hw11: Hadoop MapReduce

- 在Hadoop的MapReduce Tutorial中()，给出了一个WordCount 2.0版本的实现，相比1.0版本，它增加了对若干Hadoop MR特性的运用。请你参考这个2.0版本的实现，在你的E-BookStore中增加如下的功能，为方便起见，你可以将该功能开发成单独的工程：

  https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.

  html

1. 将你的系统中所有图书的简介按照图书类型分别存储到多个文本文件中，例如，所有计算机类图书的简介存储在CS.txt中，科幻小说的简介存储在Fiction.txt中。请你构建多个这样的文件，作为MR作业的对象。

2. 编写一个关键词列表，包含若干单词，例如，["Java","JavaScript","C++","Programming","Star","Robot"]等。

3. 编写一个MR作业，统计所有图书简介中上述每个关键词出现的次数。

- 提交物：

– 请提交你构造的图书简介文件和关键词列表，以及你编写的类文件。

– 编写一个Word文档，说明你的程序的运行方式，以及你做了哪些特殊的参数设置，然后截图展示运行结果。

– 在上述Word文档中，说明在你的程序运行时，Mapper和Reducer各有多少个？以及为什么有这样的数量。

- 评分标准：

1. 能够正确配置并运行 MR 作业，得到正确的结果：3 分

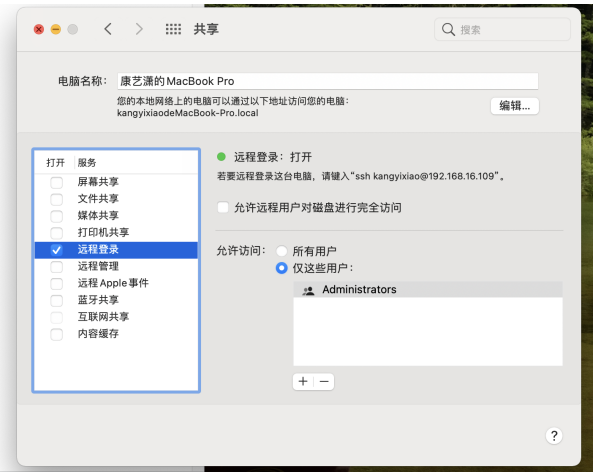
2. 文档中对相关参数的设置说明合理：1分

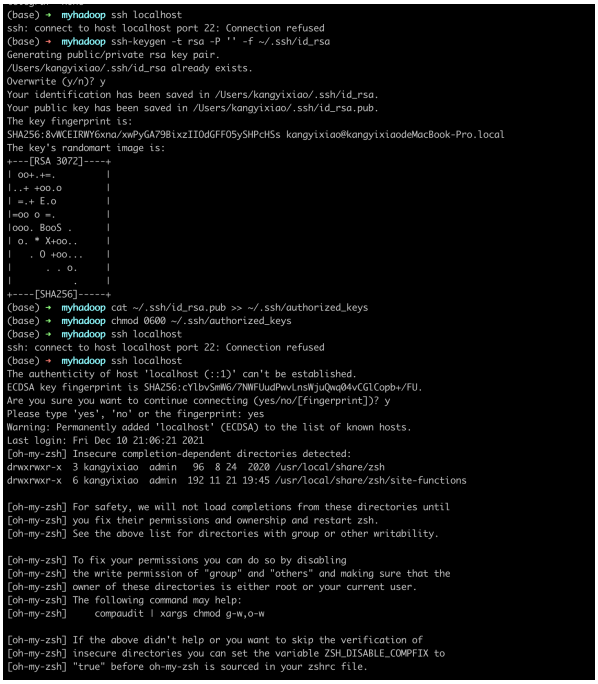
3. 文档中对程序Mapper和Reducer的数量观察正确且解释正确：1 分

4. 注：第2-3项得分在验收时会让同学们运行程序，以验证你的文档说明是否正确

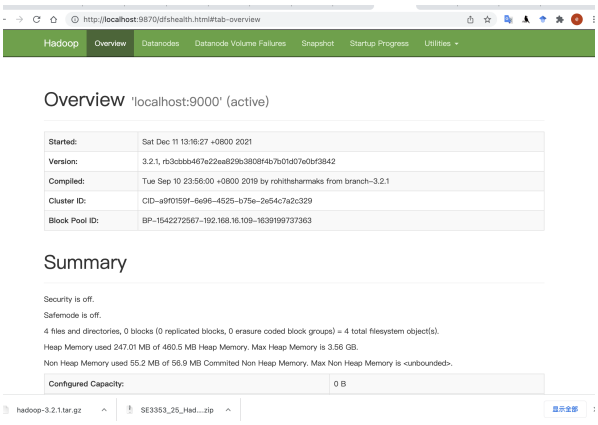
下载配置hadoop

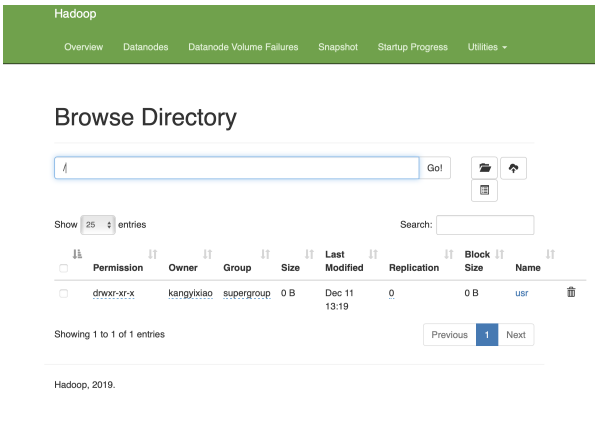
ssh localhost 报错connection refused, 需要输入一下三条命令并且设置→共享→ 远程登录（√）





输入 localhost:9870 就可以打开客户端





bin/hdfs dfs -mkdir -p /usr/WordCount

参考教程

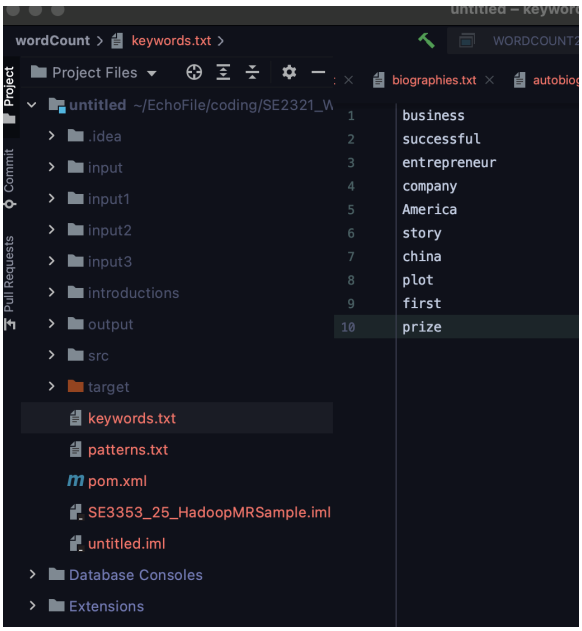
https://www.youtube.com/watch?v=BdHQFAP98_A

map reduce

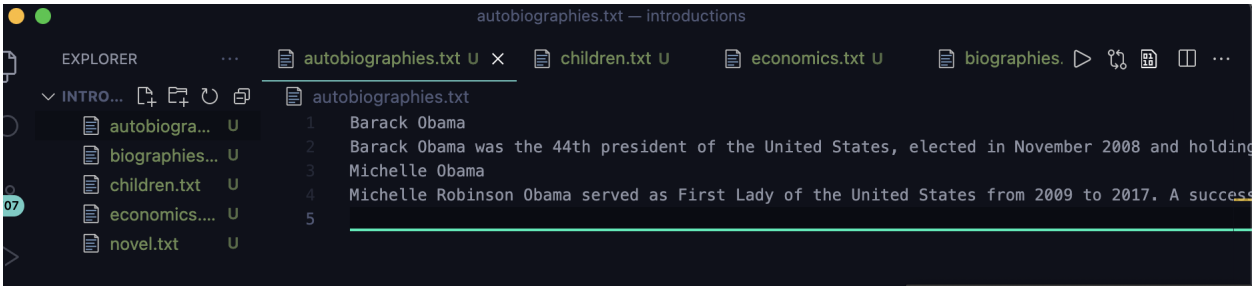
输入

设置keyword list

keywords.txt



各个书籍类的txt介绍文件, 放在introductions文件下



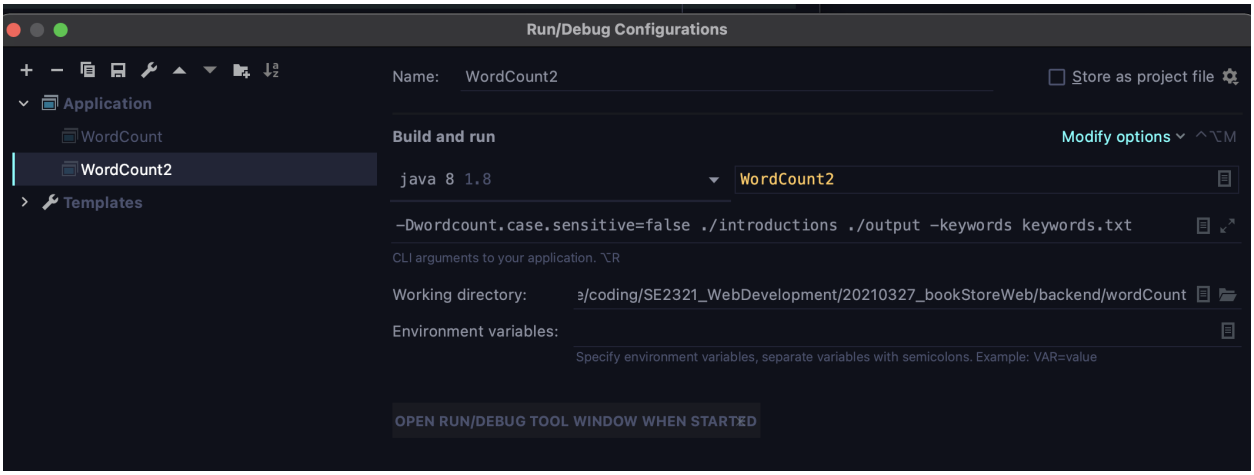
程序实现

```
public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        GenericOptionsParser optionParser = new GenericOptionsParser(conf, args);
        String[] remainingArgs = optionParser.getRemainingArgs();
        if ((remainingArgs.length != 2) && (remainingArgs.length != 4)) {
            System.err.println("Usage: wordcount <in> <out> [-skip skipPatternFile]");
            System.exit(2);
        }
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount2.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setNumReduceTasks(2);
        List<String> otherArgs = new ArrayList<String>();
        for (int i=0; i < remainingArgs.length; ++i) {
            if ("-keywords".equals(remainingArgs[i])) {
                job.addCacheFile(new Path(remainingArgs[++i]).toUri());
                job.getConfiguration().setBoolean("wordcount.skip.patterns", true);
            } else {
                otherArgs.add(remainingArgs[i]);
            }
        }
        FileInputFormat.addInputPath(job, new Path(otherArgs.get(0)));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs.get(1)));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
```

main 函数接受参数, 指定输入输出以及 keywordslist 的文件名

参数如下：

`Dwordcount.case.sensitive=false ./introductions ./output -keywords keywords.txt`



使用DistributedCache 来记录keywordslit

```
@Override
        public void setup(Context context) throws IOException,
                InterruptedException {
            UUID uuid = UUID.randomUUID();
            System.out.println("mapper uuid:"+uuid);
            conf = context.getConfiguration();
            caseSensitive = conf.getBoolean("wordcount.case.sensitive", true);
            if (conf.getBoolean("wordcount.skip.patterns", false)) {
                URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();
                for (URI patternsURI : patternsURIs) {
                    Path patternsPath = new Path(patternsURI.getPath());
                    String patternsFileName = patternsPath.getName().toString();
                    parseKeywordsFile(patternsFileName);
                }
            }
            patternsToSkip.add("\\.");
            patternsToSkip.add("\\,");
            patternsToSkip.add("\\!");
            patternsToSkip.add("\\?");

        }
```

setup 函数设置忽略的字符包括. , ! ？ 从DistributedCache 拿到文件名，穿给parseKeywordsFile函数。

```
private void parseKeywordsFile(String fileName) {
            try {
                fis = new BufferedReader(new FileReader(fileName));
                String keyword = null;
                while ((keyword = fis.readLine()) != null) {
                    keywords.add(keyword);
```

```
                }
            } catch (IOException ioe) {
                System.err.println("Caught exception while parsing the cached file '"
                        + StringUtils.stringifyException(ioe));
            }
        }
```

读出keywords.txt 的字符串，存储进keywords set.

```
@Override
        public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {

            String line = (caseSensitive) ?
                    value.toString() : value.toString().toLowerCase();
            for (String pattern : patternsToSkip) {
                line = line.replaceAll(pattern, "");
            }
            StringTokenizer itr = new StringTokenizer(line);
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                String s=word.toString();
                if(!keywords.contains(s))continue;
                context.write(word, one);
                Counter counter = context.getCounter(CountersEnum.class.getName(),
                        CountersEnum.INPUT_WORDS.toString());
                counter.increment(1);
            }
        }
```

首先将需要忽略的字符全部删除，然后对于每个单词查找是否在keywords中， 如果不在就跳过， 如果在就counter+=1, context 写下

(word, one)的keyval 对

```
public static class IntSumReducer
            extends Reducer<Text,IntWritable,Text,IntWritable> {
        UUID uuid;
        public IntSumReducer(){
            uuid=UUID.randomUUID();
            System.out.println("reducer uuid:"+uuid);
        }
        private IntWritable result = new IntWritable();
        public void reduce(Text key, Iterable<IntWritable> values,
                           Context context
        ) throws IOException, InterruptedException {

            int sum = 0;
```

```
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }
```

每个reducer会得到的输入是一个key(某个单词）， 一个values的集合， 它需要统计这个values的求和result, 将结果（key, result)写入磁盘。

mapper和reducer的数量

mapper的数量由输入文件被切分成blocks的数量决定。blocksize=128MB , 如果单个文件大于这个大小就会被分割。由于我的5个输入文件都没有超出这个大小，于是有5个block, 对应于5次job.

在本次实现中，在main函数中设置reducer为2. `job.setNumReduceTasks(2);`

理论上， 对于每个job, 会有1个mapper和2个reducer.

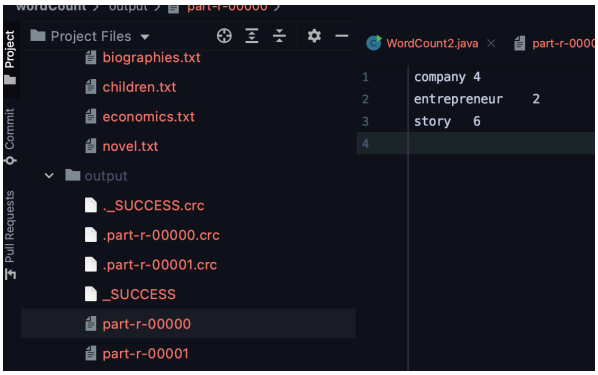
进行程序观察，在mapper和reducer都加入了成员变量uuid, 并且在初始化的时候输出UUID的值。

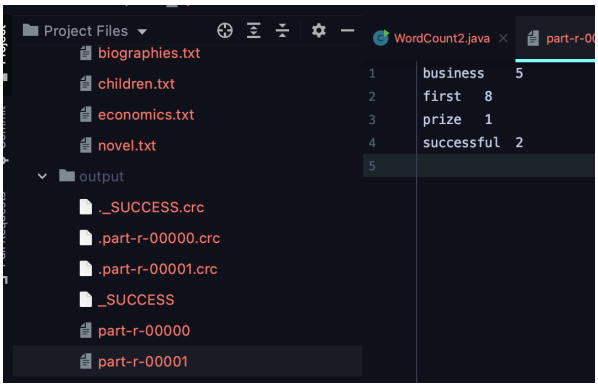
```
public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
    UUID uuid;
    public IntSumReducer(){
        uuid=UUID.randomUUID();
        System.out.println("reducer uuid:"+uuid);
    }
    private IntWritable result = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
    ) throws IOException, InterruptedException {

        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

```
public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{

    static enum CountersEnum { INPUT_WORDS }
    private final static IntWritable one = new IntWritable( value: 1);
    private Text word = new Text();
    private boolean caseSensitive;
    private Set<String> patternsToSkip = new HashSet<String>();
    private Set<String> keywords = new HashSet<String>();
    private Configuration conf;
    private BufferedReader fis;
    UUID uuid;
    @Override
    public void setup(Context context) throws IOException,
            InterruptedException {
        UUID uuid = UUID.randomUUID();
        System.out.println("mapper uuid:"+uuid);
        conf = context.getConfiguration();
```

mapper 直接在setup的时候输出和构造函数输出效果一致

运行结果






```
mapper uuid:6f71e71c-78ee-4980-a92f-9e023985b2fe
reducer uuid:7f99f34c-bc40-4e4d-94dc-0e51bf64b76b
reducer uuid:c79e1676-d9a5-4798-bdb4-bbbda6138900
mapper uuid:b38c3a39-87cb-4685-9d96-a4d0604d68d5
reducer uuid:1521c995-9503-47e3-8cd4-b0d38b3c7465
reducer uuid:9fdd0ce2-2ec7-424f-b1fe-76d8520d2c92
mapper uuid:1b234ba0-745b-457e-8cbf-33e3a9db5db6
reducer uuid:248d2a47-2753-4af8-8fa5-bb46d1cc58b9
reducer uuid:8113ac3f-befb-4ca9-a562-d52ccb6df43e
mapper uuid:5418dd74-97b8-4ae7-b85d-da36c5f00d18
reducer uuid:2fb72198-c264-4c33-be52-4b579efbadcf
reducer uuid:d359034b-2841-4eb7-9948-b52930896820
mapper uuid:7629c716-7188-4c9c-9afa-63ecd44cee62
reducer uuid:410b7a82-9df2-4871-9999-229508cdb2a3
reducer uuid:d25c54a6-0ab9-45e1-a286-93aed0749605

Process finished with exit code 0
```

在output 出现了两个文件，应该是由于reducer设置了2产生的（尝试过如果默认也就是reducer=1的情况只有一个文件part-r-00000),分别统计了不同的单词。打印出来的log和上面的mapper reducer的个数分析一致，有5个job, 也就是对应一个job，1个mapper和2个reducer.

在mapreduce 中，框架解决了很多问题，比如切分任务，分配mapper, reducer, 容错等，我们只需要写好对输入输出的处理，mapper和reducer的工作就可以了非常方便。