MapReduce 典型编程场景 1

目录

1、MapReduce 多 Job 串联

1.1、需求

一个稍复杂点的处理逻辑往往需要多个 MapReduce 程序串联处理，多 job 的串联可以借助 MapReduce 框架的 JobControl 实现

1.2、实例

以下有两个 MapReduce 任务，分别是 Flow 的 SumMR 和 SortMR，其中有依赖关系：SumMR 的输出是 SortMR 的输入，所以 SortMR 的启动得在 SumMR 完成之后

以下是代码实现：

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job jobsum = Job.getInstance(conf);
    jobsum.setJarByClass(RunManyJobMR.class);
    jobsum.setMapperClass(FlowSumMapper.class);
```

```
        jobsum.setReducerClass(FlowSumReducer.class);
        jobsum.setMapOutputKeyClass(Text.class);
        jobsum.setMapOutputValueClass(Flow.class);
        jobsum.setCombinerClass(FlowSumReducer.class);
        jobsum.setOutputKeyClass(Text.class);
        jobsum.setOutputValueClass(Text.class);
        FileInputFormat.setInputPaths(jobsum, "d:/flow/input");
        FileOutputFormat.setOutputPath(jobsum, new Path("d:/flow/output12"));

        Job jobsort = Job.getInstance(conf);
        jobsort.setJarByClass(RunManyJobMR.class);
        jobsort.setMapperClass(FlowSortMapper.class);
        jobsort.setReducerClass(FlowSortReducer.class);
        jobsort.setMapOutputKeyClass(Flow.class);
        jobsort.setMapOutputValueClass(Text.class);
        jobsort.setOutputKeyClass(NullWritable.class);
        jobsort.setOutputValueClass(Flow.class);
        FileInputFormat.setInputPaths(jobsort, "d:/flow/output12");
        FileOutputFormat.setOutputPath(jobsort, new Path("d:/flow/sortoutput12"));

        ControlledJob sumcj = new ControlledJob(jobsum.getConfiguration());
        ControlledJob sortcj = new ControlledJob(jobsort.getConfiguration());

        sumcj.setJob(jobsum);
        sortcj.setJob(jobsort);

        // 设置作业依赖关系
        sortcj.addDependingJob(sumcj);

        JobControl jc = new JobControl("flow sum and sort");
        jc.addJob(sumcj);
        jc.addJob(sortcj);

        Thread jobThread = new Thread(jc);
        jobThread.start();

        while(!jc.allFinished()){
            Thread.sleep(500);
        }
        jc.stop();
}
```

2、TopN 算法实现-自定义 GroupComparator

2.1、需求

在统计学生成绩的小项目中，现在有一个需求：
求出每个班参考学生成绩最高的学生的信息，班级，姓名和平均分

2.2、分析

1、利用“班级和平均分”作为 key，可以将 map 阶段读取到的所有学生成绩数据按照班级和成绩排倒序，发送到 reduce
2、在 reduce 端利用 GroupingComparator 将班级相同的 kv 聚合成组，然后取第一个即是最大值

2.3、实现

第一步：先把分组和排序字段都综合到一个自定义对象里

```
package com.ghgj.mr.topn;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.WritableComparable;

public class ClazzScore implements WritableComparable<ClazzScore>{

	private String clazz;
	private Double score;

	public String getClazz() {
		return clazz;
	}

	public void setClazz(String clazz) {
		this.clazz = clazz;
	}

	public Double getScore() {
		return score;
	}
```

```
public void setScore(Double score) {
	this.score = score;
}

public ClazzScore(String clazz, Double score) {
	super();
	this.clazz = clazz;
	this.score = score;
}

public ClazzScore() {
	super();
	// TODO Auto-generated constructor stub
}

@Override
public String toString() {
	return clazz + "\t" + score;
}

@Override
public void write(DataOutput out) throws IOException {
	out.writeUTF(clazz);
	out.writeDouble(score);
}

@Override
public void readFields(DataInput in) throws IOException {
	// TODO Auto-generated method stub
	this.clazz = in.readUTF();
	this.score = in.readDouble();
}

/**
 * key 排序
 */
@Override
public int compareTo(ClazzScore cs) {
	int it = cs.getClazz().compareTo(this.clazz);
	if(it == 0){
		return (int) (cs.getScore() - this.score);
	}else{
		return it;
```

```
            }
        }
}
```

第二步：编写排序之后的 ClazzScore 数据传入 ReduceTask 的分组规则

```
package com.ghgj.mr.topn;

import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

public class ClazzScoreGroupComparator extends WritableComparator{

    ClazzScoreGroupComparator(){
        super(ClazzScore.class, true);
    }
    /**
     * 决定输入到 reduce 的数据的分组规则
     */
    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        // TODO Auto-generated method stub
        ClazzScore cs1 = (ClazzScore)a;
        ClazzScore cs2 = (ClazzScore)b;
        int it = cs1.getClazz().compareTo(cs2.getClazz());
        return it;
    }
}
```

第三步：编写 MapReduce 程序

```
package com.ghgj.mr.topn;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

```
/**
 * TopN 问题
 */
public class ScoreTop1MR {

	public static void main(String[] args) throws Exception {

		Configuration conf = new Configuration();
		Job job = Job.getInstance(conf);

		job.setJarByClass(ScoreTop1MR.class);

		job.setMapperClass(ScoreTop1MRMapper.class);
		job.setReducerClass(ScoreTop1MRReducer.class);

		job.setOutputKeyClass(ClazzScore.class);
		job.setOutputValueClass(DoubleWritable.class);

		// 设置传入 reducer 的数据分组规则
		job.setGroupingComparatorClass(ClazzScoreGroupComparator.class);

		FileInputFormat.setInputPaths(job, "d:/score_all/input");
		Path p = new Path("d:/score_all/output1");
		FileSystem fs = FileSystem.newInstance(conf);
		if(fs.exists(p)){
			fs.delete(p, true);
		}
		FileOutputFormat.setOutputPath(job, p);

		boolean status = job.waitForCompletion(true);
		System.exit(status ? 0 : 1);
	}

	static class ScoreTop1MRMapper extends Mapper<LongWritable, Text, ClazzScore, DoubleWritable>{
		@Override
		protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
			String[] splits = value.toString().split("\t");
			ClazzScore cs = new ClazzScore(splits[0], Double.parseDouble(splits[2]));
			context.write(cs, new DoubleWritable(Double.parseDouble(splits[2])));
		}
	}
```

```
    static class ScoreTop1MRReducer extends Reducer<ClazzScore, DoubleWritable, ClazzScore, DoubleWritable>{

        @Override
        protected void reduce(ClazzScore cs, Iterable<DoubleWritable> scores, Context context) throws IOException, InterruptedException {
            // 按照规则，取每组的第一个就是 Top1
            context.write(cs, scores.iterator().next());
        }
    }
}
```

3、MapReduce 全局计数器

3.1、介绍

计数器是用来记录 job 的执行进度和状态的。它的作用可以理解为日志。我们可以在程序的某个位置插入计数器，记录数据或者进度的变化情况。

MapReduce 计数器（Counter）为我们提供一个窗口，用于观察 MapReduce Job 运行期的各种细节数据。对 MapReduce 性能调优很有帮助，MapReduce 性能优化的评估大部分都是基于这些 Counter 的数值表现出来的。

MapReduce 自带了许多默认 Counter，现在我们来分析这些默认 Counter 的含义，方便大家观察 Job 结果，如输入的字节数、输出的字节数、Map 端输入/输出的字节数和条数、Reduce 端的输入/输出的字节数和条数等

3.2、需求

在实际生产代码中，常常需要将数据处理过程中遇到的不合规数据行进行全局计数，类似这种需求可以借助 MapReduce 框架中提供的全局计数器来实现

3.3、实例

以下是一个利用全局计数器来统计一个目录下所有文件出现的单词总数和总行数

```
package com.ghgj.mr.counter;

import java.io.IOException;
```

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class CounterWordCount {

    enum CouterWordCountC{COUNT_WORDS, COUNT_LINES}

    public static void main(String[] args) throws Exception {

        // 指定 hdfs 相关的参数
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf);
        // 设置 jar 包所在路径
        job.setJarByClass(CounterWordCount.class);

        job.setMapperClass(WCCounterMapper.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(LongWritable.class);

        // 本地路径
        Path inputPath = new Path("d:/wordcount/input");
        FileInputFormat.setInputPaths(job, inputPath);

        job.setNumReduceTasks(0);

        Path outputPath = new Path("d:/wordcount/output");
        FileSystem fs = FileSystem.get(conf);
        if(fs.exists(outputPath)){
            fs.delete(outputPath, true);
        }
        FileOutputFormat.setOutputPath(job, outputPath);

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

```
    private static class WCCounterMapper extends Mapper<LongWritable, Text, Text,
LongWritable>{
        @Override
        protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            // 统计行数，因为默认读取文本是逐行读取，所以 map 执行一次，行数+1
            context.getCounter(CouterWordCountC.COUNT_LINES).increment(1L);
            String[] words = value.toString().split(" ");
            for(String word: words){
                // 统计单词总数，遇见一个单词就+1
                context.getCounter(CouterWordCountC.COUNT_WORDS).increment(1L);
            }
        }
    }
}
```

4、MapJoin-DistributedCache 应用

4.1、MapReduce Join 介绍

在各种实际业务场景中，按照某个关键字对两份数据进行连接是非常常见的。如果两份数据都比较小，那么可以直接在内存中完成连接。如果是大数据量的呢？显然，在内存中进行连接会发生 OOM。MapReduce 可以用来解决大数据量的链接

MapReduce 的 Join 操作主要分两类：MapJoin 和 ReduceJoin

先看 ReduceJoin:

1、 map 阶段，两份数据 data1 和 data2 会被 map 分别读入，解析成以链接字段为 key 以查询字段为 value 的 key-value 对，并标明数据来源是 data1 还是 data2。
2、 reduce 阶段，reducetask 会接收来自 data1 和 data2 的相同 key 的数据，在 reduce 端进行乘积链接，最直接的影响是很消耗内存，导致 OOM

再看 MapJoin:

MapJoin 适用于有一份数据较小的连接情况。做法是直接把该小份数据直接全部加载到内存当中，按链接关键字建立索引。然后大份数据就作为 MapTask 的输入，对 map()方法的每次输入都去内存当中直接去匹配连接。然后把连接结果按 key 输出，这种方法要使用 hadoop 中的 DistributedCache 把小份数据分布到各个计算节点，每个 maptask 执行任务的节点都需要加载该数据到内存，并且按连接关键字建立索引

4.2、需求

现有两份数据 movies.dat 和 ratings.dat
数据样式分别为：
Movies.dat

```
1::Toy Story (1995)::Animation|Children's|Comedy
2::Jumanji (1995)::Adventure|Children's|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
```

字段含义：movieid, moviename, movietype

Ratings.dat

```
1::1193::5::978300760
1::661::3::978302109
1::914::3::978301968
```

字段含义：userid, movieid, rate, timestamp

Select * from movie a join ratings b on a.movieid = b.movieid

现要求对两表进行连接，要求输出最终的结果有以上六个字段：
movieid, userid, rate, moviename, movietype, timestamp

4.3、实现

第一步：封装 MovieRate,方便数据的排序和序列化

```
package com.ghgj.mr.mymapjoin;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.WritableComparable;
public class MovieRate implements WritableComparable<MovieRate>{
	private String movieid;
	private String userid;
	private int rate;
	private String movieName;
	private String movieType;
	private long ts;
	public String getMovieid() {
		return movieid;
	}
	public void setMovieid(String movieid) {
		this.movieid = movieid;
	}
	public String getUserid() {
```

```
        return userid;
    }
    public void setUserid(String userid) {
        this.userid = userid;
    }
    public int getRate() {
        return rate;
    }
    public void setRate(int rate) {
        this.rate = rate;
    }
    public String getMovieName() {
        return movieName;
    }
    public void setMovieName(String movieName) {
        this.movieName = movieName;
    }
    public String getMovieType() {
        return movieType;
    }
    public void setMovieType(String movieType) {
        this.movieType = movieType;
    }
    public long getTs() {
        return ts;
    }
    public void setTs(long ts) {
        this.ts = ts;
    }
    public MovieRate() {
    }
    public MovieRate(String movieid, String userid, int rate, String movieName,
            String movieType, long ts) {
        this.movieid = movieid;
        this.userid = userid;
        this.rate = rate;
        this.movieName = movieName;
        this.movieType = movieType;
        this.ts = ts;
    }
    @Override
    public String toString() {
        return movieid + "\t" + userid + "\t" + rate + "\t" + movieName
                + "\t" + movieType + "\t" + ts;
```

```
    }
    @Override
    public void write(DataOutput out) throws IOException {
        out.writeUTF(movieid);
        out.writeUTF(userid);
        out.writeInt(rate);
        out.writeUTF(movieName);
        out.writeUTF(movieType);
        out.writeLong(ts);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
        this.movieid = in.readUTF();
        this.userid = in.readUTF();
        this.rate = in.readInt();
        this.movieName = in.readUTF();
        this.movieType = in.readUTF();
        this.ts = in.readLong();
    }
    @Override
    public int compareTo(MovieRate mr) {
        int it = mr.getMovieid().compareTo(this.movieid);
        if(it == 0){
            return mr.getUserid().compareTo(this.userid);
        }else{
            return it;
        }
    }
}
```

第二步：编写 MapReduce 程序

```
package com.ghgj.mr.mymapjoin;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.net.URI;
import java.util.HashMap;
import java.util.Map;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
```

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.filecache.DistributedCache;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MovieRatingMapJoinMR {

	public static void main(String[] args) throws Exception {

		Configuration conf = new Configuration();
		conf.set("fs.defaultFS", "hdfs://hadoop02:9000");
		System.setProperty("HADOOP_USER_NAME","hadoop");
		Job job = Job.getInstance(conf);

//		job.setJarByClass(MovieRatingMapJoinMR.class);
		job.setJar("/home/hadoop/mrmr.jar");

		job.setMapperClass(MovieRatingMapJoinMRMapper.class);
		job.setMapOutputKeyClass(MovieRate.class);
		job.setMapOutputValueClass(NullWritable.class);

//		job.setReducerClass(MovieRatingMapJoinMReducer.class);
//		job.setOutputKeyClass(MovieRate.class);
//		job.setOutputValueClass(NullWritable.class);

		job.setNumReduceTasks(0);

		String minInput = args[0];
		String maxInput = args[1];
		String output = args[2];

		FileInputFormat.setInputPaths(job, new Path(maxInput));
		Path outputPath = new Path(output);
		FileSystem fs = FileSystem.get(conf);
		if(fs.exists(outputPath)){
			fs.delete(outputPath, true);
		}
		FileOutputFormat.setOutputPath(job, outputPath);
```

```
            URI uri = new Path(minInput).toUri();
            job.addCacheFile(uri);

            boolean status = job.waitForCompletion(true);
            System.exit(status?0:1);
        }

        static class MovieRatingMapJoinMRMapper extends Mapper<LongWritable, Text, MovieRate, NullWritable>{
            // 用来存储小份数据的所有解析出来的 key-value
            private static Map<String, Movie> movieMap = new HashMap<String, Movie>();

            @Override
            protected void setup(Context context) throws IOException, InterruptedException {
Path[] localCacheFilePaths = DistributedCache.getLocalCacheFiles(context.getConfiguration());
                String myfilePath = localCacheFilePaths[0].toString();
                System.out.println(myfilePath);
                URI[] cacheFiles = context.getCacheFiles();
                System.out.println(cacheFiles[0].toString());

                BufferedReader br = new BufferedReader(new FileReader(myfilePath.toString()));
                // 此处的 line 就是从文件当中逐行读到的 movie
                String line = "";
                while(null != (line = br.readLine())){
                    String[] splits = line.split("::");
                    movieMap.put(splits[0], new Movie(splits[0], splits[1], splits[2]));
                }
                IOUtils.closeStream(br);
            }

            @Override
            protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
                String[] splits = value.toString().split("::");
                String userid = splits[0];
                String movieid = splits[1];
                int rate = Integer.parseInt(splits[2]);
                long ts = Long.parseLong(splits[3]);
                String movieName = movieMap.get(movieid).getMovieName();
                String movieType = movieMap.get(movieid).getMovieType();
                MovieRate mr = new MovieRate(movieid, userid, rate, movieName, movieType, ts);
                context.write(mr, NullWritable.get());
```

```
            }
        }
}
```