Spark—core阶段问题

1.LRU算法

2.spark速度快的原因

logistic regression 迭代场景

spark速度快的原因：

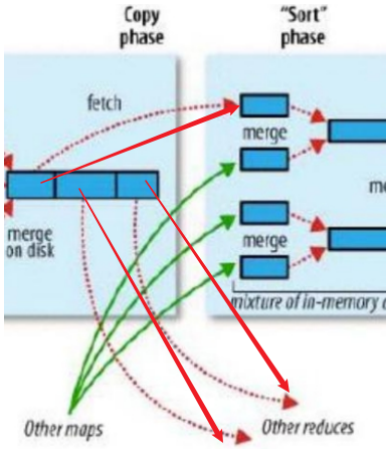
1. 基于内存，spark中间结果不落盘/hadoop频繁落盘
2. DAG

不适合迭代

3.shuffle

洗牌

数据重新分布



4.RDD总结

看源码

特性及体现

5.foreach对比

Scala中

- 遍历

spark中

- 算子
- 在遍历的基础上又`launch a job`的作用

6.分布

泊松

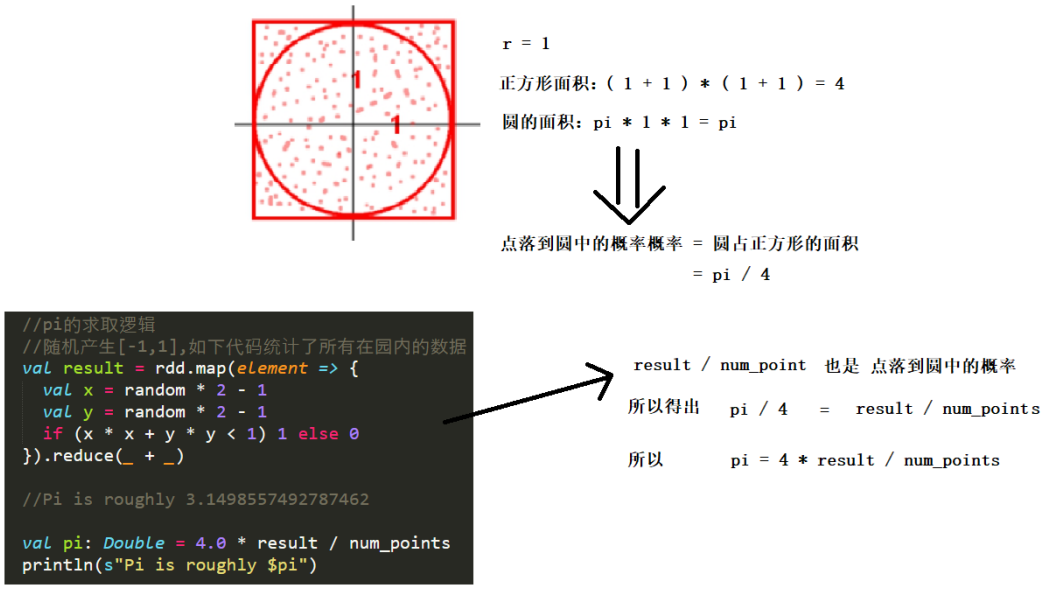
伯努利

7.combiner

MR的,作用于累加时的数据合并

8.pi的spark例子中的4

图解:

- 

9.查看spark启动的端口

sbin -> vim start-master.sh ->

master的端口是7077

WEBUI的端口是8080

10.为什么textFile用string去接收

因为spark中textFile读文件的函数沿用的MR，MR读文件是行读取器，一行一行读出来，只能string去接收。

11.spark中资源

C & M

C：cores

M：mem

master掌握资源信息

12.yarn运行模式spark

启动zookeeper：zkServer.sh start

查看状态：zkServer.sh status

启动hadoop：start-all.sh

启动日志：mr-jobhistory-daemon.sh start historyserver

启动spark：sbin下./start-all.sh

D-node01:9870

D-node01:8088

13.Driver

jps中没有，体现的是SparkSubmit

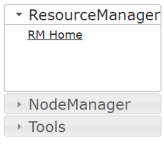
作用：

1. 请求资源
2. 任务分发
3. 过程监控
4. 结果收集

14.Logs出现Fails且无法访问

原因：日志没有收集

Failed redirect for container_e57_1615260920620_0001_01_000001



Failed while trying to construct the redirect url to the log server. Log Server url may not be configured

Local Logs:

java.lang.Exception: Unknown container. Container either has not started or has already completed or doesn't belong to this node at all.

解决：

配置文件：yarn-site.xml和mapred-site.xml中的节点配置是哪个在哪个节点中启动日志命令。

`mr-jobhistory-daemon.sh start historyserver`

如下面都是node03，保持一致。

yarn-site.xml中

```
<property>
       <name>yarn.log-aggregation-enable</name>
       <value>true</value>
</property>
<property>
        <name>yarn.log.server.url</name>
        <value>http://node03:19888/jobhistory/logs</value>
</property>
<property>
       <name>yarn.nodemanager.remote-app-log-dir</name>
       <value>/tmp/logs</value>
</property>
```

mapred-site.xml中

```
<property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <property>
        <name>mapreduce.jobhistory.address</name>
        <value>node03:10020</value>
    </property>
    <property>
        <name>mapreduce.jobhistory.webapp.address</name>
        <value>node03:19888</value>
    </property>
    <property>
        <name>mapreduce.jobhistory.done-dir</name>
        <value>/history/done</value>
    </property>
<!-- 正在运行的任务信息临时目录 -->
    <property>
        <name>mapreduce.jobhistory.intermediate.done-dir</name>
        <value>/history/done/done_intermediate</value>
    </property>
```

15.job-stag-task

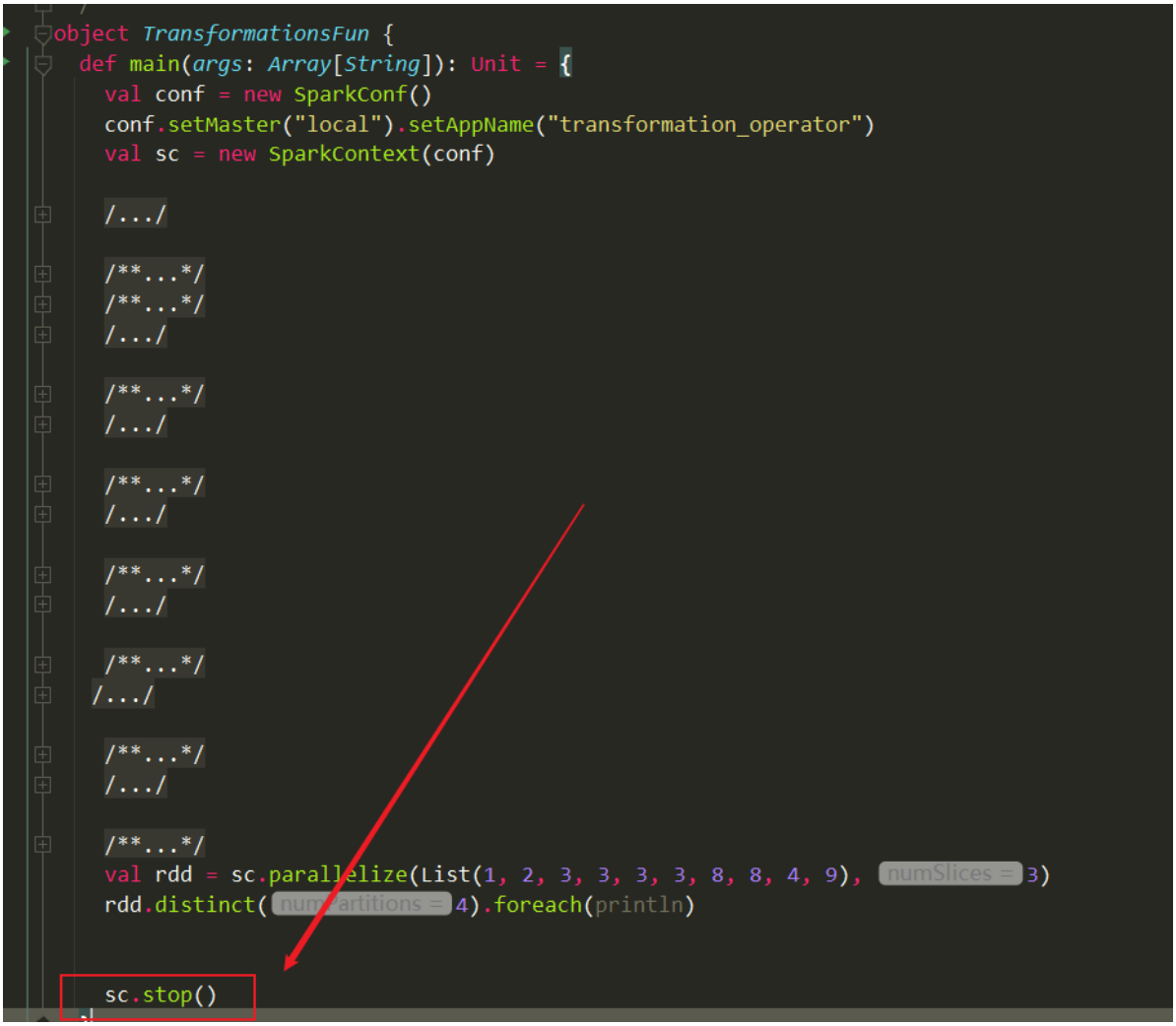
application>job（串）>stag（可并可串）>task（并，分布式计算）

16.出现空指针异常

报错：

```
Exception in thread "main" java.lang.NullPointerException Create breakpoint
    at org.apache.spark.ShuffleDependency$.$lessinit$greater$default$3(Dependency.scala:73)
    at org.apache.spark.rdd.SubtractedRDD.rddDependency$1(SubtractedRDD.scala:65)
    at org.apache.spark.rdd.SubtractedRDD.getDependencies(SubtractedRDD.scala:68)
    at org.apache.spark.rdd.RDD.$anonfun$dependencies$2(RDD.scala:256)
    at scala.Option.getOrElse(Option.scala:189)
    at org.apache.spark.rdd.RDD.dependencies(RDD.scala:252)
    at org.apache.spark.rdd.SubtractedRDD.$anonfun$getPartitions$2(SubtractedRDD.scala:76)
    at scala.collection.TraversableLike.$anonfun$map$1(TraversableLike.scala:273)
    at scala.collection.immutable.List.foreach(List.scala:392)
    at scala.collection.TraversableLike.map(TraversableLike.scala:273)
    at scala.collection.TraversableLike.map$(TraversableLike.scala:266)
    at scala.collection.immutable.List.map(List.scala:298)
    at org.apache.spark.rdd.SubtractedRDD.$anonfun$getPartitions$1(SubtractedRDD.scala:75)
    at scala.collection.immutable.Range.foreach$mVc$sp(Range.scala:158)
    at org.apache.spark.rdd.SubtractedRDD.getPartitions(SubtractedRDD.scala:73)
    at org.apache.spark.rdd.RDD.$anonfun$partitions$2(RDD.scala:273)
    at scala.Option.getOrElse(Option.scala:189)
    at org.apache.spark.rdd.RDD.partitions(RDD.scala:269)
    at org.apache.spark.rdd.MapPartitionsRDD.getPartitions(MapPartitionsRDD.scala:49)
    at org.apache.spark.rdd.RDD.$anonfun$partitions$2(RDD.scala:273)
    at scala.Option.getOrElse(Option.scala:189)
    at org.apache.spark.rdd.RDD.partitions(RDD.scala:269)
    at org.apache.spark.SparkContext.runJob(SparkContext.scala:2126)
```

解决：



17.spark中distinct是如何实现的?

A1 总述：去重

A2 思路：map -> resuceByKey -> map

A3 源码：

3.1 有参：

```
 /**
   * Return a new RDD containing the distinct elements in this RDD.
   */
  def distinct(numPartitions: Int)(implicit ord: Ordering[T] = null): RDD[T] =
withScope {
    map(x => (x, null)).reduceByKey((x, y) => x, numPartitions).map(_._1)
  }
//numPartitions:分区数
```

3.2 无参：

```
/**
   * Return a new RDD containing the distinct elements in this RDD.
   */
  def distinct(): RDD[T] = withScope {
    distinct(partitions.length)
  }
//partitions.length:分区数
```

3.3 解释

我们从源码中可以看到，distinct去重主要实现逻辑是

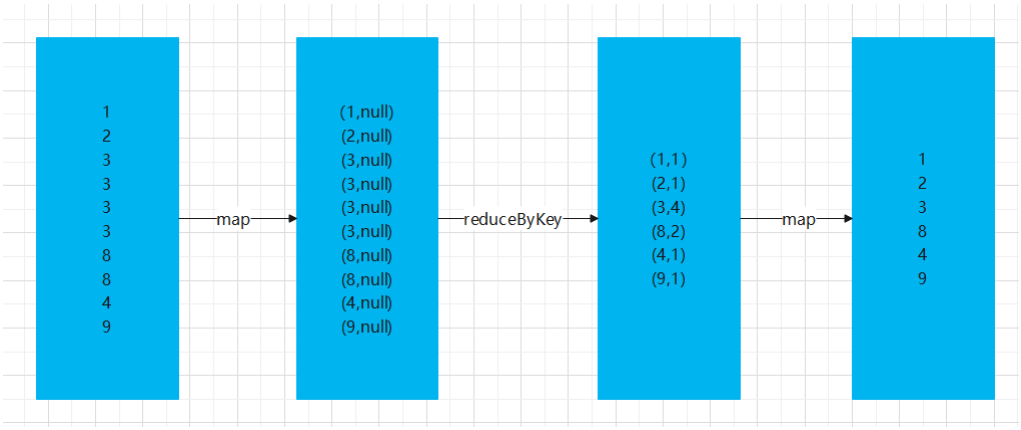
```
map(x => (x, null)).reduceByKey((x, y) => x, numPartitions).map(_._1)
```

这个过程是，先通过map映射每个元素和null，然后通过key（此时是元素）统计{reduceByKey就是对元素为KV对的RDD中Key相同的元素的Value进行binary_function的reduce操作，因此，Key相同的多个元素的值被reduce为一个值，然后与原RDD中的Key组成一个新的KV对。}，最后再同过map把去重后的元素挑出来。

A4 测试代码

```
import org.apache.spark.{SparkConf, SparkContext}
object TransformationsFun {
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf()
    conf.setMaster("local").setAppName("transformation_operator")
    val sc = new SparkContext(conf)
    //这里的3是初设定的partition数
    val rdd = sc.parallelize(List(1, 2, 3, 3, 3, 3, 8, 8, 4, 9), 3)
    //因为distinct实现用reduceByKey故其可以重设定partition数,这里设定4
    rdd.distinct(4).foreach(println)
    //这里执行时，每次结果不同，分区在4以内，每个分区处理的元素也不定
    sc.stop()
  }
}
```

图解：



解释：这里仅供理解，在实际运行中，分区会随机使用以及每个分区处理的元素也随机，所以每次运行结果会不同。

使用map算子把元素转为一个带有null的元组；使用reducebykey对具有相同key的元素进行统计；之后再使用map算子，取得元组中的单词元素，实现去重的效果。

18.reduceByKey

可以改分区数量

因为会重新分布数据

19.整理算子宽窄

思维导图中

A1 如何判断算子宽窄?

一种方法:

看参数是否可以改变分区数

可以看源码中参数是否有与分区相关的，比如numPartitions

A2 例子:

sortBy和map比较:

- sortBy最后有numPartitions，添加不报错

```
result.sortBy(_._2, ascending = false, numPartitions = 3).foreach(println)
```

- map后加上数字（表示分区数的）会报错

```
private val pairs: RDD[(String, Int)] = word.map(x => (x, 1),3)
```

A3 解释

看sortBy和map源码:

- sortBy中第三个参数是numPartitions
    - ```
      def sortBy[K](
          f: (T) => K,
          ascending: Boolean = true,
          numPartitions: Int = this.partitions.length)//这里表示分区数
      (implicit ord: Ordering[K], ctag: ClassTag[K]): RDD[T] = withScope {
          this.keyBy[K](f)
          .sortByKey(ascending, numPartitions)
          .values
      }
      ```

- map中则没有
    - ```
      def map[U: ClassTag](f: T => U): RDD[U] = withScope {
          val cleanF = sc.clean(f)
          new MapPartitionsRDD[U, T](this, (context, pid, iter) =>
      iter.map(cleanF))
      }
      ```

A4 问题

4.1 判断flatmap、reduceByKey、GroupByKey算子的宽窄。

思路：

1. 添加分区参数看是否报错
2. 看源码参数是否有与分区相关的

flatmap（窄）：

- ```
  def flatMap[U: ClassTag](f: T => TraversableOnce[U]): RDD[U] = withScope {
      val cleanF = sc.clean(f)
      new MapPartitionsRDD[U, T](this, (context, pid, iter) =>
  iter.flatMap(cleanF))
  }
  ```
- 没有，是窄依赖

reduceByKey（宽）：

- ```
  def reduceByKey(partitioner: Partitioner, func: (V, V) => V): RDD[(K, V)] =
  self.withScope {
      combineByKeyWithClassTag[V]((v: V) => v, func, func, partitioner)
  }

  def reduceByKey(func: (V, V) => V, numPartitions: Int): RDD[(K, V)] =
  self.withScope {
      reduceByKey(new HashPartitioner(numPartitions), func)
  }
  ```
- 我们可以看到参数partitioner、numPartitions
- partitioner底层包含numPartitions
    - ```
      abstract class Partitioner extends Serializable {
        def numPartitions: Int
        def getPartition(key: Any): Int
      }
      ```

GroupByKey（宽）：

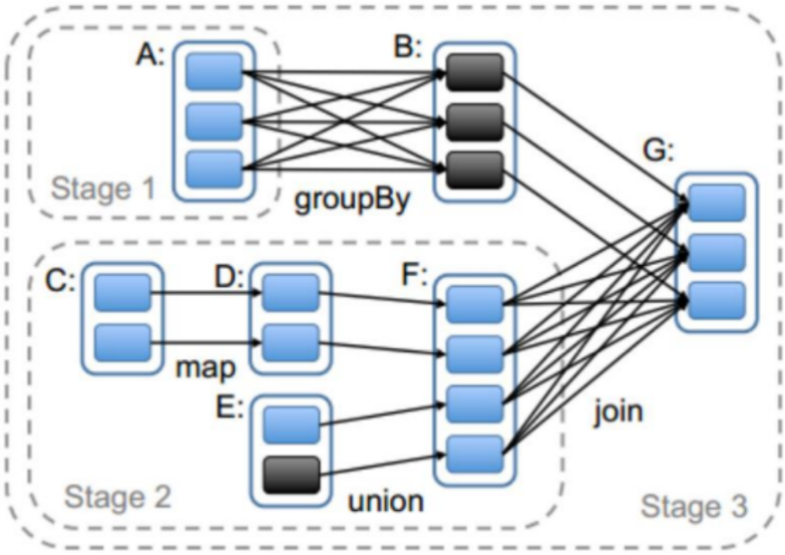
- ```
  def groupByKey(partitioner: Partitioner): RDD[(K, Iterable[V])] =
  self.withScope {
      // groupByKey shouldn't use map side combine because map side combine does
  not
      // reduce the amount of data shuffled and requires all map side data be
  inserted
      // into a hash table, leading to more objects in the old gen.
      val createCombiner = (v: V) => CompactBuffer(v)
      val mergeValue = (buf: CompactBuffer[V], v: V) => buf += v
      val mergeCombiners = (c1: CompactBuffer[V], c2: CompactBuffer[V]) => c1
  ++= c2
      val bufs = combineByKeyWithClassTag[CompactBuffer[V]](
          createCombiner, mergeValue, mergeCombiners, partitioner,
  mapSideCombine = false)
      bufs.asInstanceOf[RDD[(K, Iterable[V])]]
  }
  ```
- 我们还是可以看见partitioner的身影

4.2 宽窄依赖影响的是什么?

影响的是stage。

因为stage的切割依据是RDD之间的宽窄依赖。

stage的切割规则：从后往前，遇到宽依赖就切割stage。



从图中可以看出

1. stage中引入DAG（有向无环图，指定执行顺序ABCDEFG）
2. A->B是宽依赖，F->G是宽依赖，stage的切割从A和F
3. join有宽有窄
4. stage中串并同存在

20.stage切割规则

从后往前，遇到宽依赖就切割stage。

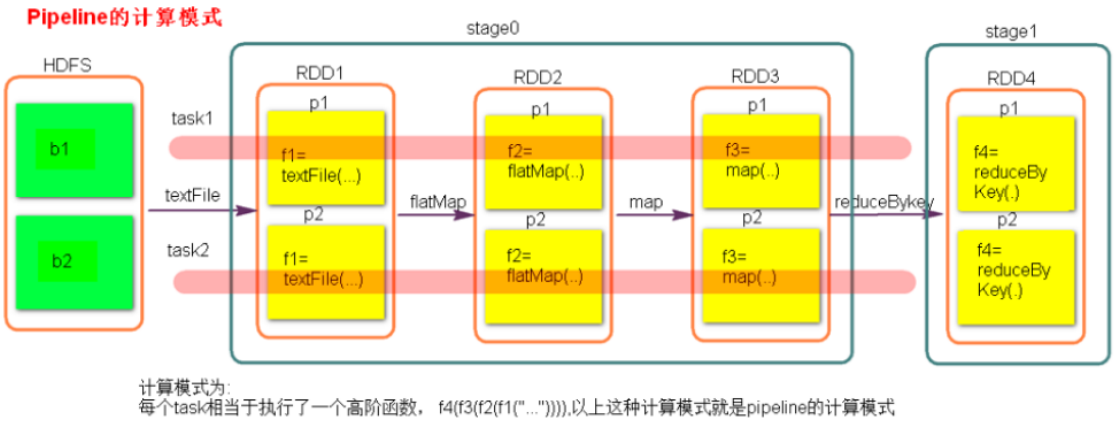
21.编码习惯

自己在测试时，最好在foreach前加个collect

rdd.collect().foreach(println)

22.stage的管道

stage的计算原理



关键：数据落地

23.groupByKey 源码@note

```scala
/**
 * Group the values for each key in the RDD into a single sequence. Allows controlling the
 * partitioning of the resulting key-value pair RDD by passing a Partitioner.
 * The ordering of elements within each group is not guaranteed, and may even differ
 * each time the resulting RDD is evaluated.
 *
 * @note This operation may be very expensive. If you are grouping in order to perform an
 * aggregation (such as a sum or average) over each key, using `PairRDDFunctions.aggregateByKey`
 * or `PairRDDFunctions.reduceByKey` will provide much better performance.
 *
 * @note As currently implemented, groupByKey must be able to hold all the key-value pairs for any
 * key in memory. If a key has too many values, it can result in an `OutOfMemoryError`.
 */
def groupByKey(partitioner: Partitioner): RDD[(K, Iterable[V])] = self.withScope {
  // groupByKey shouldn't use map side combine because map side combine does not
  // reduce the amount of data shuffled and requires all map side data be inserted
  // into a hash table, leading to more objects in the old gen.
  val createCombiner = (v: V) => CompactBuffer(v)
  val mergeValue = (buf: CompactBuffer[V], v: V) => buf += v
  val mergeCombiners = (c1: CompactBuffer[V], c2: CompactBuffer[V]) => c1 ++= c2
  val bufs = combineByKeyWithClassTag[CompactBuffer[V]](
    createCombiner, mergeValue, mergeCombiners, partitioner, mapSideCombine = false)
  bufs.asInstanceOf[RDD[(K, Iterable[V])]]
}
```

24.KryoSerializer

spark的序列化，相较于Java的序列化更迅速轻便。

序列化的作用主要是利用时间换空间

- *分发给Executor上的Task*
- *需要缓存的RDD（前提是使用序列化方式缓存）*
- *广播变量*
- *Shuffle过程中的数据缓存*
- *使用receiver方式接收的流数据缓存*
- *算子函数中使用的外部变量*

上面的六种数据，通过Java序列化（默认的序列化方式）形成一个二进制字节数组，大大减少了数据在内存、硬盘中占用的空间，减少了网络数据传输的开销，并且可以精确的推测内存使用情况，降低GC频率。

25.supervise

失败后是否重启Driver，仅限于Spark alone或者Mesos模式

26.跟cores相关的参数

27.AM和EL

28.yarn有几种调度器?

3种

https://blog.csdn.net/Jin_Lemon/article/details/114765523

29.磁盘调度算法

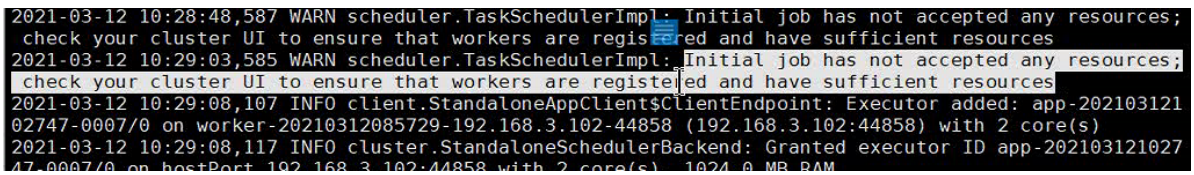
先进先出

最近寻址

电梯

30.没有资源了

报错:



31.写hadoop地址

要区分active和备，所以最好写集群名，不写下面的具体

sc.textFile("hdfs://node1:9000/spark/test/wc.txt").flatMap(*.split(" ")).map((,1)).reduceByKey(+).foreach(println)

写

sc.textFile("hdfs://bdp/spark/test/wc.txt").flatMap(*.split(" ")).map((,1)).reduceByKey(+).foreach(println)

位置：

```
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>dfs.nameservices</name>
<value>bdp</value>
</property>
<property>
<name>dfs.ha.namenodes.bdp</name>
<value>nn1,nn2</value>
</property>
<property>
<name>dfs.namenode.rpc-address.bdp.nn1</name>
<value>node001:8020</value>
</property>
<property>
<name>dfs.namenode.rpc-address.bdp.nn2</name>
<value>node002:8020</value>
</property>
"/opt/hadoop-3.1.2/etc/hadoop/hdfs-site.xml" 73L, 2090C
```

32.spark相关配置

优先级从上到下

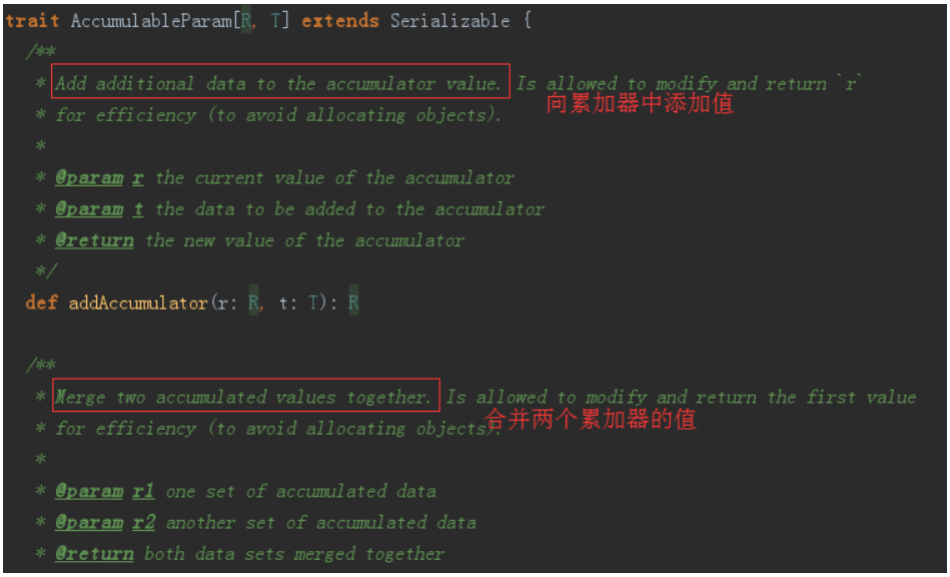
1. 代码（写死）
2. 命令行（最好，灵活）
3. 文件（默认）

33.日志压缩

lz4格式

34.spark怎么实现自定义累加器?

Spark自定义累加器需要实现 **AccumulatorParam**



35.溢写

https://blog.csdn.net/godlovedaniel/article/details/113979588

36.spark中RPC

1.6 前 akka

1.6 后 akka+netty

37.master源码

1.角色

Rpc声明周期

2.过程（生命周期）

构造

onstart

reserive

选举leader

完成恢复

删除leader

前四个需要自己理解，执行过程中，无非就是 根据mastar、worker、driver、executor、app等不同的状态有不同的应对手段。

onstop

3.关键：

schedule()方法

38.worker源码

worker的注册核心在于资源的汇报

39.遇到错误

看日志（xxx.out）