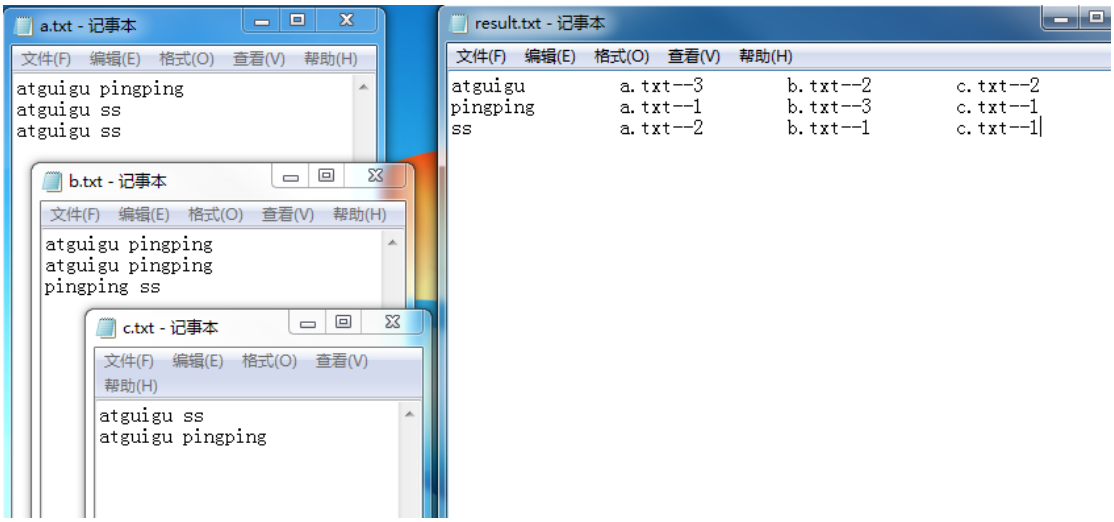
第 7 章 MapReduce 扩展案例

7.1 倒排索引案例（多 job 串联）

1）需求：有大量的文本（文档、网页），需要建立搜索索引



（1）第一次预期输出结果

```
atguigu--a.txt  3
atguigu--b.txt  2
atguigu--c.txt  2
pingping--a.txt  1
pingping--b.txt  3
pingping--c.txt  1
ss--a.txt  2
ss--b.txt  1
ss--c.txt  1
```

（2）第二次预期输出结果

```
atguigu  c.txt-->2  b.txt-->2  a.txt-->3
pingping  c.txt-->1  b.txt-->3  a.txt-->1
ss  c.txt-->1  b.txt-->1  a.txt-->2
```

2）第一次处理

（1）第一次处理，编写 OneIndexMapper

```
package com.atguigu.mapreduce.index;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
```

```
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

public class OneIndexMapper extends Mapper<LongWritable, Text, Text   , IntWritable>{

	String name;
	Text k = new Text();
	IntWritable v = new IntWritable();

	@Override
	protected void setup(Context context)
			throws IOException, InterruptedException {
		// 获取文件名称
		FileSplit split = (FileSplit) context.getInputSplit();

		name = split.getPath().getName();
	}

	@Override
	protected void map(LongWritable key, Text value, Context context)
			throws IOException, InterruptedException {
		// 1 获取 1 行
		String line = value.toString();

		// 2 切割
		String[] fields = line.split(" ");

		for (String word : fields) {
			// 3 拼接
			k.set(word+"--"+name);
			v.set(1);

			// 4 写出
			context.write(k, v);
		}
	}
}
```

（2）第一次处理，编写 OneIndexReducer

```
package com.atguigu.mapreduce.index;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
```

```
public class OneIndexReducer extends Reducer<Text, IntWritable, Text, IntWritable>{

	@Override
	protected void reduce(Text key, Iterable<IntWritable> values,
			Context context) throws IOException, InterruptedException {

		int count = 0;
		// 1 累加求和
		for(IntWritable value: values){
			count +=value.get();
		}

		// 2 写出
		context.write(key, new IntWritable(count));
	}
}
```

（3）第一次处理，编写 OneIndexDriver

```
package com.atguigu.mapreduce.index;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class OneIndexDriver {

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

		args = new String[] { "e:/input/inputoneindex", "e:/output5" };

		Configuration conf = new Configuration();

		Job job = Job.getInstance(conf);
		job.setJarByClass(OneIndexDriver.class);

		job.setMapperClass(OneIndexMapper.class);
		job.setReducerClass(OneIndexReducer.class);

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

		job.setOutputKeyClass(Text.class);
```

```
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
    }
}
```

（4）查看第一次输出结果

```
atguigu--a.txt  3
atguigu--b.txt  2
atguigu--c.txt  2
pingping--a.txt1
pingping--b.txt     3
pingping--c.txt1
ss--a.txt   2
ss--b.txt   1
ss--c.txt   1
```

3）第二次处理

（1）第二次处理，编写 TwoIndexMapper

```
package com.atguigu.mapreduce.index;
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class TwoIndexMapper extends Mapper<LongWritable, Text, Text, Text>{
    Text k = new Text();
    Text v = new Text();

    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {

        // 1  获取 1 行数据
        String line = value.toString();

        // 2 用“--”切割
        String[] fields = line.split("--");

        k.set(fields[0]);
        v.set(fields[1]);
```

```
            // 3 输出数据
            context.write(k, v);
        }
}
```

（2）第二次处理，编写 TwoIndexReducer

```
package com.atguigu.mapreduce.index;
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class TwoIndexReducer extends Reducer<Text, Text, Text, Text> {

    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
            // atguigu a.txt 3
            // atguigu b.txt 2
            // atguigu c.txt 2

            // atguigu c.txt-->2 b.txt-->2 a.txt-->3

            StringBuilder sb = new StringBuilder();
            // 1 拼接
            for (Text value : values) {
                sb.append(value.toString().replace("\t", "-->") + "\t");
            }
            // 2 写出
            context.write(key, new Text(sb.toString()));
        }
}
```

（3）第二次处理，编写 TwoIndexDriver

```
package com.atguigu.mapreduce.index;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class TwoIndexDriver {

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

            args = new String[] { "e:/input/inputtwoindex", "e:/output6" };
```

```
        Configuration config = new Configuration();
        Job job = Job.getInstance(config);

        job.setJarByClass(TwoIndexDriver.class);
        job.setMapperClass(TwoIndexMapper.class);
        job.setReducerClass(TwoIndexReducer.class);

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

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

（4）第二次查看最终结果

```
atguigu   c.txt-->2  b.txt-->2  a.txt-->3
pingping  c.txt-->1  b.txt-->3  a.txt-->1
ss    c.txt-->1  b.txt-->1  a.txt-->2
```

7.2 找博客共同好友案例

1）需求：

以下是博客的好友列表数据，冒号前是一个用户，冒号后是该用户的所有好友（数据中的好友关系是单向的）

friends.txt

求出哪些人两两之间有共同好友，及他俩的共同好友都有谁？

2）需求分析：

先求出 A、B、C、….等是谁的好友

第一次输出结果

```
A    I,K,C,B,G,F,H,O,D,
B    A,F,J,E,
C    A,E,B,H,F,G,K,
```

```
D    G,C,K,A,L,F,E,H,
E    G,M,L,H,A,F,B,D,
F    L,M,D,C,G,A,
G    M,
H    O,
I    O,C,
J    O,
K    B,
L    D,E,
M    E,F,
O    A,H,I,J,F,
```

第二次输出结果

```
A-B E C
A-C D F
A-D E F
A-E D B C
A-F O B C D E
A-G F E C D
A-H E C D O
A-I  O
A-J  O B
A-K D C
A-L  F E D
A-ME F
B-C  A
B-D A E
B-E  C
B-F  E A C
B-G C E A
B-H A E C
B-I   A
B-K C A
B-L  E
B-ME
B-O A
C-D A F
C-E  D
C-F  D A
C-G D F A
C-H D A
C-I   A
C-K A D
C-L  D F
C-MF
```

C-O I A
D-E L
D-F A E
D-G E A F
D-H A E
D-I A
D-K A
D-L E F
D-MF E
D-O A
E-F D M C B
E-G C D
E-H C D
E-J B
E-K C D
E-L D
F-G D C A E
F-H A D O E C
F-I O A
F-J B O
F-K D C A
F-L E D
F-M E
F-O A
G-H D C E A
G-I A
G-K D A C
G-L D F E
G-ME F
G-O A
H-I O A
H-J O
H-K A C D
H-L D E
H-ME
H-O A
I-J O
I-K A
I-O A
K-L D
K-O A
L-M E F

3）代码实现：

（1）第一次 Mapper

```
package com.atguigu.mapreduce.friends;
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class OneShareFriendsMapper extends Mapper<LongWritable, Text, Text, Text>{

    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, Text>.Context context)
            throws IOException, InterruptedException {
        // 1 获取一行 A:B,C,D,F,E,O
        String line = value.toString();

        // 2 切割
        String[] fields = line.split(":");

        // 3 获取 person 和好友
        String person = fields[0];
        String[] friends = fields[1].split(",");

        // 4 写出去
        for(String friend: friends){
            // 输出 <好友，人>
            context.write(new Text(friend), new Text(person));
        }
    }
}
```

（2）第一次 Reducer

```
package com.atguigu.mapreduce.friends;
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class OneShareFriendsReducer extends Reducer<Text, Text, Text, Text>{

    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

        StringBuffer sb = new StringBuffer();
```

```
            //1 拼接
            for(Text person: values){
                sb.append(person).append(",");
            }

            //2 写出
            context.write(key, new Text(sb.toString()));
        }
}
```

（3）第一次 Driver

```
package com.atguigu.mapreduce.friends;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class OneShareFriendsDriver {

    public static void main(String[] args) throws Exception {
        // 1 获取 job 对象
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);

        // 2 指定 jar 包运行的路径
        job.setJarByClass(OneShareFriendsDriver.class);

        // 3 指定 map/reduce 使用的类
        job.setMapperClass(OneShareFriendsMapper.class);
        job.setReducerClass(OneShareFriendsReducer.class);

        // 4 指定 map 输出的数据类型
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);

        // 5 指定最终输出的数据类型
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        // 6 指定 job 的输入原始所在目录
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

```
        // 7 提交
        boolean result = job.waitForCompletion(true);

        System.exit(result?0:1);
    }
}
```

（4）第二次 Mapper

```
package com.atguigu.mapreduce.friends;
import java.io.IOException;
import java.util.Arrays;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class TwoShareFriendsMapper extends Mapper<LongWritable, Text, Text, Text>{

    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        // A I,K,C,B,G,F,H,O,D,
        // 友 人，人，人
        String line = value.toString();
        String[] friend_persons = line.split("\t");

        String friend = friend_persons[0];
        String[] persons = friend_persons[1].split(",");

        Arrays.sort(persons);

        for (int i = 0; i < persons.length - 1; i++) {

            for (int j = i + 1; j < persons.length; j++) {
                // 发出 <人-人，好友> ，这样，相同的“人-人”对的所有好友就会到同
1 个 reduce 中去
                context.write(new Text(persons[i] + "-" + persons[j]), new Text(friend));
            }
        }
    }
}
```

（5）第二次 Reducer

```
package com.atguigu.mapreduce.friends;
import java.io.IOException;
import org.apache.hadoop.io.Text;
```

```
import org.apache.hadoop.mapreduce.Reducer;

public class TwoShareFriendsReducer extends Reducer<Text, Text, Text, Text>{

	@Override
	protected void reduce(Text key, Iterable<Text> values, Context context)
			throws IOException, InterruptedException {

		StringBuffer sb = new StringBuffer();

		for (Text friend : values) {
			sb.append(friend).append(" ");
		}

		context.write(key, new Text(sb.toString()));
	}
}
```

（6）第二次 Driver

```
package com.atguigu.mapreduce.friends;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class TwoShareFriendsDriver {

	public static void main(String[] args) throws Exception {
		// 1 获取 job 对象
		Configuration configuration = new Configuration();
		Job job = Job.getInstance(configuration);

		// 2 指定 jar 包运行的路径
		job.setJarByClass(TwoShareFriendsDriver.class);

		// 3 指定 map/reduce 使用的类
		job.setMapperClass(TwoShareFriendsMapper.class);
		job.setReducerClass(TwoShareFriendsReducer.class);

		// 4 指定 map 输出的数据类型
		job.setMapOutputKeyClass(Text.class);
		job.setMapOutputValueClass(Text.class);
```

```
        // 5 指定最终输出的数据类型
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        // 6 指定 job 的输入原始所在目录
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // 7 提交
        boolean result = job.waitForCompletion(true);
        System.exit(result?0:1);
    }
}
```

7.3 Top10 案例

1）需求：对需求 2.4 输出结果进行加工，输出流量使用量在前 10 的用户信息

2）实现代码

（1）编写 JavaBean 类

```
package com.atguigu.mr;

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

import org.apache.hadoop.io.WritableComparable;

public class FlowBean implements WritableComparable<FlowBean> {

    private Long sumFlow; // 总流量
    private String phoneNum; // 手机号

    // 空参构造
    public FlowBean() {
        super();
    }

    public Long getSumFlow() {
        return sumFlow;
    }

    public void setSumFlow(Long sumFlow) {
        this.sumFlow = sumFlow;
    }

    public String getPhoneNum() {
        return phoneNum;
    }

    public void setPhoneNum(String phoneNum) {
        this.phoneNum = phoneNum;
    }

    @Override
```

```
    public String toString() {
        return sumFlow + "\t" + phoneNum;
    }

    @Override
    public void write(DataOutput out) throws IOException {
        // 序列化
        out.writeLong(sumFlow);
        out.writeUTF(phoneNum);

    }

    @Override
    public void readFields(DataInput in) throws IOException {
        // 反序列化
        this.sumFlow = in.readLong();
        this.phoneNum = in.readUTF();

    }

    @Override
    public int compareTo(FlowBean o) {
        int result;

        result = this.sumFlow.compareTo(o.sumFlow);
        if (result == 0) {
            result = this.phoneNum.compareTo(o.phoneNum);
        }
        return result;
    }
}
```

（2）编写 TopTenMapper 类

```
package com.atguigu.mr;

import java.io.IOException;
import java.util.TreeMap;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class TopTenMapper extends Mapper<LongWritable, Text, NullWritable, Text> {

    // 定义一个 TreeMap 作为存储数据的容器（天然按 key 排序）
    private TreeMap<FlowBean, Text> flowMap = new TreeMap<FlowBean, Text>();

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

```
            FlowBean bean = new FlowBean();
            // 1.获取一行
            String line = value.toString();

            // 2.切割
            String[] fields = line.split("\t");

            long sumFlow = Long.parseLong(fields[3]);

            bean.setSumFlow(sumFlow);
            bean.setPhoneNum(fields[0]);

            // 3.向 TreeMap 中添加数据
            flowMap.put(bean, new Text(value));

            // 4.限制 TreeMap 的数据量，超过 10 条就删除掉流量最小的一条数据
            if (flowMap.size() > 10) {
                flowMap.remove(flowMap.firstKey());
            }

        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            // 输出
            for (Text t : flowMap.values()) {
                context.write(NullWritable.get(), t);
            }
        }
}
```

（3）编写 TopTenReducer 类

```
package com.atguigu.mr;

import java.io.IOException;
import java.util.TreeMap;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class TopTenReducer extends Reducer<NullWritable, Text, NullWritable, Text> {

    @Override
```

```
    protected void reduce(NullWritable key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

        // 1.定义一个 TreeMap 作为存储数据的容器（天然按 key 排序）
        TreeMap<FlowBean, Text> flowMap = new TreeMap<FlowBean, Text>();

        for (Text value : values) {
            FlowBean bean = new FlowBean();
            bean.setPhoneNum(value.toString().split("\t")[0]);
            bean.setSumFlow(Long.parseLong(value.toString().split("\t")[3]));
            flowMap.put(bean, new Text(value));

            // 2.限制 TreeMap 的数据量，超过 10 条就删除掉流量最小的一条数据
            if (flowMap.size() > 10) {
                flowMap.remove(flowMap.firstKey());
            }
        }

        // 3.输出
        for (Text t : flowMap.descendingMap().values()) {
            context.write(NullWritable.get(), t);
        }
    }
}
```

（4）编写驱动类

```
package com.atguigu.mr;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class TopTenDriver {
    public static void main(String[] args)
            throws IllegalArgumentException, IOException, ClassNotFoundException,
InterruptedException {


        // 1 获取配置信息，或者 job 对象实例
```

```
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);

        // 6 指定本程序的 jar 包所在的本地路径
        job.setJarByClass(TopTenDriver.class);

        // 2 指定本业务 job 要使用的 mapper/Reducer 业务类
        job.setMapperClass(TopTenMapper.class);
        job.setReducerClass(TopTenReducer.class);

        // 3 指定 mapper 输出数据的 kv 类型
        job.setMapOutputKeyClass(NullWritable.class);
        job.setMapOutputValueClass(Text.class);

        // 4 指定最终输出的数据的 kv 类型
        job.setOutputKeyClass(NullWritable.class);
        job.setOutputValueClass(Text.class);

        // 5 指定 job 的输入原始文件所在目录
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // 7 将 job 中配置的相关参数，以及 job 所用的 java 类所在的 jar 包， 提交给
yarn 去运行
        boolean result = job.waitForCompletion(true);
        System.exit(result ? 0 : 1);
    }
}
```