理解这么复杂的图（如下图二所示）的时候就很简单了。

图二：

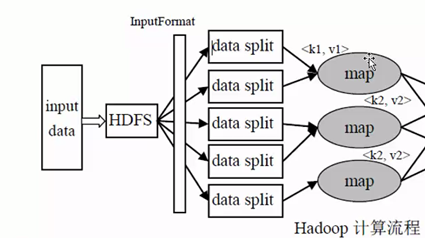


这复杂的图跟刚刚图一的图很相似，也是数据从左到右进行一个数据流转，同样如图二所示数据源从input data，但是这个数据源并不是存在某个机器上，而是存在了HDFS上，也就是这么一坨数据它是分布到散落到不同集群的机器上，这些机器可能都会存那么小份数据，所以HDFS相当于是大的数据都存在HDFS上，而HDFS是分布式文件系统，可以认为这个分布式文件系统是对磁盘做了一些个格式化处理，那你只是把这个磁盘从小店里买来，而你不做格式化你也存不了数据，所以要提前对这个磁盘进行格式化处理，格式化成可以识别数据

那样，然后数据已经存储到HDFS上了，接下来有个叫InputFormat这么一个接口，这个接口是什么呢？你可以认为是MapReduce的一个实现类，在这个类里面提供了两种功能：第一种是数据切分功能和记录读取器功能，那我们先不关心它的记录读取器功能，我们先关心数据切分功能。

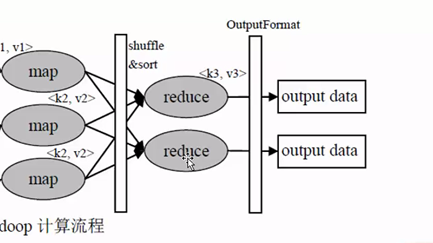
在这个HDFS上（如下图图三所示），通过InputFormat这么一个数据接口会把HDFS里面的大的数据会切分成5个split（就是5个分片），你把一个很大的数据像切蛋糕一样切成了5个不同的小份，然后每一份代表不同的一个小块，那么这个时候切分完的这个每一个子的数据作为后续真正Map这么一个算子要读入的数据源，假设说在这里面分配了3个Map，这个时候可能第一个Map它的输入是来自两个split的，第二个Map它也是输入来自两个的split的，最后一个Map来自一个split。

图三：



然后这个Map读完这个数据处理之后，开始做一些个逻辑处理，那这个逻辑处理相当于开始把你的钱进行分门别类，分门别类之后相当于这时候Map输出就是都是已经整理好的一个结构，然后这个时候分配了两个Reduce（如下图四所示）

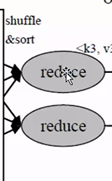
图四：



那这个时候Map就把数据传给reduce，那么哪些数据传给reduce呢？比如当前这个面值，这个面值是key，然后按照这个key做一个哈希算法，然后分配到这个reduce上，然后这个reduce会去读这个数据然后做一些计算相当于后续在整个Mapreduce第二阶段主要找三个人，这三个人分别来去数数的，对这个所有每一个面值的钞票进行一个加法计算，比如说第一个reduce是数100元面值的钞票，那相当于它是把所有的Map节点输出关于100面值的数据全都汇总到第一个reduce里面去，然后reduce计算完以后直接输出，输出到output data大概是这么一个流程。那这个就是mapreduce集成框架。

shuffle 和&sort，shuffle是随机分配，就是把map的那个key做一个哈希，因为后面有两个reduce，所以取模就取模了2，那取模了2就是得到了0和1,然后0就是得到的是如下图五所示的最上面的reduce而1就是下图五下面的最下面的reduce

图五：



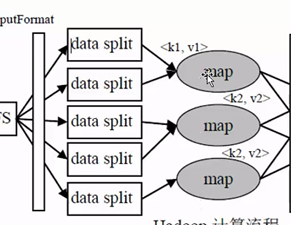
最后得到你当前的这个key取出的那个模然后把这个数据分配到对应的reduce上去。

而这个排序是什么意思呢？

会发现如果说一堆钱，钱里面有100元的，有50元的，有10元的，发现如果按照这个key排序之后，你会发现100元的会统一的跑到一个区间里面去，50元的跑到一个区间，10元的跑到一个区间。

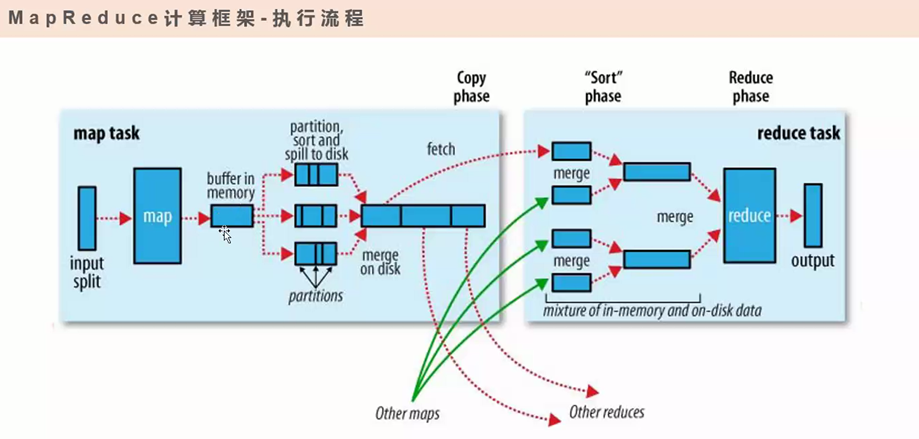
一个split是不是对应一个map？这个是的。那么图六是对应3个map（如下图六所示）

图六：



有一种场景：如果要是这个数据是正常按照文本格式的去存储的话呢，它自然会把每个block成一个record，然后其实每一个split对应一个Map，那如果要说你输入的这个数据它是一个压缩的时候那么它就不能按照按一个split对应一个Map去映射了。但是这图六里面确实是一个问题，一个split对应一个Map。那么这个reduce是可以自己指定的，但这个map也是可以指定（split对应一个map是自动对应的）。

图7：



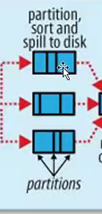
1. 其实这个图7暴露出一些细节，相当于再把刚刚的那个流程再做一次复习把。

在这个map阶段，一个map其实就是对应着一个split分片。

在inputsplit的split分片读进来到map，读进来，因为这个map他是一个程序，他在这个计算机上这个系统里面是启动了一个进程，那么启动了一个进程，那么它会自己维护了一个进程空间，那把这个split数据读进来之后把这个数据直接存到了内存（buffer in memory）上，然后就开始往这个内存去写数据，那这个内存它也是有大小的，默认的内存大小是100M，但是100M很容易就满了，那么这个时候当它写到80M的时候，也就是它已经写到了整个内存的百分之80的时候就不能动了，只能往百分之20上去写，那把这个百分之80已经写好的区域锁住，然后把这百分之80里面的数据开始统一的往磁盘上写并且清空那原先锁住的百分之80的内存数据，但是这个转储的过程中不仅是简简单单的把数据内存直接存到了磁盘上，但是在这个存储的过程中还需要中间涉及到了一次stop排序，会发现这个数据不断的引入，那这个内存（buffer in memory）就会慢慢的产生出很多很多的小的数据，然后每一个数据都是有不同的一个区域，然后这个partition小的数据，然后把内存的数据往这个磁盘上去写的时候它会做排序，比如说我在这个内存里面都是一些100元钞票的数据，50元钞票的数据，10元钞票的数据，那开始拿这个数据的时候它是一个扎龙文章的数据，那么通过一次排序之后会发现100元的会排到一块，50元排到一块，10元排到一块，无论是从大到小排还是从小到大排最后这个key肯定是排到一块很整齐的队列。

假设说就只有这三张不同的面值，那这个时候（如下图8所示）

图8：



这最上面第一个三个长方形是一个小的数据，这小的数据里面又切成了三个部分（如上图8所示）那这三个部分都代表着100元钞票和50元钞票和10元钞票的数据。

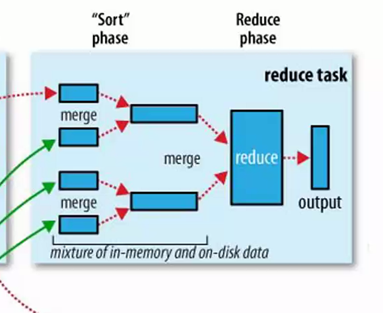
然后接下来又产生出了一个新的文件（如上图8中间那个长方形数据），那这个新的文件可能是100元占的内存很少，而50元占得很多，而10元占得比较少，然后后面它慢慢又产出新的文件数据。每一个小的数据里面都分三个区域。随着数据不断的引入你会发现从这个Map里面会产生出很多很多的这样的小数据文件。那么这个时候发现磁盘上已经有很多这样小的文件了，那应该把这些小的文件进行合并，怎么合并呢？把这小的文件合成一个大的文件并且要保证数据准确的顺序性，那从小文件进行合并是怎么得到的呢？就是用了一次归并排序。

归并排序：开始把指针都指向各自文件的文件头上，然后开始互相之间比较，就是比如三个文件，每一个文件都有一个指针，然后相当于每一个指针都相当于按照从大到小已经排序好的，那么这个时候去往这个大的数据上开始合并。归并排序的前提是你指的每一个数据的都是先排好序的，这就是归并排序。然后做好归并排序之后产生大的文件也是排好序的。

然后就是reduce，因为reduce设定了一定的个数，那么我们只看一个reduce（如下图9所

示）

图9：



比如这个reduce只处理一个100元面值的金额的钞票，把每个map上属于100元的区域的数据全部归纳到reduce里面的merge节点机器上，就是把数据硬拷贝过去reduce里面，那么这时候reduce这机器上面发现可以拷贝出多个数据（merge），那这个数据量也是很大的，因为数据小的文件比较多，这时候就会慢慢的再去做合并(merge)，通过不断的合并慢慢的就合并成大的数据，然后再统一的扔给reduce，然后reduce再进行数据计数工作最后输出，这么一个过程。

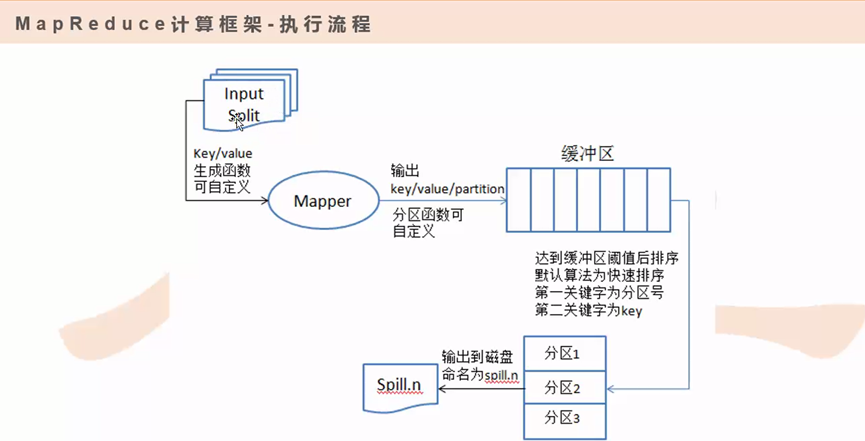
当然reduce读的过程中，它也是把这个数据读到了reduce机器的内存中，然后在内存里面也会有相应的排序。

一个reduce可以处理不同的数据也可以处理一种面值的数据

或者是一个reduce可以处理多个面值钞票。

reduce也是有缓存的。

图10：

1. 

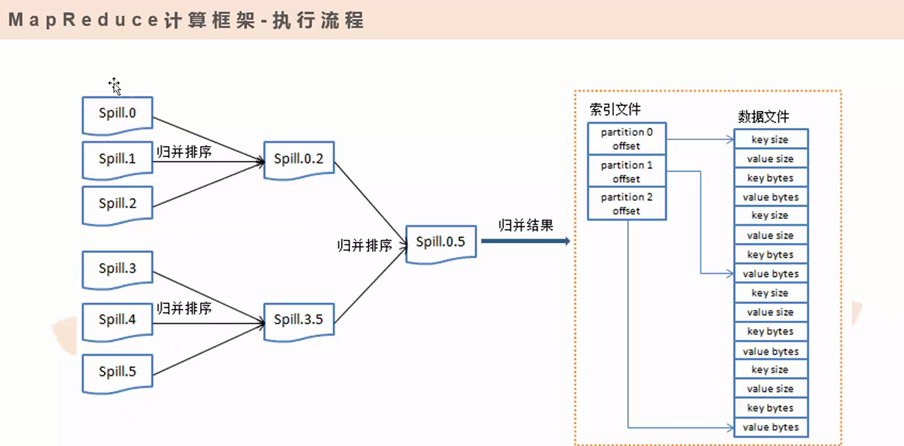
如图10，数据（InputSplit）进来一个map进程(Mapper)，然后把这个数据直接输出到了一个缓冲内存区里面，在这个内存区里面做排序，在这内存区域里面在往磁盘上写数据的时候，这个过程是经历一个排序的，那么排序根据key排序，那么谁是key呢？就是你这个数据在缓冲区里面的会有两列数据一列是一百元钞票，第二列是代表一个1。为什么会有排序呢？

把这个缓冲区的数据不经过排序直接存到磁盘上会发现这个数据是乱的，目的是把相同的key排到一块，然后把缓冲区排序好的数据输出到磁盘上，每次输出都产生了很多小的文件数据(Spill.n)。

接下来有很多的这样一个小的文件数据(Spill.n,n表示数字)

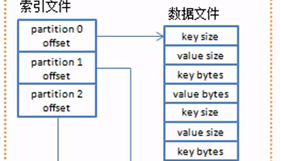
(如下图11所示)

图11：



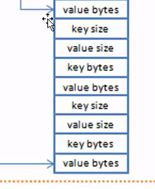
然后再做一个归并排序，变成一个大的数据文件，归并完以后，它是按照key来做一个整体的排序，做完排序以后这里面的key是相同的，既不相同的它这个key也是排序排好的（表示图11中的数据文件）。

如果要是说相同的key呢，那么就是(如下图12所示和图13所示)

图12：  


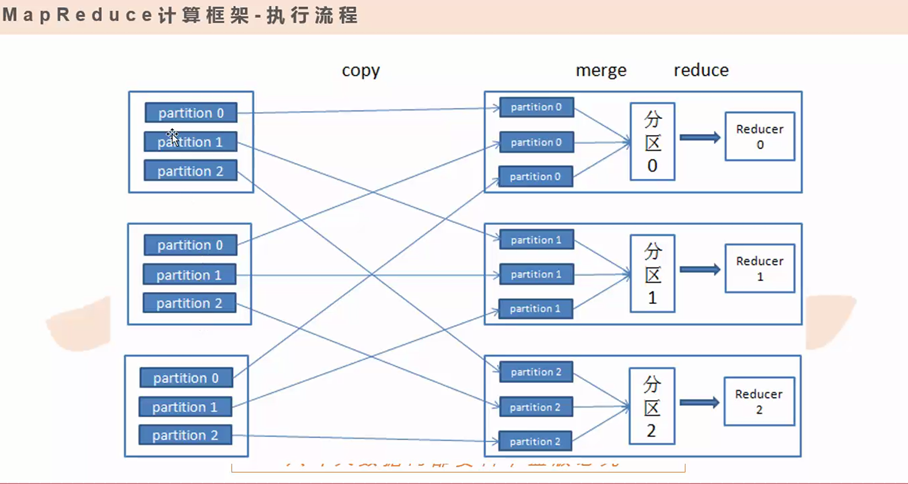
这个数据文件区域就是patrtition0

图13：



这个数据文件区域就是patrtition1，这么去分别key

图14：



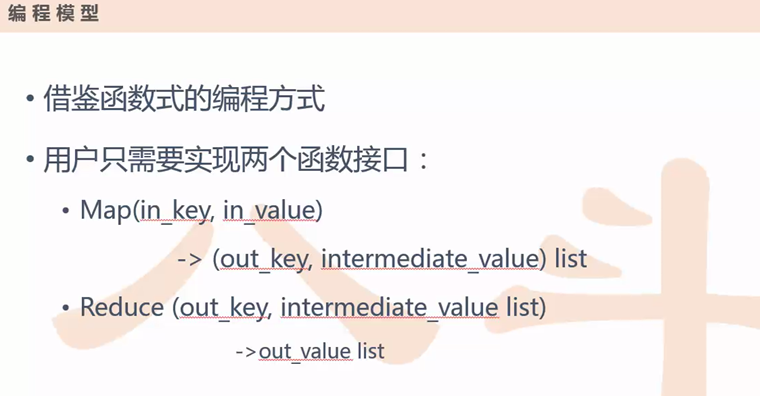
拥有这么多partition之后，那么就是左上角的一个框是每一个map的partition的一个集合

其实每一个map都有那么三个区或者说是都有这么几个partition。

然后右手边的是reduce，有三个reduce，那么这个时候要求是所有0编号的partition全部发送给相应的reduce，相当于把所有全局的partition0编号都发送到了reduce0分区当中，发送到了以后做一个数据的默值，就是变成了一个分区0，这个分区0就是一个把所有这个0编号的这个partition变成一个大的文件(分区0)，然后这个大的文件直接过滤到这个reduce这个进程上面去，每一个reduce都对应着各自的分区进行处理数据，然后最后产出。

1. 那么在MapReduce的开发也是很简单，只需要考虑两个函数就可以了（如下图15所示），那这两个函数分别是一个map函数和一个reduce函数 ，在这个MapReduce框架上开发是不需要考虑中间的一些sort排序和一些partition或者是一些怎么数据读入数据读出这些d偶不需要考虑的，只需要把Map功能和reduce功能这两个功能实现就可以了。

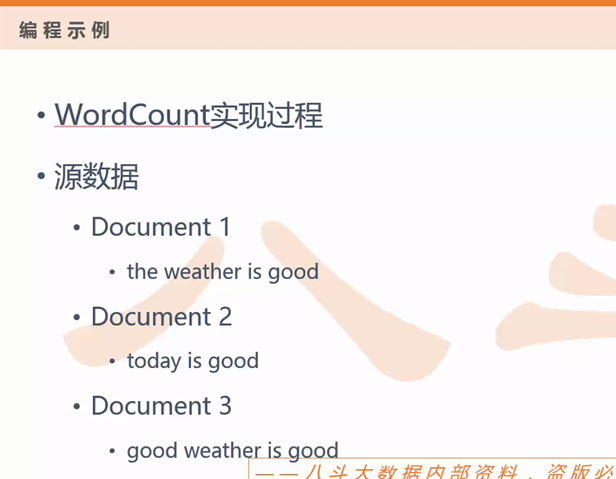
图15：



那么这两个map和reduce功能怎么去实现呢？（如下图16所示）

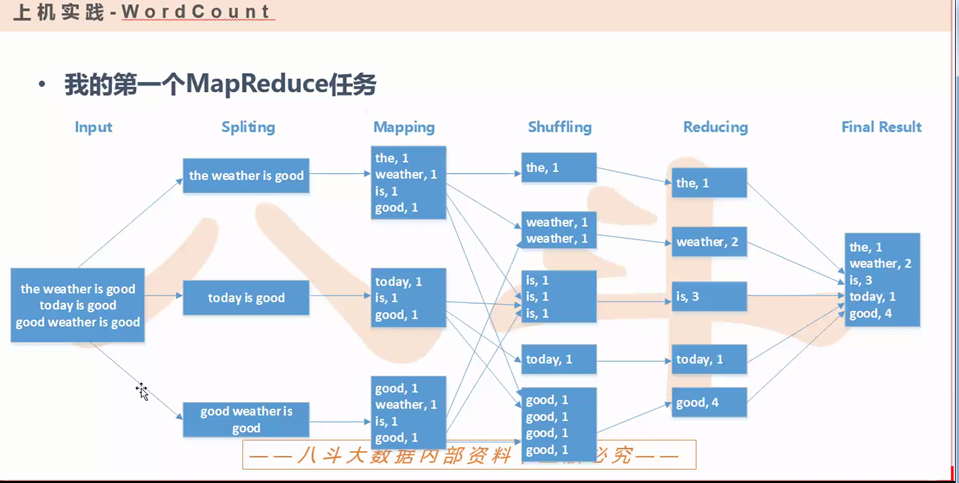
例子：

图16：



这张图16里面有三句话，那么把这个三句话套用刚才图7的这么一个框架来去分析变成了（如下图17所示）的样子

图17：



首先这三句话作为一个大数据源作为input输入进来，然后开始做Spliting分类出三个不同的句子，每一个Spliting分量都是一句话，那么Mapping是整理的，就是把这个句子从头往后读，一遇到一个单词后面就加一个1，那么这个1代表是当前发现的这个单词，这个单词出现过一次，每一个Map节点都有这样的操作。然后做一个Shuffing,这个Shuffing相当于是把这些个相同的key去做一些整理，然后Reducing这个时候开始做一个Reduce计算，这个Reduce里面开始做一个partition，把不同的key去做一个取模，然后找到对应的那个reduce节点上去。那么图17里面有5个reduce，这5个reduce一类key的数据的统计。

reduce做完统计之后，结果就出来了，然后把这个结果进行一个合并然后得到最终的一个结果

（整个流程如下图18和图19所示和图20所示）

图18：

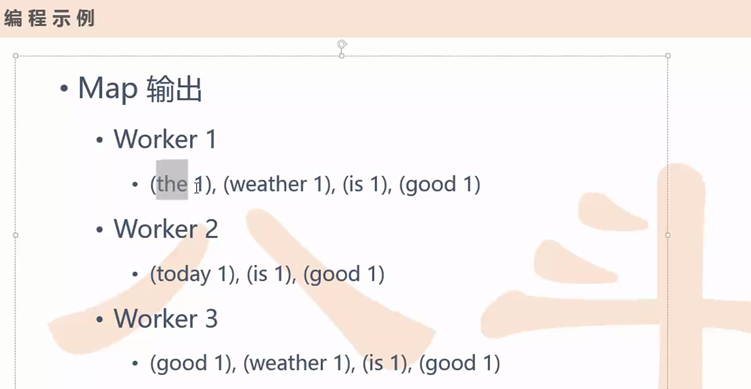


图19：

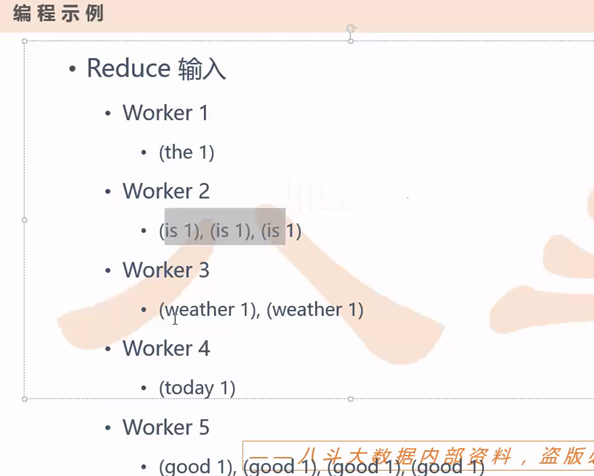
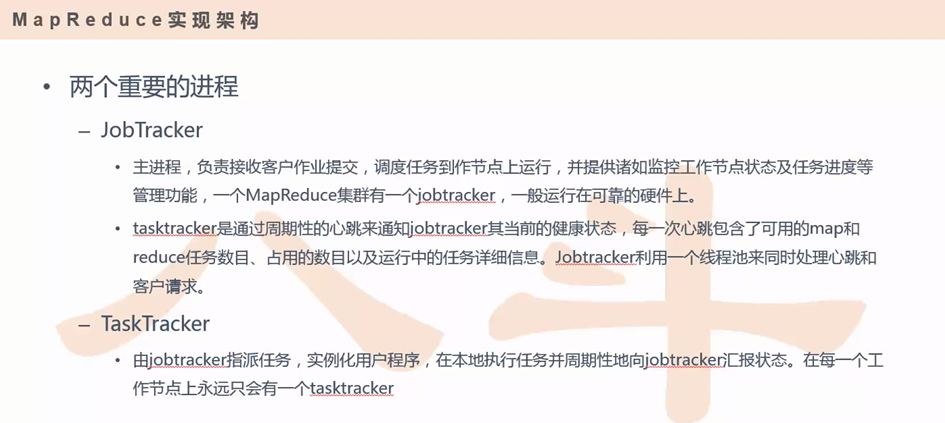


图20：



6.MapReduce集成框架有两个很重要的进程（如下图21所示）

图21：



第一个进程是JobTracker，第二个进程是TaskTracker，

JobTracker是主进程，TaskTracker是从进程。这个主进程是用来接收客户的请求并且做任务调度，然后把这个任务分配到从节点。

而TaskTracker从进程是由JobTracker来派任务的，并且在本地开始进行工作，然后每隔3秒钟会向主进程做一次工作的汇报。

JobTracker不仅是分派任务的也是做监控的，监控从进程的状态是否良好，如果有一台从进程出现故障，那么JobTracker主进程会把任务分派给其他的从进程当中去（如下图22所示）

图22：

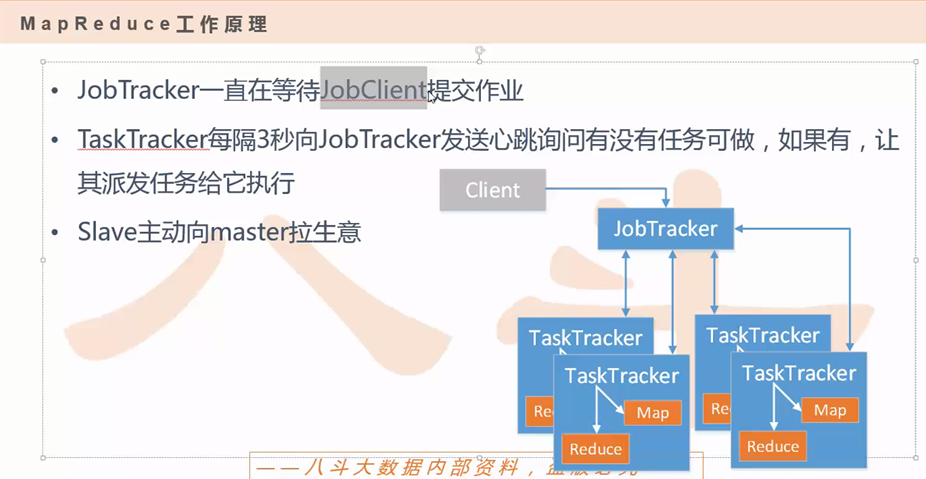
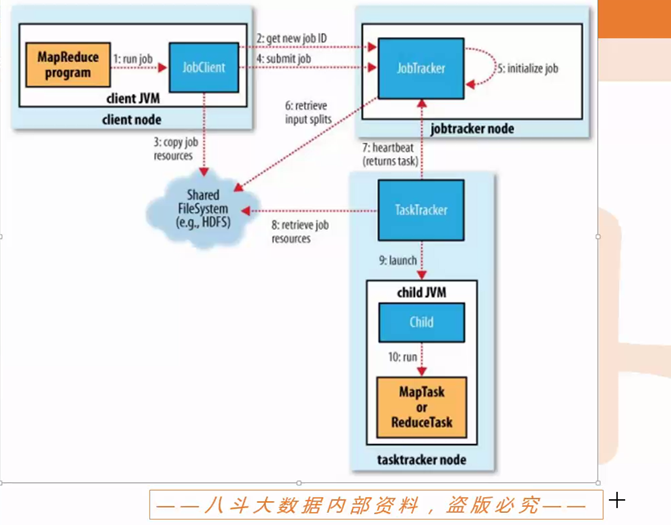


图23是整个Mapreduce提交作业的流程

图23：



如图23所示：首先用户是作为我们工程师，然后提交作业，这个作业是在我们机器上提交，这个机器就是客户端（JobClient），我们JobClient在请求一次任务的时候需要跟JobTracker打交道，首先得跟老板建立关系，然后老板就会按照JobClient所给的要求制定方案，然后会寻找空闲的机器进行工作。这个时候会反馈给这个JobClient一个id号，这个id代表着你的这个任务，这个时候JobClient拿到了这个id可以在HDFS分布式文件系统上创建一个目录，这个目录名字就叫job ID，那么在这个目录里面把客户写的代码要求写的程序要求（就像需求一样）从本地进行拷贝到HDFS上，接下来JobTracker收到了需求以后就开始去工作了，然后这个JobTracker接收了这个客户需求以后开始初始化，然后分配工作把HDFS上的需求数据进一些个分发，分发给TaskTracker从进程进行处理，然后TaskTracker在工作处理过程中也是不断的去汇报给JobTracker当前的工作一个状态。

这个是MapReduce执行的流程。

1. MapReduce作业调度（如下图24所示）

图24：



MapReduce有五种优先级，每一个优先级代表着每一个用户它提交作业的时候，都会分配一个优先级，如果说当有一个用户特别着急，这个任务很急着要做，那么就可以跨越那些低优先级的，然后可以提前做。