MapReduce 基础入门

目录

1、MapReduce 入门

1.1、什么是 MapReduce

首先让我们来重温一下 hadoop 的四大组件：
HDFS：分布式存储系统
MapReduce：分布式计算系统
YARN：hadoop 的资源调度系统
Common：以上三大组件的底层支撑组件，主要提供基础工具包和 RPC 框架等

MapReduce 是一个分布式运算程序的编程框架，是用户开发“基于 Hadoop 的数据分析应用”的核心框架

MapReduce **核心功能**是**将用户编写的业务逻辑代码和自带默认组件整合成一个完整的分布式运算程序，并发运行在一个 Hadoop 集群上**

1.2、为什么需要 MapReduce

为什么需要 MapReduce？

1、海量数据在单机上处理因为硬件资源限制，无法胜任

2、而一旦将单机版程序扩展到集群来分布式运行，将极大增加程序的复杂度和开发难度
3、引入 MapReduce 框架后，开发人员可以将绝大部分工作集中在业务逻辑的开发上，而将分布式计算中的复杂性交由框架来处理

设想一个海量数据场景下的数据计算需求：

单机版：磁盘受限，内存受限，计算能力受限
分布式版： 1、 数据存储的问题，hadoop 提供了 hdfs 解决了数据存储这个问题 2、 运算逻辑至少要分为两个阶段，先并发计算（map），然后汇总（reduce）结果 3、 这两个阶段的计算如何启动？如何协调？ 4、 运算程序到底怎么执行？数据找程序还是程序找数据？ 5、 如何分配两个阶段的多个运算任务？ 6、 如何管理任务的执行过程中间状态，如何容错？ 7、 如何监控？ 8、 出错如何处理？抛异常？重试？

可见在程序由单机版扩成分布式版时，会引入大量的复杂工作。为了提高开发效率，可以将分布式程序中的公共功能封装成框架，让开发人员可以将精力集中于业务逻辑。

Hadoop 当中的 MapReduce 就是这样的一个分布式程序运算框架，它把大量分布式程序都会涉及的到的内容都封装进了，让用户只用专注自己的业务逻辑代码的开发。它对应以上问题的整体结构如下：

MRAppMaster：MapReduce Application Master，分配任务，协调任务的运行
MapTask：阶段并发任，负责 mapper 阶段的任务处理　　YARNChild
ReduceTask：阶段汇总任务，负责 reducer 阶段的任务处理　YARNChild

1.3、MapReduce 程序运行演示

在 MapReduce 组件里，官方给我们提供了一些样例程序，其中非常有名的就是 wordcount 和 pi 程序。这些 MapReduce 程序的代码都在 hadoop-mapreduce-examples-2.6.4.jar 包里，这个 jar 包在 hadoop 安装目录下的/share/hadoop/mapreduce/目录里

下面我们使用 hadoop 命令来试跑例子程序，看看运行效果
先看 MapReduce 程序求 pi 的程序：

命令： [hadoop@hadoop03 mapreduce]$ pwd /home/hadoop/apps/hadoop-2.6.4/share/hadoop/mapreduce [hadoop@hadoop03 mapreduce]$ **hadoop jar hadoop-mapreduce-examples-2.6.4.jar pi 5 5**
执行过程和结果展示：

```
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=590
        File Output Format Counters
                Bytes Written=97
Job Finished in 50.158 seconds
Estimated value of Pi is 3.68000000000000000000
[hadoop@hadoop03 mapreduce]$
```

再来看 wordcount 程序：

首先我们，准备一份数据放到 HDFS 上：

```
[hadoop@hadoop03 hadoop-2.6.4]$ hadoop fs -put README.txt /
[hadoop@hadoop03 hadoop-2.6.4]$ hadoop fs -ls /
Found 3 items
-rw-r--r--   2 hadoop supergroup       1366 2017-02-11 22:21 /README.txt
drwx-wx-wx   - hadoop supergroup          0 2017-02-11 19:16 /tmp
drwxr-xr-x   - hadoop supergroup          0 2017-02-11 19:16 /user
```

命令：

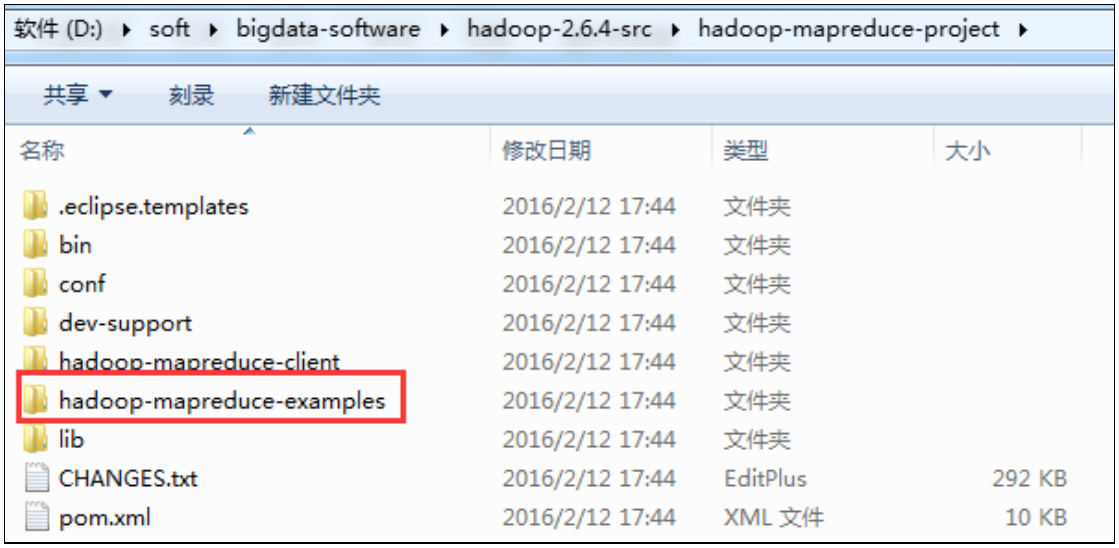
[hadoop@hadoop03 mapreduce]$ **hadoop jar hadoop-mapreduce-examples-2.6.4.jar wordcount /README.txt /wordcount**

查看结果：

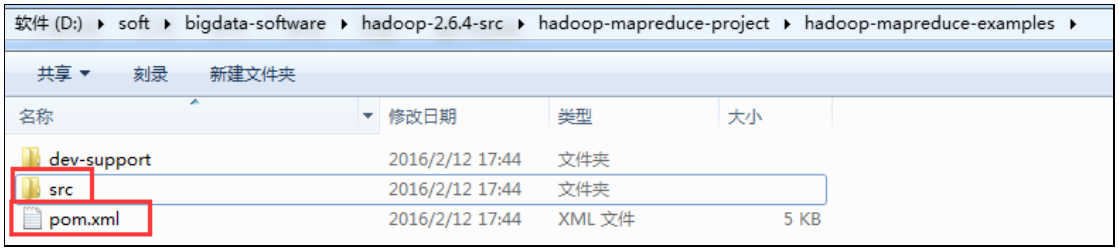
```
[hadoop@hadoop03 mapreduce]$ hadoop fs -cat /wordcount/part-r-00000
(BIS),  1
(ECCN)  1
(TSU)   1
(see    1
5D002.C.1,      1
740.13) 1
<http://www.wassenaar.org/>     1
Administration  1
Apache  1
BEFORE  1
BIS     1
```

那除了这两个程序以外，还有没有官方提供的其他程序呢，还有就是它们的源码在哪里呢？

我们打开 mapreduce 的源码工程，里面有一个 hadoop-mapreduce-project 项目：



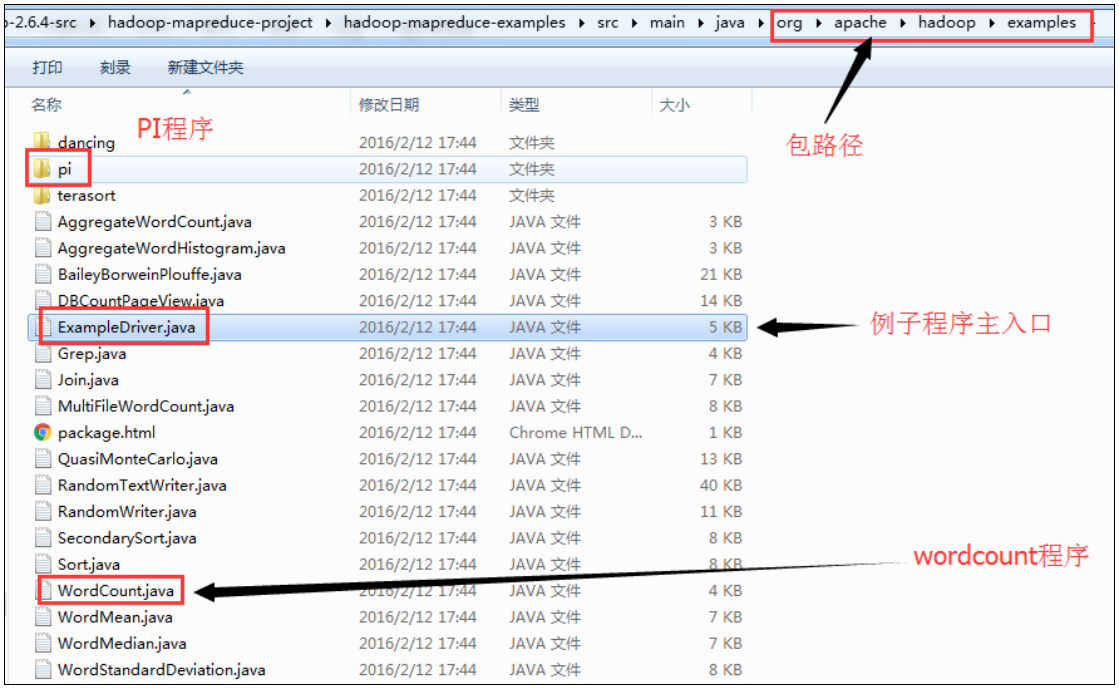
里面有一个例子程序的子项目：hadoop-mapreduce-examples



其中 src 是例子程序源码目录，pom.xml 是该项目的 maven 管理配置文件，我们打开该文件，找到第 127 行，它告诉了我们例子程序的主程序入口：

```
121     <plugin>
122     <groupId>org.apache.maven.plugins</groupId>
123      <artifactId>maven-jar-plugin</artifactId>
124       <configuration>
125        <archive>
126          <manifest>
127            <mainClass>org.apache.hadoop.examples.ExampleDriver</mainClass>
128          </manifest>
129        </archive>
130       </configuration>
131     </plugin>
```

找到该目录：



打开主入口程序，看源代码：

```
35 ⊟ public static void main(String argv[]){
36     int exitCode = -1;
37     ProgramDriver pgd = new ProgramDriver();
38 ⊟   try {
39 ⊟     pgd.addClass("wordcount", WordCount.class,
40             "A map/reduce program that counts the words in the input files.");
41 ⊟     pgd.addClass("wordmean", WordMean.class,
42             "A map/reduce program that counts the average length of the words in the input files.");
43 ⊟     pgd.addClass("wordmedian", WordMedian.class,
44             "A map/reduce program that counts the median length of the words in the input files.");
```

找到这一步，我们就能知道其实 wordcount 程序的实际程序就是 WordCount.class，这就是我们想要找的例子程序的源码

1.4、MapReduce 示例程序编写及编码规范

上一步，我们查看了 WordCount 这个 MapReduce 程序的源码编写，可以得出几点结论：

1、 该程序有一个 main 方法，来启动任务的运行，其中 job 对象就存储了该程序运行的必要信息，比如指定 Mapper 类和 Reducer 类

job.setMapperClass(TokenizerMapper.class);

job.setReducerClass(IntSumReducer.class);

2、 该程序中的 TokenizerMapper 类继承了 Mapper 类

3、 该程序中的 IntSumReducer 类继承了 Reducer 类

总结：MapReduce 程序的业务编码分为两个大部分，一部分配置程序的运行信息，一部分编写该 MapReduce 程序的业务逻辑，并且业务逻辑的 map 阶段和 reduce 阶段的代码分别继承 Mapper 类和 Reducer 类

那么现在就来编写我们自己的 Wordcount 程序

按照上面的编码规范，主体结构应该是这样：

```
public class WordCount {

    public static void main(String[] args) throws Exception {
        Job job = Job.getInstance();
        job.setMapperClass(WCMapper.class);
        job.setReducerClass(WCReducer.class);
        job.submit();
    }
    private static class WCMapper extends Mapper<LongWritable, Text, Text, LongWritable>{
        @Override
        protected void map(LongWritable key, Text value,
                Mapper<LongWritable, Text, Text, LongWritable>.Context context)
                throws IOException, InterruptedException {
            // 在此写maptask的业务代码
        }
    }
    private static class WCReducer extends Reducer<Text, LongWritable, Text, LongWritable>{
        @Override
        protected void reduce(Text arg0, Iterable<LongWritable> arg1,
                Reducer<Text, LongWritable, Text, LongWritable>.Context arg2)
                throws IOException, InterruptedException {
            // 在此写reducetask的业务代码
        }
    }
}
```

mapreduce的主入口，其中用job来管理该程序

WordCount的maptask业务代码

WordCount的reducetask业务代码

MapReduce 程序编写规范：

1、用户编写的程序分成三个部分：Mapper，Reducer，Driver(提交运行 MR 程序的客户端)
2、Mapper 的输入数据是 KV 对的形式（KV 的类型可自定义）
3、Mapper 的输出数据是 KV 对的形式（KV 的类型可自定义）
4、Mapper 中的业务逻辑写在 map()方法中
5、map()方法（maptask 进程）对每一个<K,V>调用一次
6、Reducer 的输入数据类型对应 Mapper 的输出数据类型，也是 KV 对的形式
7、Reducer 的业务逻辑写在 reduce()方法中
8、Reducetask 进程对每一组相同 k 的<K,V>组调用一次 reduce()方法
9、用户自定义的 Mapper 和 Reducer 都要继承各自的父类
10、整个程序需要一个 Drvier 来进行提交，提交的是一个描述了各种必要信息的 job 对象

WordCount 的业务逻辑：

1、 maptask 阶段处理每个数据分块的单词统计分析，思路是每遇到一个单词则把其转换成一个 key-value 对，比如单词 hello，就转换成<'hello',1>发送给 reducetask 去汇总
2、 reducetask 阶段将接受 maptask 的结果，来做汇总计数

下面是具体实现，**首先看 Map：**

```
/**
 * Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>
 *
 * KEYIN 是指框架读取到的数据的key的类型，在默认的InputFormat下，读到的key是一行文本的起始偏移量，所以key的类型是Long
 * VALUEIN 是指框架读取到的数据的value的类型，在默认的InputFormat下，读到的value是一行文本的内容，所以value的类型是String
 * KEYOUT 是指用户自定义逻辑方法返回的数据中key的类型，由用户业务逻辑决定，在此wordcount程序中，我们输出的key是单词，所以是String
 * VALUEOUT 是指用户自定义逻辑方法返回的数据中value的类型，由用户业务逻辑决定，在此wordcount程序中，我们输出的value是单词的数量，所以是Integer
 *
 * 但是，String ，Long等jdk中自带的数据类型，在序列化时，效率比较低，hadoop为了提高序列化效率，自定义了一套序列化框架
 * 所以，在hadoop的程序中，如果该数据需要进行序列化（写磁盘，或者网络传输），就一定要用实现了hadoop序列化框架的数据类型
 *
 * Long ----> LongWritable
 * String ----> Text
 * Integer ----> IntWritable
 * Null ----> NullWritable
 */
```

```
static class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable>{
    @Override
    protected void map(LongWritable key, Text value,Context context)
            throws IOException, InterruptedException {

        // 计算任务代码：切割单词，输出每个单词计 1 的 key-value 对
        String[] words = value.toString().split(" ");
        for(String word: words){
            context.write(new Text(word), new IntWritable(1));
        }
    }
}
```

其次看 Reduce：

```
/**
 * 首先，和前面一样，Reducer类也有输入和输出，输入就是Map阶段的处理结果，输出就是Reduce最后的输出
 * reducetask在调我们写的reduce方法,reducetask应该收到了前一阶段（map阶段）中所有maptask输出的数据中的一部分
 * （数据的key.hashcode%reducetask数==本reductask号），所以reducetaks的输入类型必须和maptask的输出类型一样
 *
 * reducetask将这些收到kv数据拿来处理时，是这样调用我们的reduce方法的：
 * 先将自己收到的所有的kv对按照k分组（根据k是否相同）
 * 将某一组kv中的第一个kv中的k传给reduce方法的key变量，把这一组kv中所有的v用一个迭代器传给reduce方法的变量values
 */
static class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable>{
    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {

        // 汇总计算代码：对每个 key 相同的一组 key-value 做汇总统计
        int sum = 0;
        for(IntWritable v: values){
            sum += v.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

在看 Job：

```
/**
 * 该main方法是该mapreduce程序运行的入口，其中用一个Job类对象来管理程序运行时所需要的很多参数：
 * 比如，指定用哪个组件作为数据读取器、数据结果输出器
 *      指定用哪个类作为map阶段的业务逻辑类，哪个类作为reduce阶段的业务逻辑类
 *      指定wordcount job程序的jar包所在路径
 *      ....
 *      以及其他各种需要的参数
 */
public static void main(String[] args) throws Exception {
    // 指定 hdfs 相关的参数
    Configuration conf = new Configuration();
    conf.set("fs.defaultFS", "hdfs://hadoop02:9000");
    System.setProperty("HADOOP_USER_NAME", "hadoop");

    // 新建一个 job 任务
    Job job = Job.getInstance(conf);

    // 设置 jar 包所在路径
    job.setJarByClass(WordCountMR.class);

    // 指定 mapper 类和 reducer 类
    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducer.class);
```

```
        // 指定 maptask 的输出类型
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        // 指定 reducetask 的输出类型
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // 指定该 mapreduce 程序数据的输入和输出路径
        Path inputPath = new Path("/wordcount/input");
        Path outputPath = new Path("/wordcount/output");
        FileInputFormat.setInputPaths(job, inputPath);
        FileOutputFormat.setOutputPath(job, outputPath);

        // 最后提交任务
        boolean waitForCompletion = job.waitForCompletion(true);
        System.exit(waitForCompletion?0:1);
}
```

1.5、MapReduce 运行方式及 Debug

MapReduce 程序运行方式：

本地运行模式：Eclipse 开发环境下本地运行，好处是方便调试和测试

要点一：MapReduce 程序是被提交给 LocalJobRunner 在本地以单进程的形式运行

要点二：数据输入输出可以在本地，也可以在 HDFS

要点三：怎么实现本地运行？在你的 MapReduce 程序当中不要带集群的配置文件（本质就是由 **mapreduce.framework.name** 和 **yarn.resourcemanager.hostname** 两个参数决定）

要点四：要想实现本地运行，还得做一件事：给 eclipse 安装 hadoop，
有两种方式处理：
1、这个 hadoop 安装必须是 hadoop 在 windows 平台上编译的对应版本。

需要在 windows 本地配置环境变量：
%HADOOP_HOME% = c:/myProgram/hadoop-2.6.1
%PATH% = %HADOOP_HOME%\bin
并且要将 c:/myProgram/hadoop-2.6.1 的 lib 和 bin 目录替换成 windows 平台编译版本

2、这个 hadoop 也可以 apache 官网的安装包，也可以是其他平台编译的，但是需要 windows 平台运行的其他工具支持。

注意，windows 本地也需要安装 hadoop，并且配置环境变量

然后，寻找对应 hadoop 版本的 hadoop.dll 和 winutils.exe 文件，分别做如下处理：
1、hadoop.dll 放置在 c:/windows/system32 文件夹中
2、winutils.exe 放置在$HADOOP_HOME/bin 目录中
3、给 eclipse 配置 hadoop 安装目录，
windows → preferences → Hadoop Map/Reduce → Hadoop Installation Direcotry
然后重启 eclipse 即可

集群运行模式：打 jar 包，提交任务到集群运行

要点一：首先要把代码打成 jar 上传到 linux 服务器

要点二：用 hadoop jar 的命令去提交代码到 yarn 集群运行

要点三：处理的数据和输出结果应该位于 hdfs 文件系统

要点四：如果需要在 windows 中的 eclipse 当中直接提交 job 到集群，则需要修改 YarnRunner 类，这个比较复杂，不建议使用

综合以上两种方式的的运行发现，第 1 种和第 2 种比较方便做调试。

2、MapReduce 程序的核心运行机制

2.1、概述

一个完整的 MapReduce 程序在分布式运行时有两类实例进程：
1、MRAppMaster：负责整个程序的过程调度及状态协调
2、Yarnchild：负责 map 阶段的整个数据处理流程
3、Yarnchild：负责 reduce 阶段的整个数据处理流程
以上两个阶段 MapTask 和 ReduceTask 的进程都是 YarnChild，并不是说这 MapTask 和 ReduceTask 就跑在同一个 YarnChild 进行里

2.2、MapReduce 程序的运行流程

1、一个 mr 程序启动的时候，最先启动的是 MRAppMaster，MRAppMaster 启动后根据本次 job 的描述信息，计算出需要的 maptask 实例数量，然后向集群申请机器启动相应数量的 maptask 进程

2、 maptask 进程启动之后，根据给定的数据切片(哪个文件的哪个偏移量范围)范围进行数据处理，主体流程为：
A、利用客户指定的 InputFormat 来获取 RecordReader 读取数据，形成输入 KV 对
B、将输入 KV 对传递给客户定义的 map()方法，做逻辑运算，并将 map()方法输出的 KV 对收集到缓存
C、将缓存中的 KV 对按照 K 分区排序后不断溢写到磁盘文件

3、 MRAppMaster 监控到所有 maptask 进程任务完成之后（真实情况是，某些 maptask 进程处理完成后，就会开始启动 reducetask 去已完成的 maptask 处 fetch 数据），会根据客户指定的参数启动相应数量的 reducetask 进程，并告知 reducetask 进程要处理的数据范围（数据分区）

4、Reducetask 进程启动之后，根据 MRAppMaster 告知的待处理数据所在位置，从若干台 maptask 运行所在机器上获取到若干个 maptask 输出结果文件，并在本地进行重新归并排序，然后按照相同 key 的 KV 为一个组，调用客户定义的 reduce()方法进行逻辑运算，并收集运算输出的结果 KV，然后调用客户指定的 OutputFormat 将结果数据输出到外部存储

2.3、MapTask 并行度决定机制

maptask 的并行度决定 map 阶段的任务处理并发度，进而影响到整个 job 的处理速度
那么，mapTask 并行实例是否越多越好呢？其并行度又是如何决定呢？

一个 job 的 map 阶段并行度由客户端在提交 job 时决定，客户端对 map 阶段并行度的规划的基本逻辑为：
将待处理数据执行逻辑切片（即按照一个特定切片大小，将待处理数据划分成逻辑上的多个 split），然后每一个 split 分配一个 mapTask 并行实例处理

这段逻辑及形成的切片规划描述文件，是由 FileInputFormat 实现类的 getSplits()方法完成的。该方法返回的是 List<InputSplit>，InputSplit 封装了每一个逻辑切片的信息，包括长度和位置信息，而 getSplits()方法返回一组 InputSplit

2.4、切片机制

FileInputFormat 中默认的切片机制

1、简单地按照文件的内容长度进行切片
2、切片大小，默认等于 block 大小
3、切片时不考虑数据集整体，而是逐个针对每一个文件单独切片
比如待处理数据有两个文件：
File1.txt　　200M
File2.txt　　100M
经过 getSplits()方法处理之后，形成的切片信息是：
File1.txt-split1　　0-128M
File1.txt-split2　　129M-200M
File2.txt-split1　　0-100M

FileInputFormat 中切片的大小的参数配置

通过分析源码，在 FileInputFormat 中，计算切片大小的逻辑：
long splitSize = computeSplitSize(blockSize, minSize, maxSize)，翻译一下就是求这三个值的中间值

切片主要由这几个值来运算决定：
blocksize：默认是 128M，可通过 dfs.blocksize 修改
minSize：默认是 1，可通过 mapreduce.input.fileinputformat.split.minsize 修改
maxsize：默认是 Long.MaxValue，可通过 mapreduce.input.fileinputformat.split.maxsize 修改

因此，如果 maxsize 调的比 blocksize 小，则切片会小于 blocksize
　　如果 minsize 调的比 blocksize 大，则切片会大于 blocksize
但是，不论怎么调参数，都不能让多个小文件“划入”一个 split

2.5、MapTask 并行度经验之谈

如果硬件配置为 2*12core + 64G，恰当的 map 并行度是大约每个节点 20-100 个 map，最好每个 map 的执行时间至少一分钟。
1、如果 job 的每个 map 或者 reduce task 的运行时间都只有 30-40 秒钟，那么就减少该 job 的 map 或者 reduce 数，每一个 task(map|reduce)的 setup 和加入到调度器中进行调度，这个中间的过程可能都要花费几秒钟，所以如果每个 task 都非常快就跑完了，就会在 task 的开始和结束的时候浪费太多的时间。
配置 task 的 JVM 重用可以改善该问题：
mapred.job.reuse.jvm.num.tasks，默认是 1，表示一个 JVM 上最多可以顺序执行的 task 数目（属于同一个 Job）是 1。也就是说一个 task 启一个 JVM。这个值可以在 mapred-site.xml 中进行更改，当设置成多个，就意味着这多个 task 运行在同一个 JVM 上，但不是同时执行，是排队顺序执行

2、如果 input 的文件非常的大，比如 1TB，可以考虑将 hdfs 上的每个 blocksize 设大，比如设成 256MB 或者 512MB

2.6、ReduceTask 并行度决定机制

reducetask 的并行度同样影响整个 job 的执行并发度和执行效率，但与 maptask 的并发数由切片数决定不同，Reducetask 数量的决定是可以直接手动设置：

job.setNumReduceTasks(4);
默认值是 1，
手动设置为 4，表示运行 4 个 reduceTask，
设置为 0，表示不运行 reduceTask 任务，也就是没有 reducer 阶段，只有 mapper 阶段

如果数据分布不均匀，就有可能在 reduce 阶段产生数据倾斜
注意：reducetask 数量并不是任意设置，还要考虑业务逻辑需求，有些情况下，需要计算全局汇总结果，就只能有 1 个 reducetask

尽量不要运行太多的 reducetask。对大多数 job 来说，最好 rduce 的个数最多和集群中的 reduce 持平，或者比集群的 reduce slots 小。这个对于小集群而言，尤其重要。