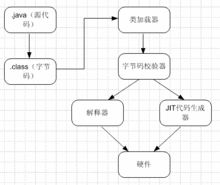
JVM是java的核心和基础，在java编译器和os平台之间的虚拟处理器。它是一种基于下层的操作系统和硬件平台并利用软件方法来实现的抽象的计算机，可以在上面执行java的字节码程序。

[](http://baike.baidu.com/pic/JVM/2902369/0/0b7b02087bf40ad1bb99b82a562c11dfa8eccefe?fr=lemma&ct=single)JVM运行原理

java编译器只需面向JVM，生成JVM能理解的代码或字节码文件。Java源文件经编译器，编译成字节码程序，通过JVM将每一条指令翻译成不同平台机器码，通过特定平台运行。

JVM是Java程序运行的容器,但是他同时也是操作系统的一个进程,因此他也有他自己的运行的生命周期,也有自己的代码和数据空间。

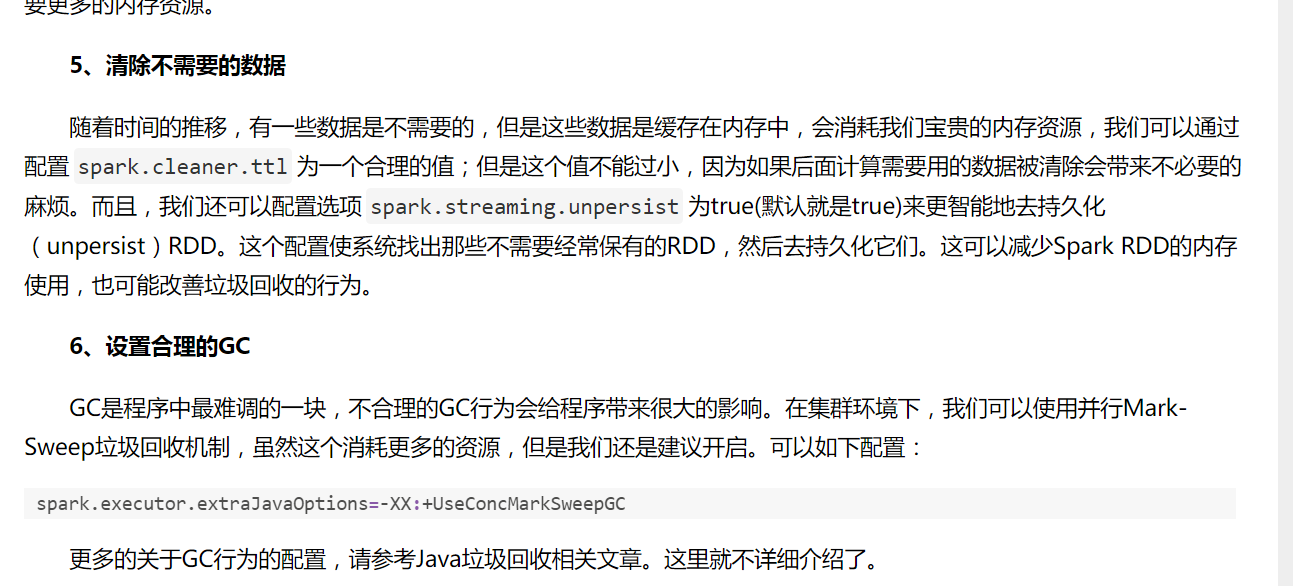
JVM在整个jdk中处于最底层,负责与操作系统的交互,用来屏蔽操作系统环境,提供一个完整的Java运行环境,因此也就虚拟计算机.操作系统装入JVM是通过jdk中Java.exe来完成,通过下面4步来完成JVM环境。

spark.driver.extraJavaOptions：A string of extra JVM options to pass to the driver. For instance, GC settings or other logging.   
Note: In client mode, this config must not be set through the SparkConf directly in your application, because the driver JVM has already started at that point. Instead, please set this through the --driver-java-options command line option or in your default properties file.

spark.executor.extraJavaOptions

spark.memory.fraction 0.75

spark.memory.storageFraction 0.5



jstat –gcutil

<http://www.cnblogs.com/kabi/p/6429608.html>

http://blog.csdn.net/jiafu1115/article/details/7024323

其中4846是某一进程ID。

jmap -heap 4846

jstat -gcutil -h5 4846 4s 100

Jobhistoryserver开启日志：

+ export 'HADOOP\_JOB\_HISTORYSERVER\_OPTS=-Xms351272960 -Xmx351272960 -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=70 -XX:+CMSParallelRemarkEnabled -Dhadoop.event.appender=,EventCatcher -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/tmp/yarn\_yarn-JOBHISTORY-528e9b4460184b43e22e98262d3f169d\_pid18620.hprof -XX:OnOutOfMemoryError=/opt/cm-5.10.1/lib64/cmf/service/common/killparent.sh'

+ HADOOP\_JOB\_HISTORYSERVER\_OPTS='-Xms351272960 -Xmx351272960 -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=70 -XX:+CMSParallelRemarkEnabled -Dhadoop.event.appender=,EventCatcher -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/tmp/yarn\_yarn-JOBHISTORY-528e9b4460184b43e22e98262d3f169d\_pid18620.hprof -XX:OnOutOfMemoryError=/opt/cm-5.10.1/lib64/cmf/service/common/killparent.sh'

+ make\_scripts\_executable

#### **glom**(): [RDD](http://spark.apache.org/docs/1.6.0/api/scala/org/apache/spark/rdd/RDD.html)[Array[T]]:将每个分区的元素组成一个数组，然后再用每个数组组成一个RDD

#### Return an RDD created by coalescing all elements within each partition into an array.

读取文件时，可在页面看到record数，通过：a = sc.parallelize([1,2,3,4],3)生成的RDD，计算时，页面看不到record数，但可通过：k.glom().map(lambda x: len(x)).collect() 来查看分区记录数。

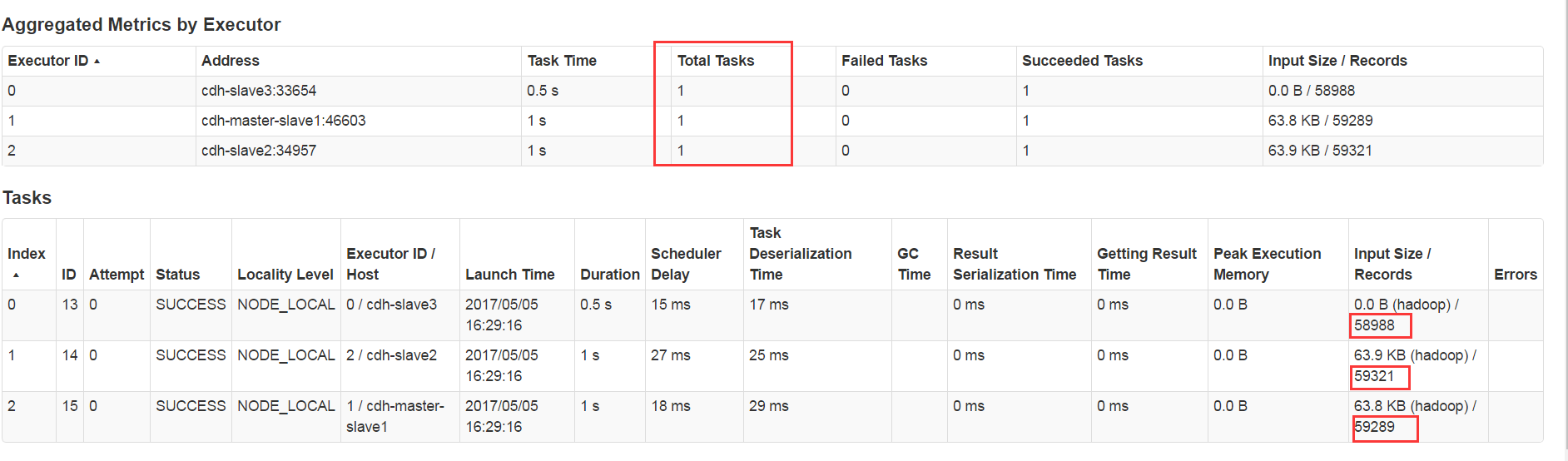
h = sc.textFile('/data/visits2.txt',3)

k = h.map(lambda x:(x + g))

k.glom().map(lambda x: len(x)).collect()

结果：[58988, 59321, 59289]

如下图就是每个task的输入record，对应[58988, 59321, 59289]



h = sc.textFile('/data/inputfile.txt',5) #由于文件太小，就4行，因此这里有6个分区，由图可知，空分区也会有task，也参与计算！

h.glom().collect()

结果如下：

[[u'hadoop spark hdfs'],

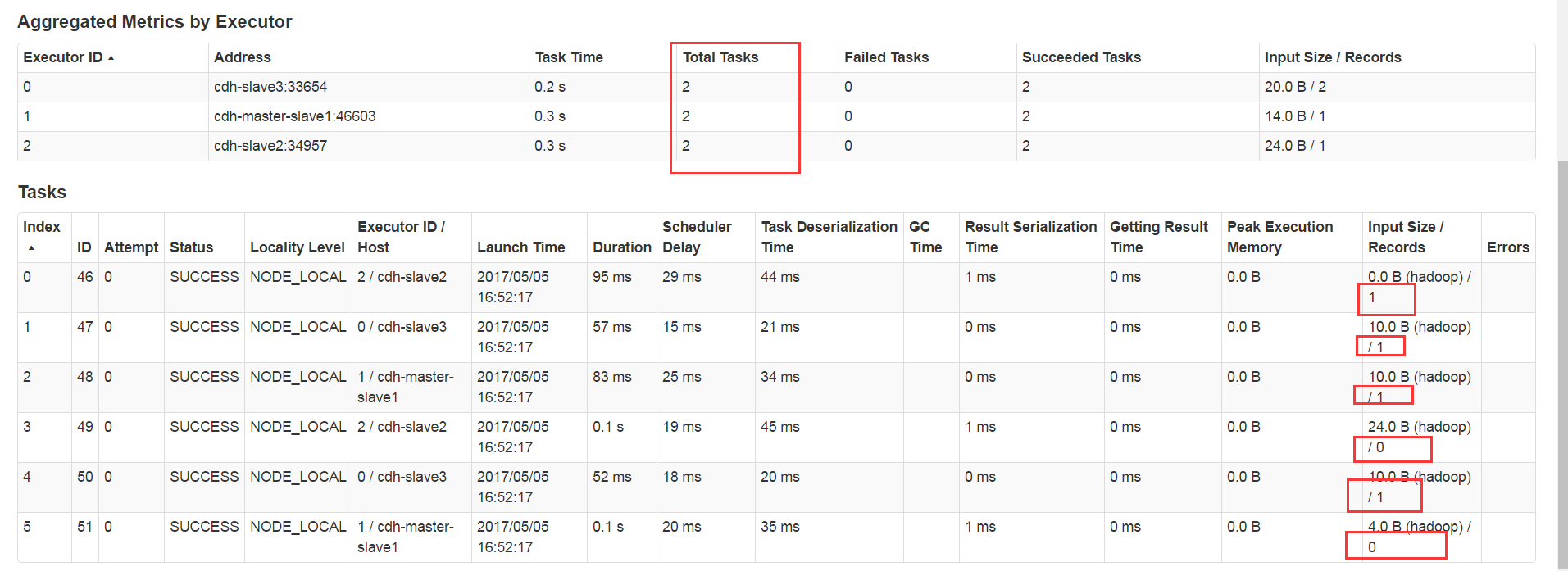
[u'hive hbase'],

[u'hdfs hive spark'],

[],

[u'mr sqoop'],

[]]



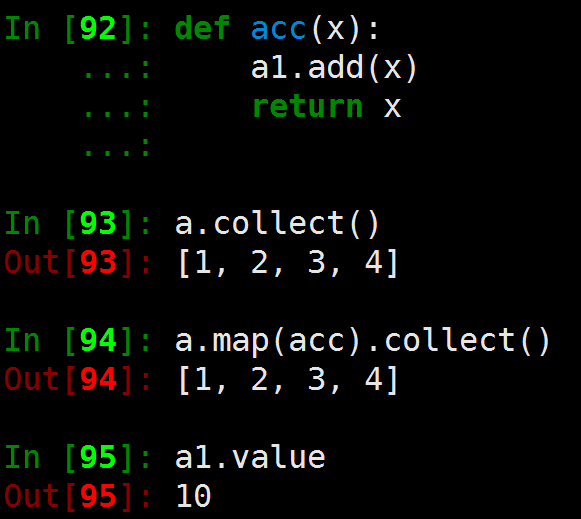
累加器：

a1= sc.accumulator(0) #Accumulator<id=0, value=0>

a1.value #0

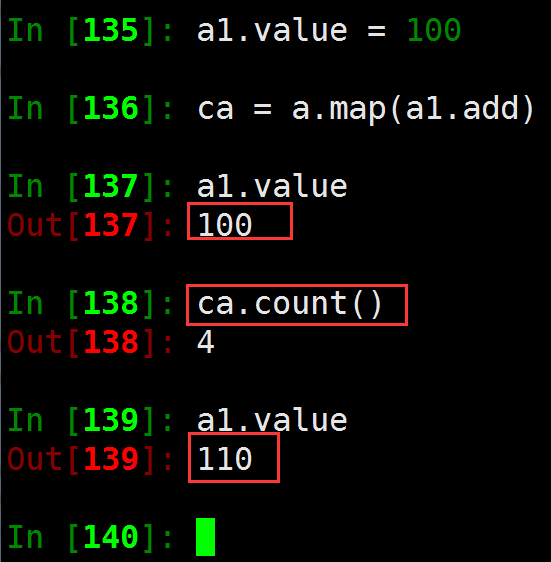
a1.add(2) #2

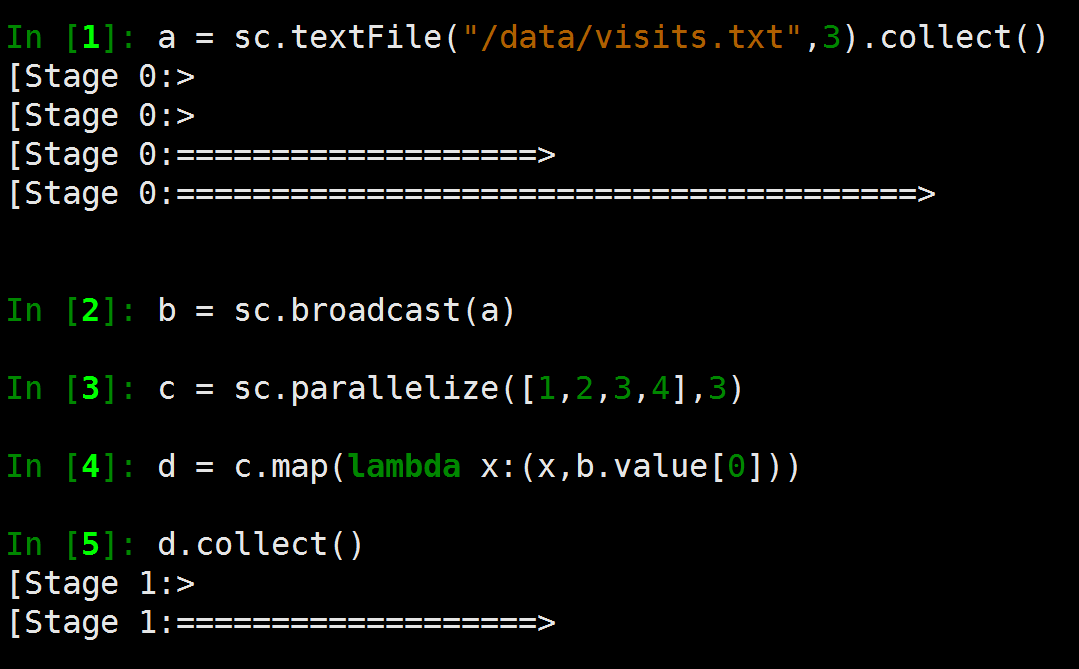
a1.add(-2) #0



a1.value = 100 #可对累加器赋值！

算子中用到累加器，也是lazy的，当执行action时，才计算，如下图：





d.collect()

d.clooect()

