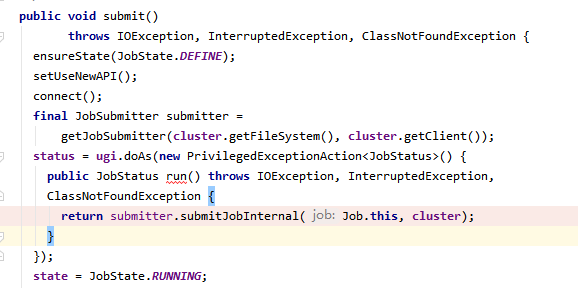
**Mr作业处理流程**

一：客户端提交作业

程序会调用org.apache.hadoop.mapreduce.job 中submit（）方法来提交作业

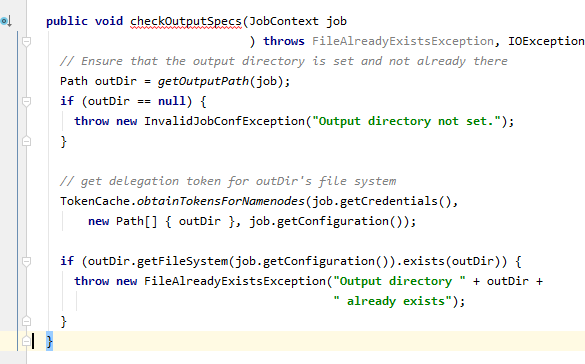


其中connect（）通过读取配置文件获取集群连接（所在文件系统，客户端对象），真正提交作业的是JobSubmitter对象的submitJobInternal方法



该方法中部分执行代码如下：

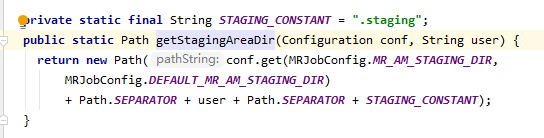
1.checkSpecs(JOB job)：该方法里面会最终会调用org.apache.hadoop.mapreduce.lib.output.FileOutputFormat.java 的checkOutputSpecs（JOB job）方法检查输出路径是否存在，mr任务是不允许输出路径已存在。

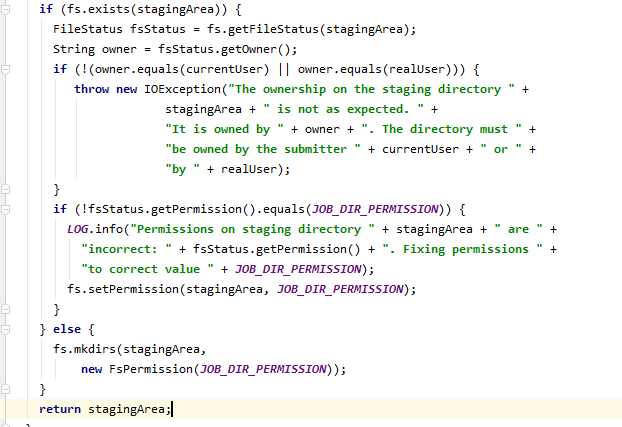


2.JobSubmissionFiles.*getStagingDir*(Cluster cluster, Configuration conf

);

该方法会首先得到并创建一个文件夹，该文件夹路径就是该用户使用yarn框架的初始路径，以后该用户的每个作业目录都会放在该路径下

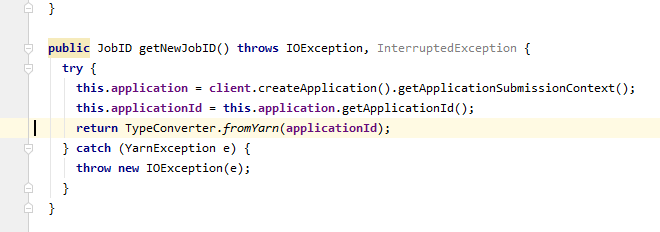




3.设置jobid唯一标志本次提交的作业，用于hadoop能定位作业执行操作：

JobID jobId = **submitClient**.getNewJobID();

调用类org.apache.hadoop.mapred.ResourceMgrDelegate的getNewJobID



4.Path submitJobDir = **new** Path(jobStagingArea, jobId.toString());

将获得的jobId和之前获得的stagingdir拼接处理成一个作业的工作目录，以后作业所需的jar等文件将存放在这里面

1. copyAndConfigureFiles(job, submitJobDir)创建submitJobDir文件夹并从本地上传所需jar等文件到文件系统的submitJobDir下。

该方法会执行JobResourceUploader.upLoadFiles（Job job, Path submitJobDir）

6.切片：

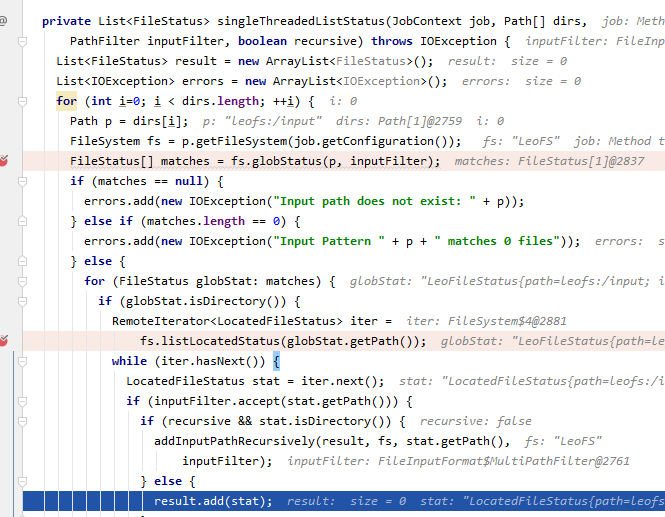
writeSplits(job, submitJobDir)

该代码实际会到FileInputFormat.java的getSplits(JobContext job)方法

getSplit()中根据job中输入路径信息，将输入路径下所有文件的信息返回到一个FlieStatus 类型的list集合中 List<FileStatus> files = listStatus(job)

在listStatus(job)方法下会执行到





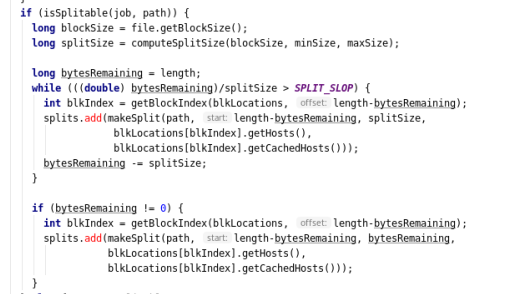
该方法会将返回输入路径下所有文件的LocatedFileStatus，最后FileInputFormat.java的getSplits()方法List<FileStatus> files会得到所有文件的LocatedFileStatus

然后会遍历这个List<FileStatus> files，对每个文件进行分片操作：

在每个文件下首先计算每个切片大小：

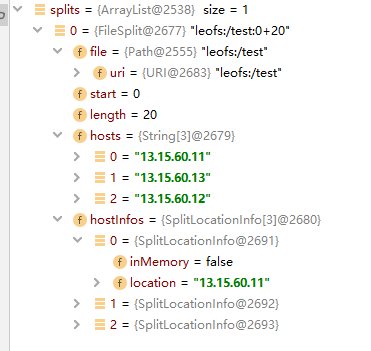
long splitSize = computeSplitSize(blockSize, minSize, maxSize) 该方法默认一个切片大小等于文件块大小

然后进行切片操作：



其中getBlockIndex方法检查每个切片的起始偏移量是否在文件块的大小返回内，并返回所在文件块的索引。

最后返回所有的切片信息集合到List<InputSplit> splits

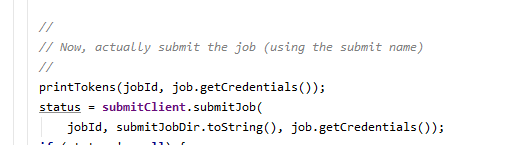


分片结束后，程序会调用*createSplitFiles写两个文件*job.split和job.splitmetainfo到作业目录下，这两个文件会记录所有切片信息。

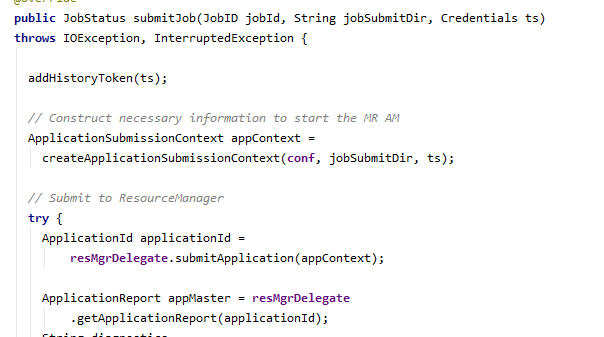
整个分片操作结束后，根据返回的分片数设置map任务个数



接着真正提交任务到yarn框架执行;



该SubmitJob方法执行**YARNRunner.submitJob()**



其中createApplicationSubmissionContext是构建AM启动的必要信息

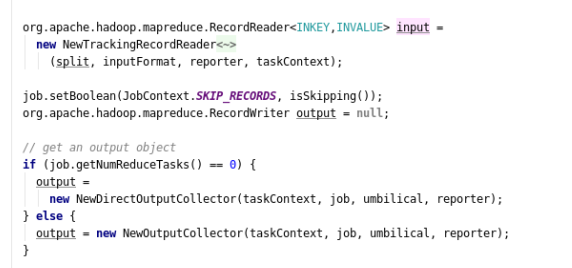
设置资源：默认内存为1536M，cpu的core为1，设置本地资源，比如临时工作目录，jar包等，设置启动AM的命令，检查map和reduce的配置信息，设置环境CLASSPATH等，设置MRAppMaster的执行路径等，并把配置块conf中当前的相关信息、已上传资料所在的目录路径以及有关作业的全部信息都复制转移过去。

1. 将每个Map任务分配到相应主机上并启动map任务
2. 确定在那台主机上启动map任务

2.map启动

程序调用MapTask的run方法启动map任务，接着会调用runNewMapper方法，如下图：该方法首先会获取要用到的类对象



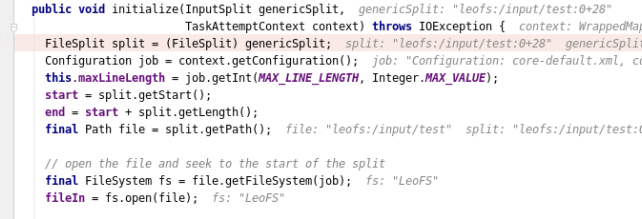


通过反射获取mapper类，初始化输入输出对象，inputsplit类等

接着会调用输入对象（创建的是NewTrackingRecordReade类）的initialize初始化输入



该方法实际会调用到LineRecordReader类的initialize方法，获取切片信息，获取文件输入流，seek到该切片所在文件的起始位置，设置pos值，即读取切片的起始偏移量





其中

if (start != 0) {

start += in.readLine(new Text(), 0, maxBytesToConsume(start));

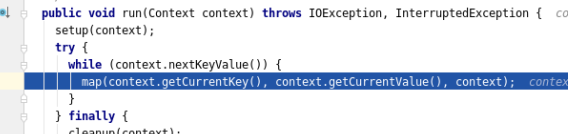
}

this.pos = start;

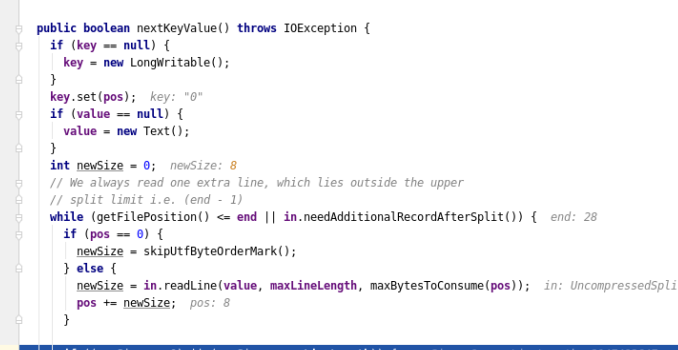
这段代码使得切片不是第一个的时候会丢弃读取该切片的第一行，这里保证了能正确读取一个完整数据被切分到两个不同分片的情况。

初始化输入完成后，开始执行mapper.run方法





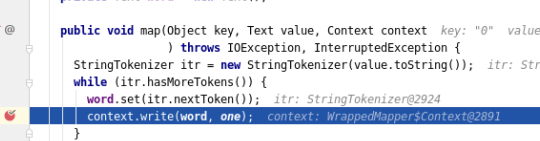
首先context.nextKeyValue会调用LineRecordReader的netxKeyValue方法来对key，value赋值



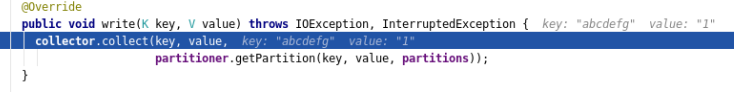
其中readline方法读取一行数据到value中，而该方法最终是调用LineReader的readDefaultLine来完成读取工作的。

map(context.getCurrentKey(), context.getCurrentValue(), context)中getCurrentKey()也最终调用的是LineRecordReader的getCurrentXXX来获得输入的key和value

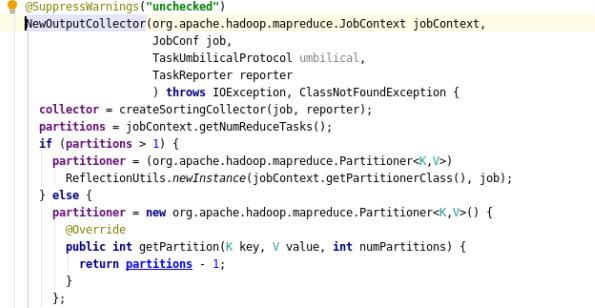
接着map函数处理完一条数据后，将处理后重新得到的<key,value>写出



最终会执行到MapTask中一个内部类NewOutputCollector的write方法



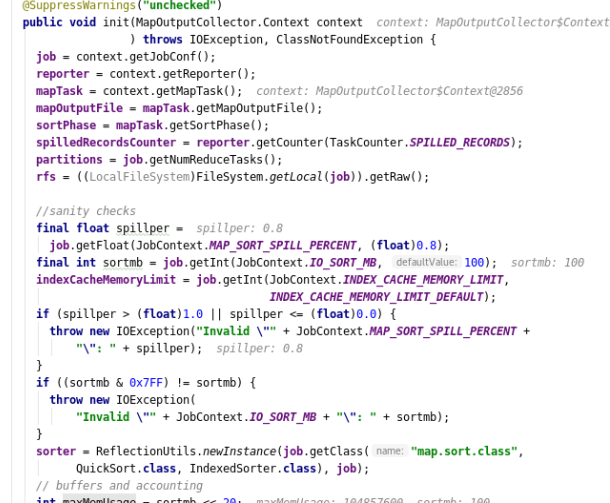
而其中的collector 和 partitioner都是在前面创建输出类NewOutputCollector时初始化的



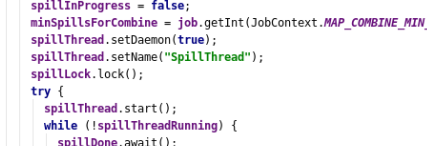
在createSortingCollector方法中会将这个collector做一个初始化操作



Inti方法里面设置溢写阈值，初始化一个快速排序类，以及设置combine操作等



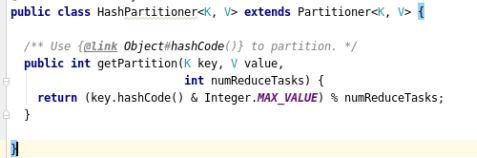
开启溢出写文件的线程



在SpillThread线程中会调用sortAndSpill方法对内存里的数据进行排序并溢写成文件

初始化partitioner

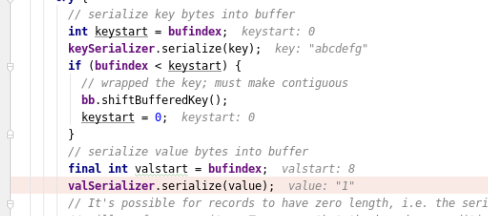
在分区数partitions>1 也就是reduce任务大于一个的时候，去反射得到一个Partitioner对象，当执行该对象的getPartion方法时，会调用Partitioner.java 的子类HashPartioner中重写的getPartion方法来算出当前key所对应的分区号



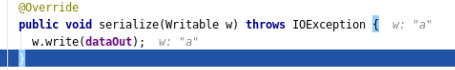
当分区数等于一时，那么返回的partitioner对象中getPartion方法返回的分区号是partitions-1 也就是0

接着执行collector.collect 实际执行到MapTask的内部类MapOutputBuffer.collect方法中

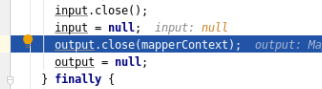
该方法将数据写到缓冲区中

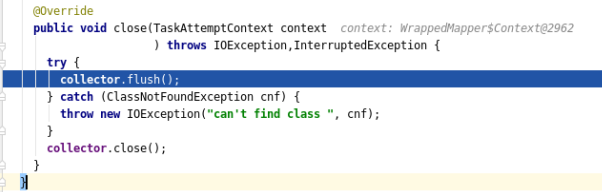


Serialize方法实现写数据到缓冲区

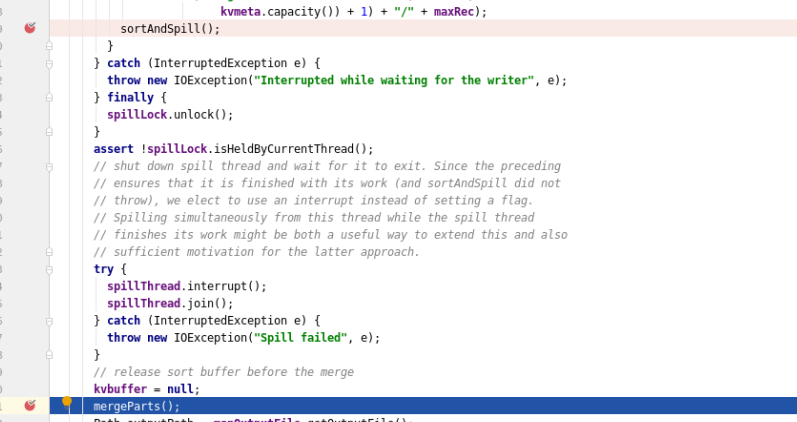


数据全部完成写到缓冲区这步后，程序执行





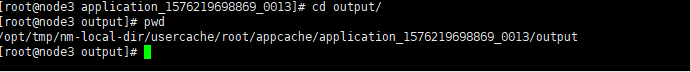
在flush中有如下代码：



这里还执行了一次sortAndSpill方法 来完成对那些没达到溢写条件的还存在缓冲区里的文件做一次排序写文件操作

最后，调用mergeParts（）合并所有溢写出的文件，即该map任务产生的中间结果

最终可在执行该map任务的节点上查看中间结果



三．Reduce