Hadoop 学习笔记

最简单的例子

main函数几乎一样都

> 最简单的例子datecount

```
public class DateCount {

        public static class DateCountMapper extends Mapper<Object, Text, Text, IntWritable> {
                private final static IntWritable one = new IntWritable(1);

                public void map(Object key, Text value, Context context )
                                throws IOException, InterruptedException {
                String[] strs = value.toString().split(" ");    //按空格分割输入
                Text date = new Text(strs[0]);          //获取日期
                        context.write(date, one);                       //将日期和常数1作为Map输出
            }
        }

        public static class DateCountReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
                public void reduce(Text key, Iterable<IntWritable> values, Context context)
                                throws IOException, InterruptedException {
                        int sum = 0;
                        for (IntWritable val : values) {
                                sum += val.get();
                        }
                        context.write(key, new IntWritable(sum));
            }
        }

        public static void main(String[] args) throws Exception {
                String namenode_ip = "192.168.17.10";
                String hdfs = "hdfs://" + namenode_ip + ":9000";
                Configuration conf = new Configuration();
                conf.set("fs.defaultFS", hdfs);
                conf.set("mapreduce.app-submission.cross-platform", "true");
                conf.set("mapreduce.framework.name", "yarn");
                conf.set("yarn.resourcemanager.hostname", namenode_ip);
                conf.set("yarn.application.classpath", "/opt/hadoop/etc/hadoop:/opt/hadoop/share/hadoop/commo
                conf.set("mapreduce.jobhistory.address", namenode_ip+":10020");

                String jobName = "DateCount";                                   //作业名称
                Job job = Job.getInstance(conf, jobName);
                job.setJarByClass(DateCount.class);                             //指定运行时作业类
                job.setJar("export//DateCount.jar");                    //指定本地jar包
                job.setMapperClass(DateCountMapper.class);              //指定Mapper类
                job.setMapOutputKeyClass(Text.class);                   //设置Mapper输出Key类型
                job.setMapOutputValueClass(IntWritable.class);  //设置Mapper输出Value类型
                job.setReducerClass(DateCountReducer.class);    //指定Reducer类
                job.setOutputKeyClass(Text.class);                              //设置Reduce输出Key类型
```

```
                job.setOutputValueClass(IntWritable.class);     //设置Reduce输出Value类型

                String dataDir = "/expr/datecount/data";                //实验数据目录
                String outputDir = "/expr/datecount/output";    //实验输出目录
                Path inPath = new Path(hdfs + dataDir);
                Path outPath = new Path(hdfs + outputDir);
                FileInputFormat.addInputPath(job, inPath);
                FileOutputFormat.setOutputPath(job, outPath);
                FileSystem fs = FileSystem.get(conf);
                if(fs.exists(outPath)) {
                        fs.delete(outPath, true);
                }

                System.out.println("Job: " + jobName + " is running...");
                if(job.waitForCompletion(true)) {
                        System.out.println("success!");
                        System.exit(0);
                } else {
                        System.out.println("failed!");
                        System.exit(1);
                }
        }

}
```

分析map

> map端负责把数据从文件读出来，进行清洗，map函数里的操作只针对文件里的一行，将<key,value>写入context。

分析reduce

> reduce接受map输出的数据，reduce函数里的操作是针对一个key来做的，将key为相同值元素技术，然后写入context，即最后的输出。

注意：extends Mapper<Object, Text, Text, IntWritable>map函数的输入输出在一开始就定义好了，同理reduce也是

分区

```
job.setPartitionerClass(YearPartitioner.class);         //自定义分区方法
job.setNumReduceTasks(3);       //设置reduce任务的数量,该值传递给Partitioner.getPartition()方法的numPartitions参数
```

> 设置setPartitionerClass，setNumReduceTask(n);在主函数中增加了这两个设置，n是分区的数量

```
    public static class YearPartitioner extends Partitioner<Text, IntWritable> {
                @Override
                public int getPartition(Text key, IntWritable value, int numPartitions) {
                        //根据年份对数据进行分区，返回不同分区号
                        if (key.toString().startsWith("2015"))
                                return 0 % numPartitions;
                        else if (key.toString().startsWith("2016"))
                                return 1 % numPartitions;
                        else
                                return 2 % numPartitions;
                }
    }
```

在类中增加Partitioner类，重写getPartition(Text key, IntWritable value, int numPartitions)，里面代码一般根据key值来分

排序

自定义类实现WritableComparable接口

```
    public static class MyKey implements WritableComparable<MyKey> {
                //成员变量
                private String date;
                private int num;

                public String getDate() {
                        return date;
                }

                public void setDate(String date) {
                        this.date = date;
                }

                public int getNum() {
                        return num;
                }

                public void setNum(int num) {
                        this.num = num;
                }

                //构造方法
                public MyKey() {
                }

                public MyKey(String date, int num) {
                        this.date = date;
                        this.num = num;
                }

                public void write(DataOutput out) throws IOException {
                        out.writeUTF(date);
                        out.writeInt(num);
                }

                public void readFields(DataInput in) throws IOException {
                        date = in.readUTF();
                        num = in.readInt();
                }

                public int compareTo(MyKey o) {
                        if (!date.equals(o.date))
                                return date.compareTo(o.date);
                        else
                                return o.num-num;
                }
        }
```

> 排序a.compareTO(b)，若a>b返回1，a=b返回0，a<b返回-1，如果是一个正数则调换顺序。

默认按key值升序排序，如有需要倒序需要自行设置**注意：继承extends WritableComparator**

```
    public static class MySort extends WritableComparator {
                public MySort() {
                        super(IntWritable.class, true);
                }

                @SuppressWarnings({ "rawtypes", "unchecked" })
                public int compare(WritableComparable a, WritableComparable b) {
                        //b与a交换位置，倒序
                        return b.compareTo(a);
                }
        }

//main里设置
job.setSortComparatorClass(MySort.class);               //设置自定义排序类
```

分组

默认按照升序比较，把给定key出现的序号写到一个列表里输出,默认情况reduce下

```
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
                                throws IOException, InterruptedException {
                        StringBuilder sb = new StringBuilder();
                        sb.append("[ ");
                        for (IntWritable val : values) {        //将value值串联
                                sb.append(val.toString()).append(" ");
                        }
                        sb.append("]");
                        context.write(key, new Text(sb.toString()));
           }
```

自定义分组方法

```
public static class MyGroup extends WritableComparator {
                public MyGroup() {
                        super(Text.class, true);
                }

                @SuppressWarnings("rawtypes")
                @Override
                public int compare(WritableComparable a, WritableComparable b) {
                        String d1 = a.toString();
                        String d2 = b.toString();

                        if (d1.startsWith("2015"))
                                d1 = "2015";
                        else if (d1.startsWith("2016"))
                                d1 = "2016";
                        else
                                d1 = "2017";

                        if (d2.startsWith("2015"))
                                d2 = "2015";
                        else if (d2.startsWith("2016"))
                                d2 = "2016";
                        else
                                d2 = "2017";

                        return d1.compareTo(d2);        //将原本KEY(年月日)的比较变成年份的比较
                }
        }
//main里设置
job.setGroupingComparatorClass(MyGroup.class);  //设置自定义分组类
```

TopN问题

> Mapper和Reducer类里增加了一个cleanup函数，且增加了一个TreeMap<key,value>类型变量

Mapper

```
public class TopTenMapper extends Mapper<Object, Text, NullWritable, Text> {
        private TreeMap<Integer, Text> visitTimesMap = new TreeMap<Integer, Text>();    //TreeMap是有序KV集合

        @Override
        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
                if (value == null) {    //空行不做处理
                        return;
                }
                String[] strs = value.toString().split(" ");
                String tId = strs[0];
                String tVisitTimes = strs[1];
                if (tId == null || tVisitTimes == null) {       //ID或访问次数为空，则不处理
                        return;
                }

                //将访问次数作为KEY、将行记录（帖子ID+访问次数）作为VALUE，放入TreeMap中按KEY自动升序排列
                visitTimesMap.put(Integer.parseInt(tVisitTimes), new Text(value));

                //如果TreeMap中元素超过N个，将第一个（KEY最小的）元素删除
                if (visitTimesMap.size() > 10) {
                        visitTimesMap.remove(visitTimesMap.firstKey());
                }
        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
                for (Text val : visitTimesMap.values()) {
                        context.write(NullWritable.get(), val); //在cleanup()中完成Map输出
                }
        }
}
```

Reducer

```
public class TopTenReducer extends Reducer<NullWritable, Text, NullWritable, Text> {
        private TreeMap<Integer, Text> visitTimesMap = new TreeMap<Integer, Text>();

        @Override
        public void reduce(NullWritable key, Iterable<Text> values, Context context)
                        throws IOException, InterruptedException {
                //因为key为空，所以各个文件中的value都在同一个组内作为reduce的输入
                for (Text val : values) {
                        String[] strs = val.toString().split(" ");
                        visitTimesMap.put(Integer.parseInt(strs[1]), new Text(val));    //同map方法
                        if (visitTimesMap.size() > 10) {
                                visitTimesMap.remove(visitTimesMap.firstKey());
                        }
                }
        }

        public void cleanup(Context context) throws IOException, InterruptedException {
                //将TreeMap反序处理，降序输出top10
                NavigableMap<Integer, Text> inverseMap = visitTimesMap.descendingMap(); //获得TreeMap反序
                for (Text val : inverseMap.values()) {
                        context.write(NullWritable.get(), val);
                }
        }
}
```

计数器

> 在map函数里使用计数器

```
public class YearCounter {

	//自定义年份计数器
	private enum YCounter {		//枚举名称代表分组
		Y2015, Y2016, Y2017		//成员名称代表计数器
	}

	public static class YearCounterMapper extends Mapper<Object, Text, Text, IntWritable> {
		private final static IntWritable one = new IntWritable(1);
		private Text date =new Text();

		public void map(Object key, Text value, Context context )
				throws IOException, InterruptedException {
		String[] strs = value.toString().split(" ");
		date.set(strs[0]);
			context.write(date, one);

			//根据KEY值不同，增加对应计数器的值
			String year = strs[0].substring(0, 4);
			switch (year) {
			case "2015" :
				//动态自定义计数器：组名+计数器
		context.getCounter("DynamicCounter", "Y2015").increment(1);
		//枚举声明计数器，通过上下文对象获取计数器对象并计数
				context.getCounter(YCounter.Y2015).increment(1);
				break;
			case "2016" :
				context.getCounter("DynamicCounter", "Y2016").increment(1);
				context.getCounter(YCounter.Y2016).increment(1);
				break;
			case "2017" :
				context.getCounter("DynamicCounter", "Y2017").increment(1);
				context.getCounter(YCounter.Y2017).increment(1);
				break;
			}

			//在控制台输出计数器值
			//System.out.println("Y2015="+context.getCounter(YCounter.Y2015).getValue());
	    }
	}

	public static class YearCounterReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
		public void reduce(Text key, Iterable<IntWritable> values, Context context)
				throws IOException, InterruptedException {
			int sum = 0;
			for (IntWritable val : values) {
				sum += val.get();
			}
			context.write(key, new IntWritable(sum));
	    }
	}
}
```

代码问题我觉得知道这些足矣，剩下的可以自己再好好看看上机作业。