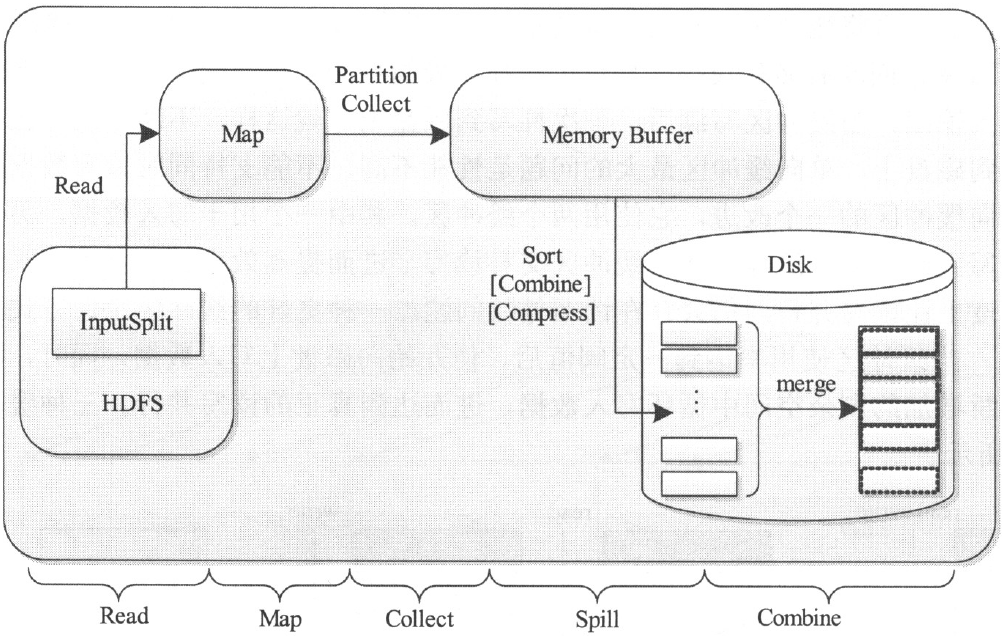
**AM监控MapTask和ReduceTask的进度,如果MapTask中有sort阶段,当map任务执行进度到66.7%后,AM会启动ReduceTask任务,ReduceTask通过协议请求AM去拉取Map任务输出的文件并执行reduce操作.**

## Map task计算流程

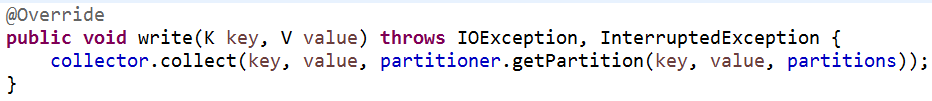


Collect过程:

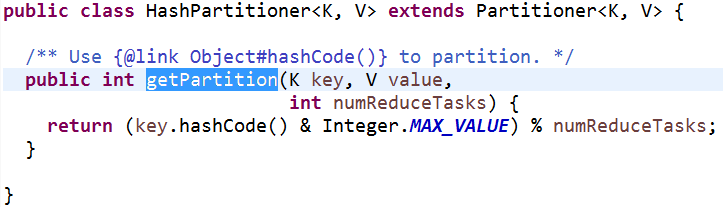
runNewMapper()核心代码：

output = **new** NewOutputCollector(taskContext, job, umbilical, reporter);

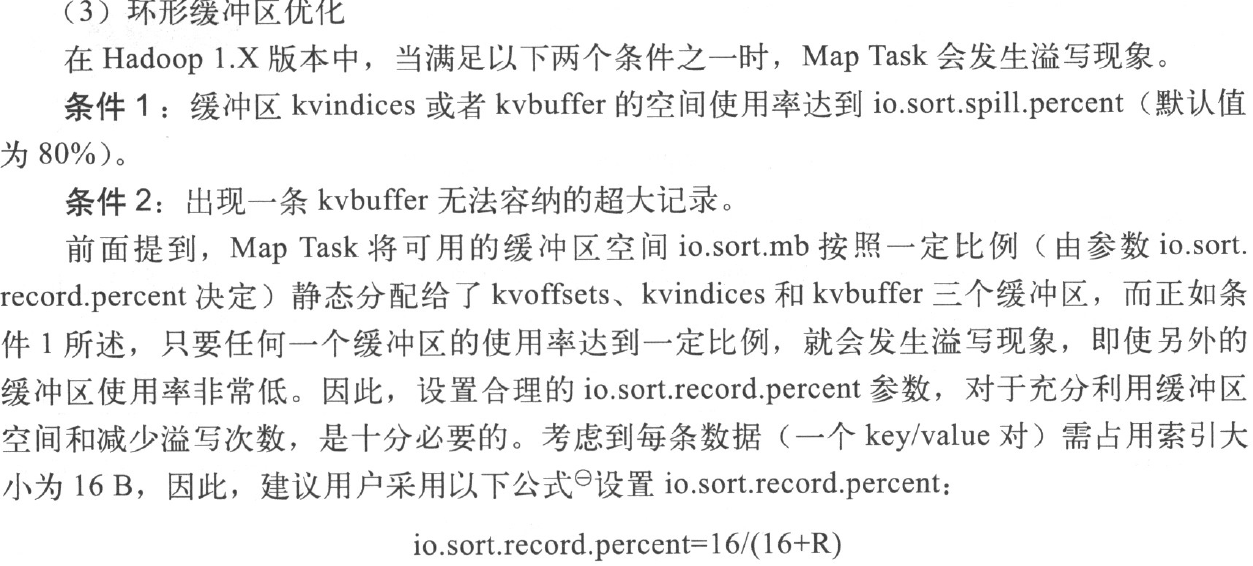
* NewOutputCollector类write方法



**\*partitioner.getPartition(key, value, partitions)获得记录的分区号(即map阶段处理的<key,value>对应的哪个reduce,相同的key分到一个同一地方)**

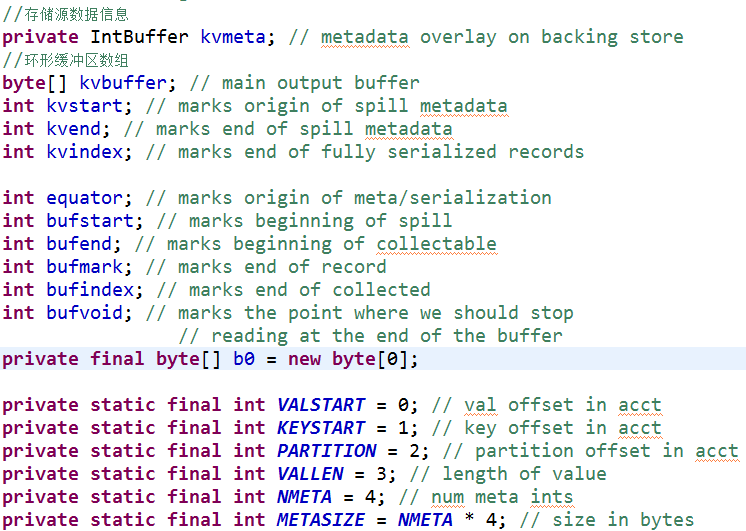


* **MapOutputBuffer 的 collect()方法 : 使用环形缓冲区,读写同时进行。环形内存缓冲区： kvbuffer.总内存空间:io.sort.mb(默认100MB)实际大小(io.sort.mb)\*(mapreduce.map.sort.spill.percent默认0.8)。环形内存缓冲区是Collect,spill阶段同时进行**



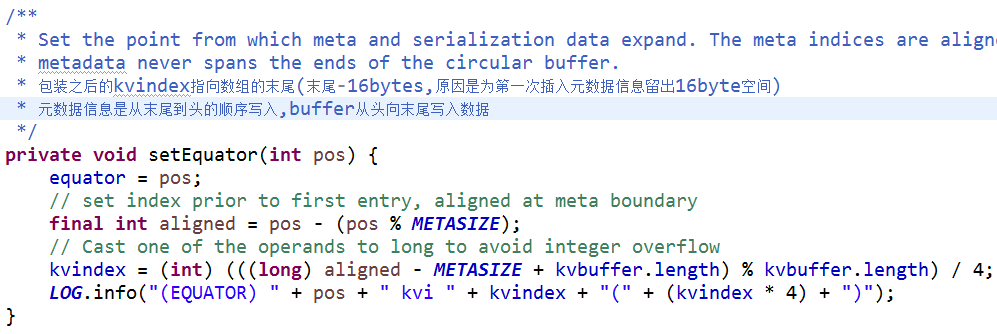
## MapTask中的Collect,Spill,Conbine过程

1. 初始化MR共享环形缓冲区(MapTask.MapOutputBuffer)

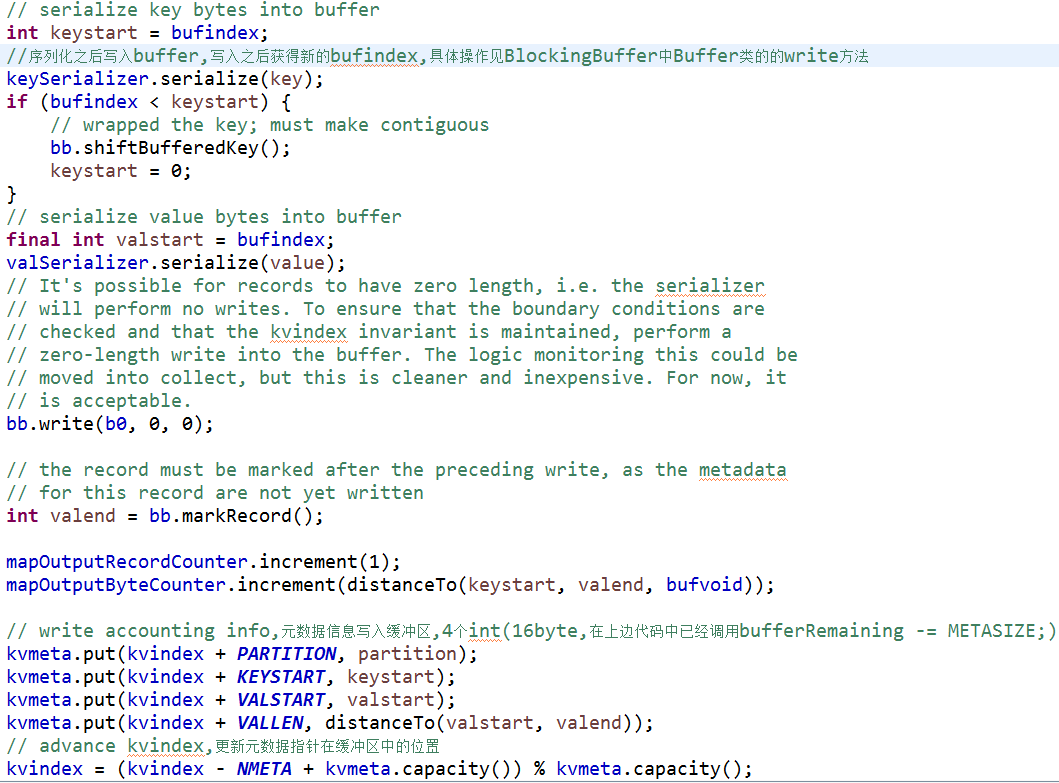


init方法核心代码 :

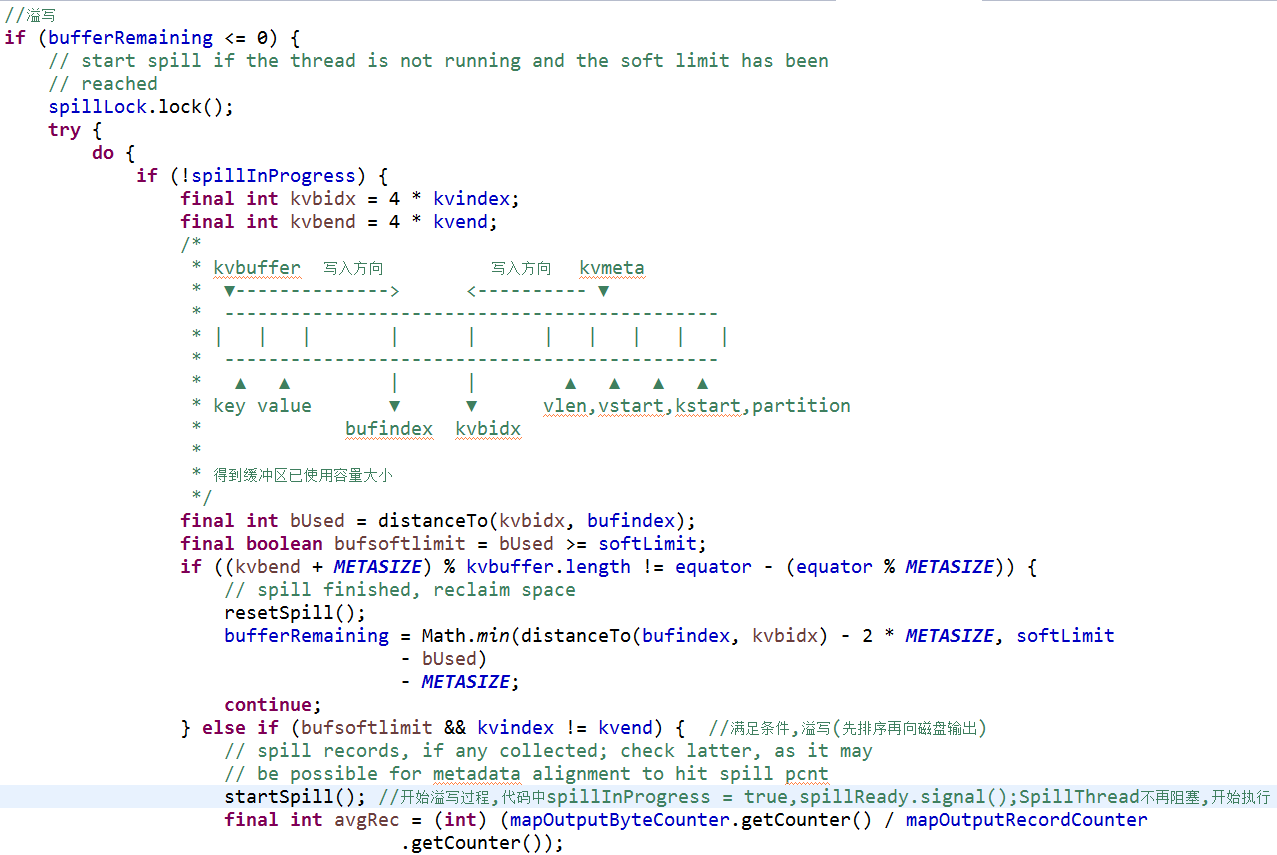




1. 环形缓冲区的数据写入(不分析spill):MapTask.MapOutputBuffer.collect()

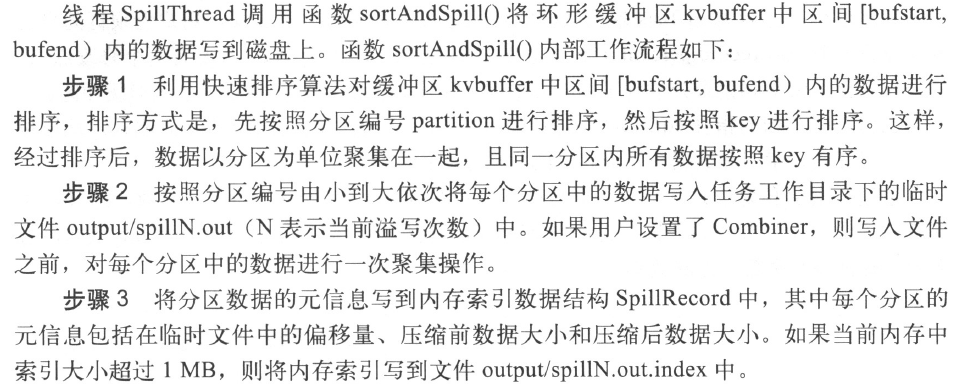


1. 环形缓冲区溢写过程(spill过程,即缓冲区缓存数据超过阈值,由SpillThread线程完成)





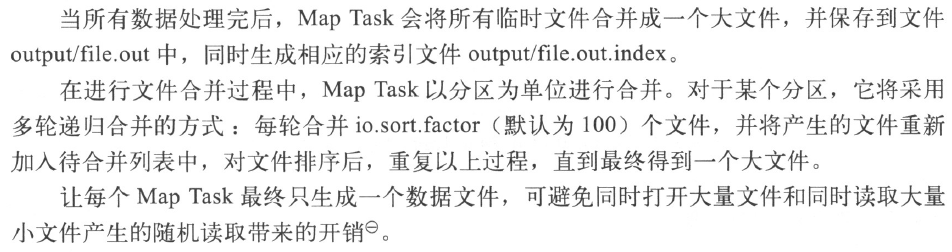
sortAndSpill()方法 ：



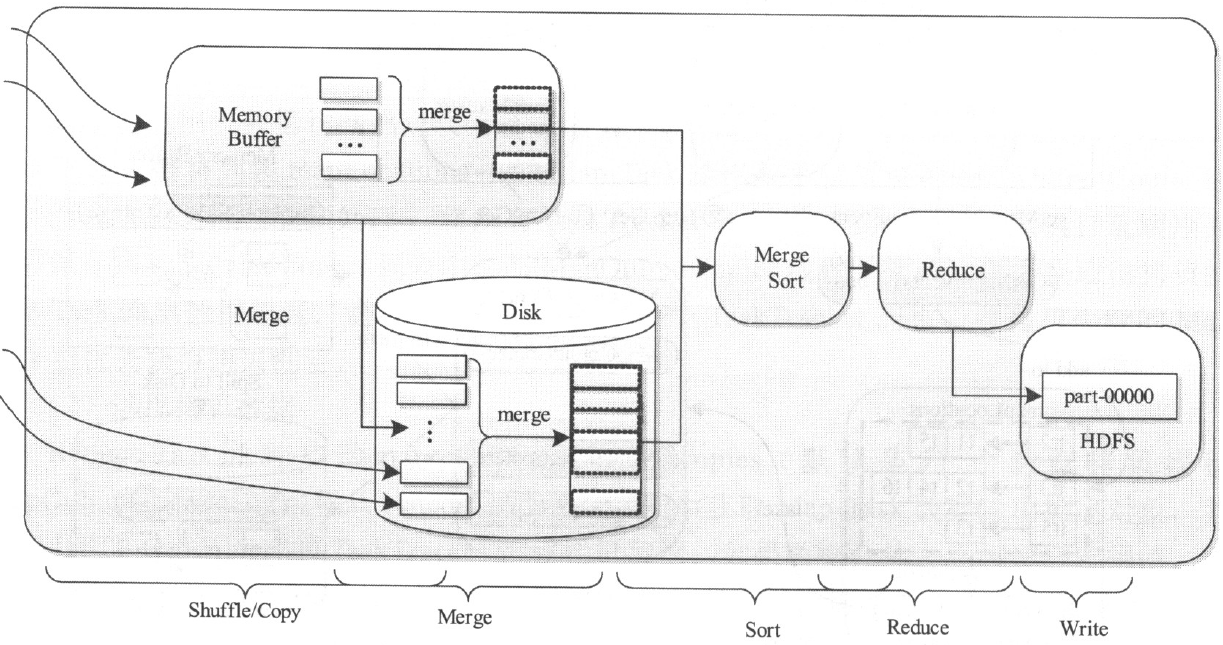
Combine过程:

[MapOutputBuffer](eclipse-javadoc:%E2%98%82=hadoop-2.6.0-src/hadoop-mapreduce-project%5C/hadoop-mapreduce-client%5C/hadoop-mapreduce-client-core%5C/src%5C/main%5C/java%3Corg.apache.hadoop.mapred%7BMapTask.java%E2%98%83MapTask%E2%98%83MapOutputBuffer).flush()

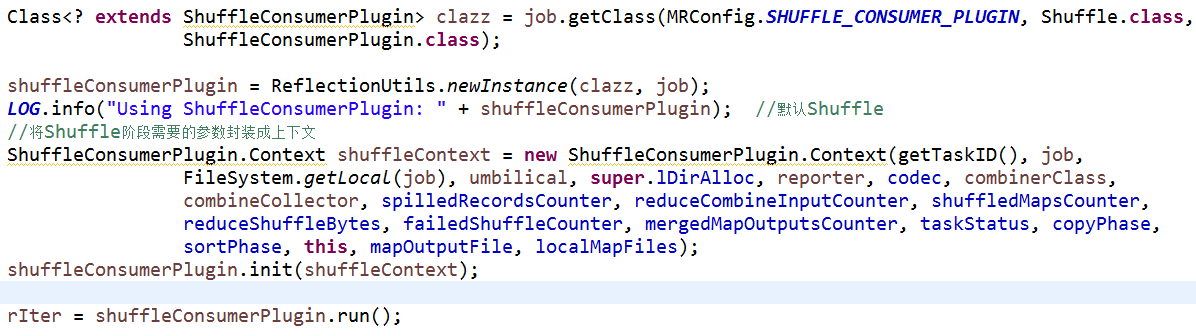
mergeParts()



## Reduce task计算流程



## ReduceTask的Shuffle过程分析(Shullfe是MR的核心)



* Shuffle.run()



### EventFetcher 核心代码:

run()

**int** numNewMaps = getMapCompletionEvents();

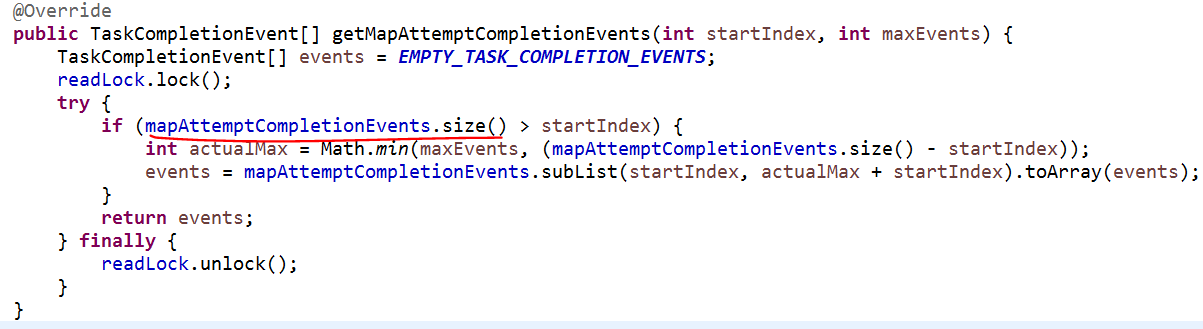
* getMapCompletionEvents()

MapTaskCompletionEventsUpdate update = umbilical.getMapCompletionEvents(

(org.apache.hadoop.mapred.JobID) reduce.getJobID(), fromEventIdx, maxEventsToFetch,

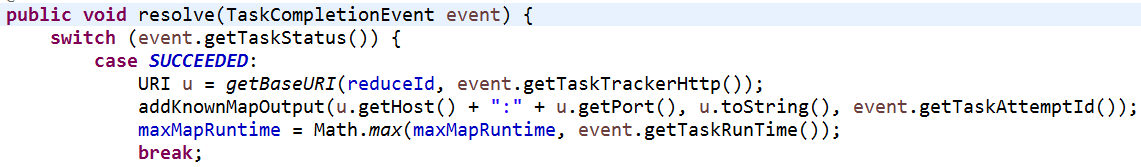
(org.apache.hadoop.mapred.TaskAttemptID) reduce);

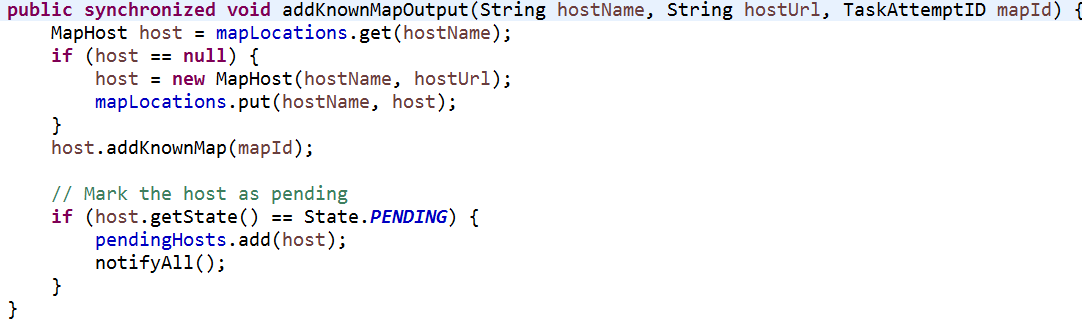
* JobImpl：拉取完成的MapTask



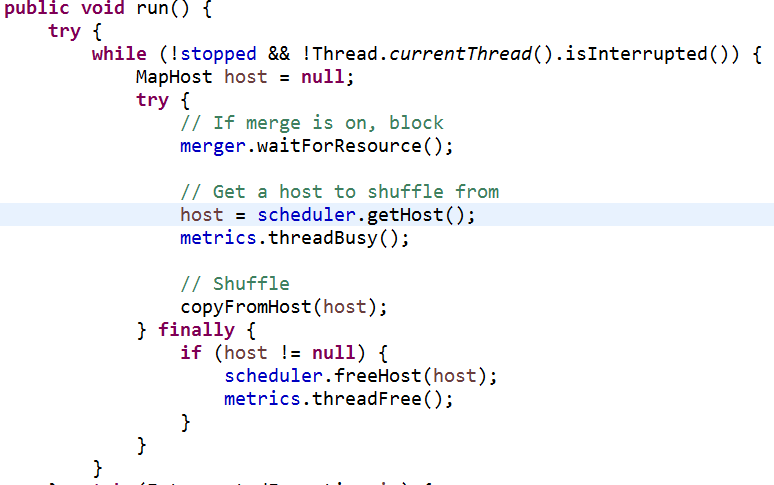
拉取完成后放入mapLocations映射中

* ShuffleSchedulerImpl

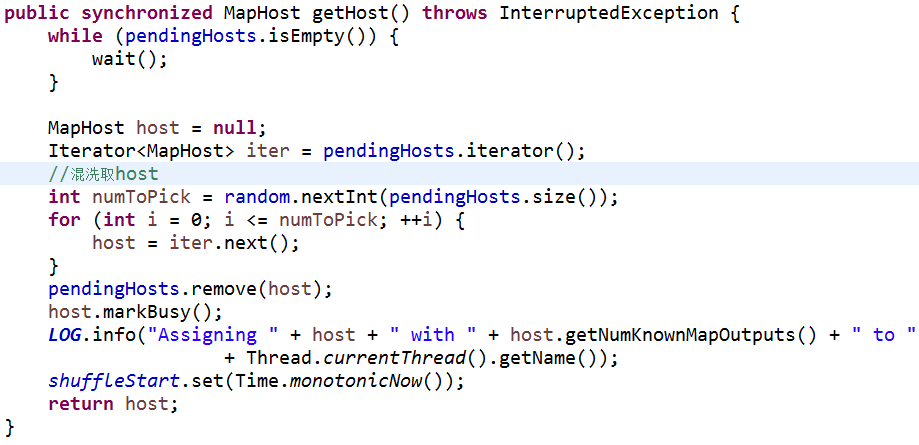




### Fetcher

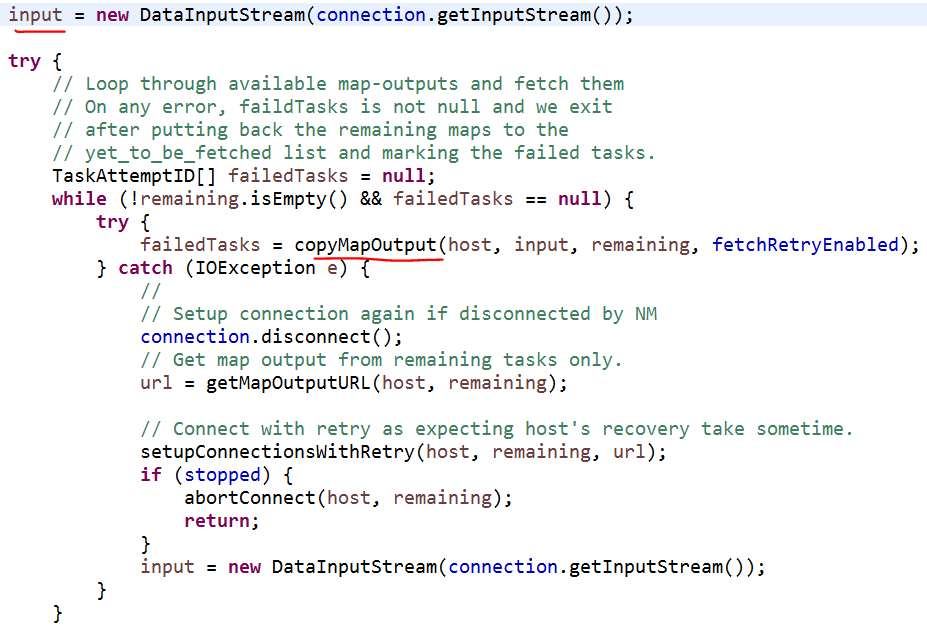


Scheduler.getHost():防止出现网络热点,通过对host进行’混洗’操作以打乱copy顺序





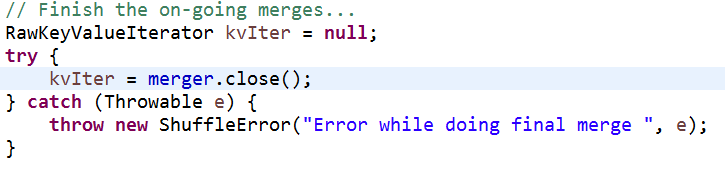
...



copyMapOutput()核心代码:



经上使用InMemoryOutPut和OnDiskMapOutput后,已经将MapTask的输出拉到此ReduceTask上,最后Shuffle.java中执行



会将内存和磁盘上的做最终的sort(Map端已经做了,reduce只需做一次归并排序即可)和merge并赋值给RawKeyValueIterator

将其返回给ReduceTask,将RawKeyValueIterator封装给Reduce的参数上下文Context,接下来就是执行用户自己实现的reduce类了,会不断的调用context.getNextValue/key来去RawKeyValueIterator中的<key,value>

kvIter = merger.close();



