Day07_MapReduce的关联

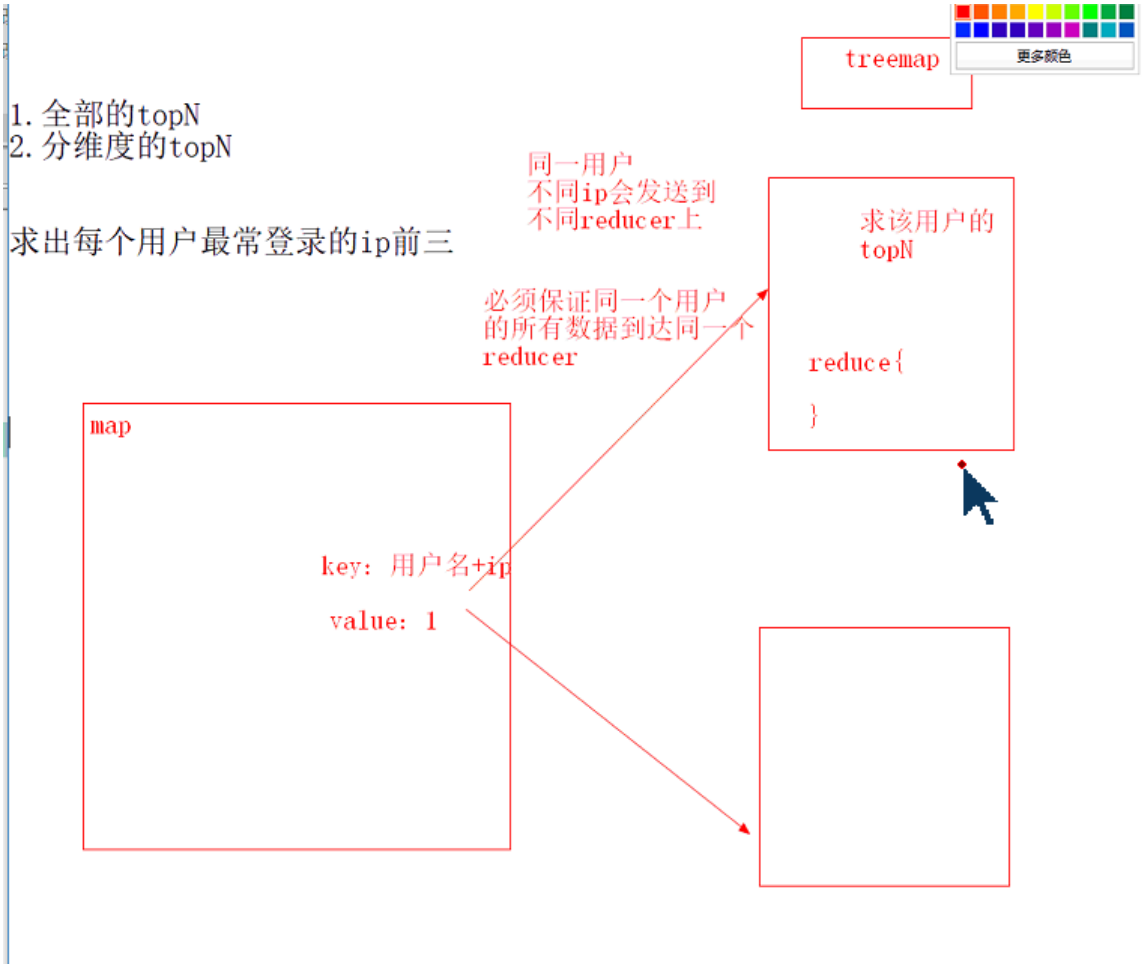
大数据-张军锋 Day07 分维度topN问题 mr关联 mr串联

分维度topN问题

> 分维度求topN是将数据按照某种需求进行分割之后，求出分割之后的topN的数量。例如计算用户常用最常用的三个ip，这个需求就是讲数据按照用户名进行分类，然后求出每一个用户最常用的三个ip地址。满足这个需求只需要将数据按照用户名进行分类，然后将用后面相同的发送到同一个reduce上即可

1. 全部的topN
2. 分维度的topN

求出每个用户最常登录的ip前三



```
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Partitioner;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

/**
* 项目名称：mapreeduce
* 类名称：GroupTopN
* 类描述：求个用户最常登录的ip前三
* @version
*/
public class GroupTopN {
    /**
    * 项目名称：mapreeduce
    * 类名称：GroupTopNMap
    * 类描述：输出(jim_192.168.6.212,1)
    * @version
    */
    public static class GroupTopNMap extends Mapper<LongWritable, T
ext, Text, IntWritable>{
        private String[] infos;
        private Text oKey = new Text();
        private IntWritable ONE = new IntWritable(1);
        @Override
        protected void map(LongWritable key, Text value, Mapper<Lon
gWritable, Text, Text, IntWritable>.Context context)
                throws IOException, InterruptedException {
            infos = value.toString().split("\\s");
            if(infos != null && infos.length > 0 && infos[1].equal
s("login")){
                oKey.set(infos[0] + "_" +infos[2]);
                context.write(oKey, ONE);
            }
        }
```

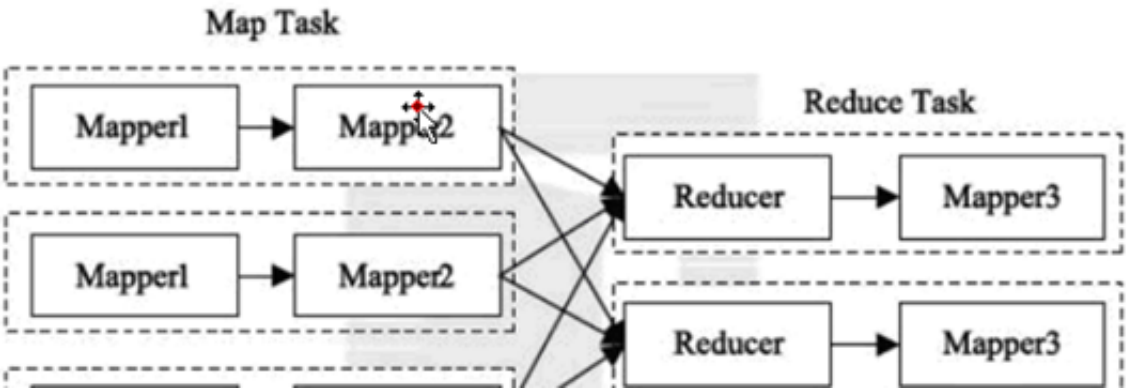
```
    }

    /**
    * 项目名称：mapreeduce
    * 类名称：GroupTopNPartitioner
    * 类描述：将名字相同的，ip地址不同的分到同一个reduce上
    * @version
    */
    public static class GroupTopNPartitioner extends Partitioner<Te
xt, IntWritable>{
        private String[] infos;
        @Override
        public int getPartition(Text key, IntWritable value, int nu
mPartitions) {
            infos = key.toString().split("_");
            return (infos[0].hashCode() & Integer.MAX_VALUE) % numP
artitions;
        }
    }

    /**
    * 项目名称：mapreeduce
    * 类名称：GroupTopNReducer
    * 类描述：计算用户在每一个ip上登录的次数，同时也求topN
    * @version
    */
    public static class GroupTopNReducer extends Reducer<Text, IntW
ritable,Text, IntWritable>{
        private TreeMap<Integer, String> topN;
        private Map<String, Integer> ipLoginTimes;
        private Text oKey = new Text();
        private IntWritable oValue = new IntWritable();
        @Override
        protected void reduce(Text key, Iterable<IntWritable> value
s,
                Reducer<Text, IntWritable, Text, IntWritable>.Conte
xt context) throws IOException, InterruptedException {
            topN = new TreeMap<Integer,String>();
            ipLoginTimes = new HashMap<String,Integer>();
            for (IntWritable value : values) {
                if(ipLoginTimes.containsKey(key.toString())){
                    ipLoginTimes.put(key.toString(), ipLoginTimes.g
et(key.toString()) + value.get());
                }else {
                    ipLoginTimes.put(key.toString(), value.get());
                }
            }

            // 放入topN
```

```
            for (String userIp : ipLoginTimes.keySet()) {
                if(topN.size() < 3){
                    topN.put(ipLoginTimes.get(userIp), userIp);
                }else{
                    topN.put(ipLoginTimes.get(userIp), userIp);
                    topN.remove(topN.firstKey());
                }
            }

            // 输出topN
            for(int times : topN.descendingKeySet()){
                oKey.set(topN.get(times));
                oValue.set(times);
                context.write(oKey, oValue);
            }
        }
    }

    /**
    * 项目名称：mapreeduce
    * 类名称：GroupTopNComparetor
    * 类描述：设置分组，保证用户名相同的，ip不同的进入同一个reduce中
    * @version
    */
    // 方式一
    public static class GroupTopNComparetor extends WritableCompara
tor{
        public GroupTopNComparetor() {
            super(Text.class,true);
        }
        @Override
        public int compare(WritableComparable a, WritableComparable
b) {
            Text ca = (Text)a;
            Text cb = (Text)b;
            return ca.toString().split("_")[0].compareTo(cb.toStrin
g().split("_")[0]);
        }
    }
    // 方式二
/*  public static class GroupTopNComparetor extends Text.Comparato
r{
        @Override
        public int compare(byte[] b1, int s1, int l1, byte[] b2, in
t s2, int l2) {
            byte[] cb1 = Arrays.copyOfRange(b1, 2, b1.length);
            byte[] cb2 = Arrays.copyOfRange(b2, 2, b2.length);
            String str1 = new String(cb1);
            String str2 = new String(cb2);
```

```
                return str1.split("_")[0].compareTo(str2.split("_")
[0]);
            }
        }
*/
    public static void main(String[] args) throws Exception {
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);
        job.setJarByClass(GroupTopN.class);
        job.setJobName("求个用户最常登录的ip前三");

        job.setMapperClass(GroupTopNMap.class);
        job.setPartitionerClass(GroupTopNPartitioner.class);
        job.setReducerClass(GroupTopNReducer.class);

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

        Path inputPath = new Path("/user-logs-large.txt");
        Path outputDir = new Path("/bd14/GroupTopN");
        outputDir.getFileSystem(configuration).delete(outputDir,tru
e);

        FileInputFormat.addInputPath(job, inputPath);
        FileOutputFormat.setOutputPath(job, outputDir);

        job.setGroupingComparatorClass(GroupTopNComparetor.class);
//      job.setNumReduceTasks(2);

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

Mr串联

hadoop的mr作业支持链式处理流程,就好比我们linux中的管道一样,将上次的输出作为下次的输入继续执行操作.
为了解决这类问题,提出了ChainMapper和ChainReduce,这个过程和我们一个map一个reducer的状态是一样的,,下面是具体的实现代码


Map Task
Mapper1
Mapper2
Mapper1
Mapper2
Reduce Task
Reducer
Mapper3
Reducer
Mapper3

```java
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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.chain.ChainMapper;
import org.apache.hadoop.mapreduce.lib.chain.ChainReducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

/**
* 项目名称：mapreeduce
* 类名称：MRChain
* 类描述：
* @version
*/
public class MRChain {

    /**
    * 项目名称：mapreeduce
    * 类名称：MRChainMap1
    * 类描述：map过滤销售数量大于1亿的数据
    * @version
    */
    public static class MRChainMap1 extends Mapper<LongWritable, Te
xt, Text, IntWritable>{
        private Text oKey = new Text();
        private IntWritable oValue = new IntWritable();
        private String[] infos;
        @Override
        protected void map(LongWritable key, Text value, Mapper<Lon
gWritable, Text, Text, IntWritable>.Context context)
                throws IOException, InterruptedException {
            infos = value.toString().split("\\s");
            if(Integer.valueOf(infos[1]) <= 100000000){
                oKey.set(infos[0]);
                oValue.set(Integer.valueOf(infos[1]));
                System.out.println(oKey+"---" + oValue);
                context.write(oKey, oValue);
            }
        }
    }
```

```
    /**
    * 项目名称：mapreeduce
    * 类名称：MRChain2
    * 类描述：过滤掉100-10000之间的数据
    * @version
    */
    public static class MRChainMap2 extends Mapper<Text, IntWritable,Text, IntWritable>{

        @Override
        protected void map(Text key, IntWritable value, Mapper<Text, IntWritable, Text, IntWritable>.Context context)
                throws IOException, InterruptedException {
            if(value.get() <= 100 || value.get() >= 10000){
                context.write(key, value);
            }
        }
    }


    /**
    * 项目名称：mapreeduce
    * 类名称：MRChainReducer
    * 类描述：聚合商品总数量
    * @version
    */
    public static class MRChainReducer extends Reducer<Text, IntWritable, Text, IntWritable>{
        private IntWritable oValue = new IntWritable();
        private int sum;
        @Override
        protected void reduce(Text key, Iterable<IntWritable> values,
                Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException, InterruptedException {
            sum = 0;
            for (IntWritable value : values) {
                sum += value.get();
            }
            oValue.set(sum);
            context.write(key, oValue);
        }
    }

    /**
    * 项目名称：mapreeduce
    * 类名称：MRChainMap3
    * 类描述：商品名称大于3的过滤掉
    * @version
```

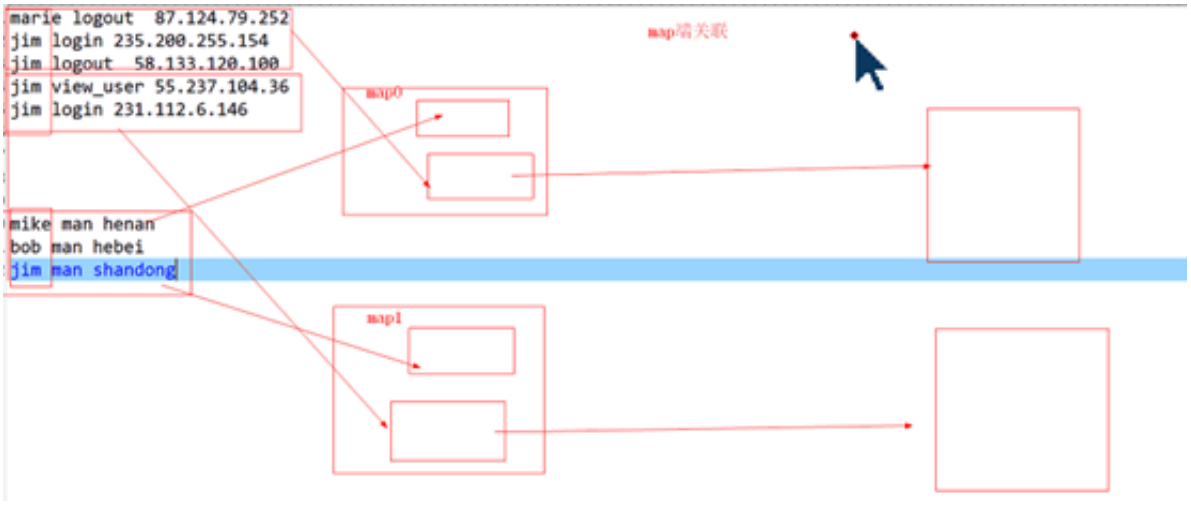
```
     */
    public static class MRChainMap3 extends Mapper<Text, IntWritabl
e, Text, IntWritable>{

        @Override
        protected void map(Text key, IntWritable value, Mapper<Tex
t, IntWritable, Text, IntWritable>.Context context)
                throws IOException, InterruptedException {
            if(key.toString().length() < 3){
                context.write(key, value);
            }
        }
    }


    public static void main(String[] args) throws Exception {
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);
        job.setJarByClass(MRChain.class);
        job.setJobName("mapreduce串联");
        // 设置map端执行
        ChainMapper.addMapper(job, MRChainMap1.class, LongWritabl
e.class, Text.class, Text.class, IntWritable.class, configuration);
        ChainMapper.addMapper(job, MRChainMap2.class, Text.class, I
ntWritable.class,Text.class, IntWritable.class, configuration);
        // 设置reduce端执行
        ChainReducer.setReducer(job, MRChainReducer.class, Text.cla
ss, IntWritable.class, Text.class, IntWritable.class, configuratio
n);
        ChainReducer.addMapper(job, MRChainMap3.class, Text.class,
IntWritable.class, Text.class, IntWritable.class, configuration);

        Path inputPath = new Path("/goods.txt");
        Path outputDir = new Path("/bd14/goods");
        outputDir.getFileSystem(configuration).delete(outputDir,tru
e);
        FileInputFormat.addInputPath(job, inputPath);
        FileOutputFormat.setOutputPath(job, outputDir);

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

mr关联

mr关联有三种形式: map端关联,reduce端关联,semijoin关联.map端关联时效率最高的,因为map端关联,关联在一起的数据小于或者等于没有关联的数据,减少了数据的传输过程和shuffle过程,shuffle过程是最消耗资源的.在reduce端进行关联是最消耗资源的,增大了shuffle过程,但是对于大文件我们只能在reduce端进行关联,另外一个是semijoin,semijoin是对reduce端关联的一种优化.是map端和reduce端的结合体.semijoin是将需要关联的文件中其中一个比较小的文件读取到内存中,原理和map端关联相似，然后将两个文件中的数据和内存中的数据进行比较，如果内存中存在，则表示是需要关联的数据，否则，不能关联，但同时引入了一个问题，如果连个需要关联的数据大小差不多，并且很大，使用semijoin进行关联，使效率更低，所以semijoin比较适合一个大文件，一个相对比较小的文件之间的关联

map端关联

map端关联的特点是一个小文件和一个大文件之间的关联，将小文件读取到内存中，然后读取大文件，看大文件中的关联项是否在内存中存在，如果存在，则可以关联，不存在，不能关联



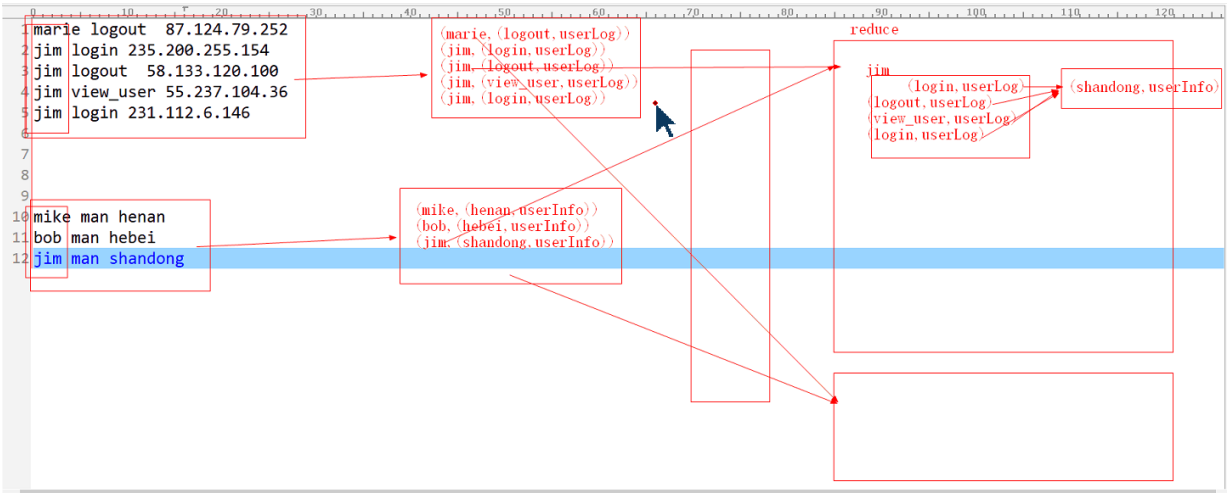
```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;
import java.util.HashMap;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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;

/**
* 项目名称：mapreeduce
* 类名称：MapJoin
* 类描述：Map端关联，计算每个省份的用户对系统的访问次数
* @version
*/
public class MapJoin {
    /**
    * 项目名称：mapreeduce
    * 类名称：MapJoinMap
    * 类描述：map端读取分布式缓存文件，把它加载到一个hashmap中关联字段作为ke
y，计算相关字段值作为value
    * @version
    */
    public static class MapJoinMap extends Mapper<LongWritable, Tex
t, Text, IntWritable> {
        private String[] infos;
        private HashMap<String, String> userInfos = new HashMap<>
();
        private Text okey = new Text();
        private IntWritable ONE = new IntWritable(1);

        // 读取分布式缓存文件，将数据放入hashmap中
        @Override
        protected void setup(Mapper<LongWritable, Text, Text, IntWr
itable>.Context context)
                throws IOException, InterruptedException {
            // 获取分布式缓存文件路径
            URI[] cacheFiles = context.getCacheFiles();
```

```
            FileSystem fileSystem = FileSystem.get(context.getConfi
guration());
            for (URI uri : cacheFiles) {
                if (uri.toString().contains("user_info.txt")) {
                    FSDataInputStream inputStream = fileSystem.ope
n(new Path(uri));
                    InputStreamReader inputStreamReader = new Input
StreamReader(inputStream, "UTF-8");
                    BufferedReader bufferedReader = new BufferedRea
der(inputStreamReader);
                    String line = bufferedReader.readLine();
                    while (line != null) {
                        infos = line.split("\\s");
                        userInfos.put(infos[0], infos[2]);
                        line = bufferedReader.readLine();
                    }
                }
            }
        }

        // map方法中处理大表数据，每处理一条就取出关联字段，看hashmap中是否存
在，存在代表关联，不存在代表关联不上。输出(shangdong,1)
        @Override
        protected void map(LongWritable key, Text value, Mapper<Lon
gWritable, Text, Text, IntWritable>.Context context)
                throws IOException, InterruptedException {
            infos = value.toString().split("\\s");
            if (userInfos.containsKey(infos[0])) {
                okey.set(userInfos.get(infos[0]));
                context.write(okey, ONE);
            }
        }
    }

    /**
    * 项目名称：mapreeduce
    * 类名称：MapJoinReducer
    * 类描述：聚合求值
    * @version
    */
    public static class MapJoinReducer extends Reducer<Text, IntWri
table, Text, IntWritable> {
        private IntWritable oValue = new IntWritable();
        private int sum;

        @Override
        protected void reduce(Text key, Iterable<IntWritable> value
s,
                Reducer<Text, IntWritable, Text, IntWritable>.Conte
```

```
xt context)
                throws IOException, InterruptedException {
            sum = 0;
            for (IntWritable value : values) {
                sum += value.get();
            }
            oValue.set(sum);
            context.write(key, oValue);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);
        job.setJarByClass(MapJoin.class);
        job.setJobName("map端聚合");

        job.setMapperClass(MapJoinMap.class);
        job.setReducerClass(MapJoinReducer.class);

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

        // 设置分布式缓存文件(小表)
        Path cacheFilePath = new Path("/user_info.txt");
        job.addCacheFile(cacheFilePath.toUri());

        // 大表
        Path inputPath = new Path("/user-logs-large.txt");
        Path outputDir = new Path("/bd14/MapJoin");
        outputDir.getFileSystem(configuration).delete(outputDir, tr
ue);
        FileInputFormat.addInputPath(job, inputPath);
        FileOutputFormat.setOutputPath(job, outputDir);

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

reduce端关联

reduce端关联的特点是大文件与大文件之间的关联，这个过程是非常消耗资源的，但是不这么做，也没有其他的什么办法。对于reduce端进行关联的原理是这样的：首先map端读取文件，并且将读取的kv打上标记，目的就是为了确定文件的出处，可以是文件名，也可以是表名，以及其他的区分标记都是可以的，然后将数据发送到reduce端，在reduce端，reduce接收到数据之后，按照标记将数据分成两个部分，然后将这俩部分做笛卡尔乘积，得到的结果就是关联后的结果



```java
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
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.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


/**
* 项目名称：mapreeduce
* 类名称：ReduceJoin
* 类描述：计算每个省份的用户对系统的访问次数
* @version
*/
public class ReduceJoin {
    /**
    * 项目名称：mapreeduce
    * 类名称：ValueWithFlag
    * 类描述：定义表示类型,为了序列化
    * @version
    */
    public static class ValueWithFlag implements Writable{
        private String value;
        private String flag;
        public String getValue() {
            return value;
        }
        public void setValue(String value) {
            this.value = value;
        }
        public String getFlag() {
            return flag;
        }
        public void setFlag(String flag) {
            this.flag = flag;
        }
        @Override
```

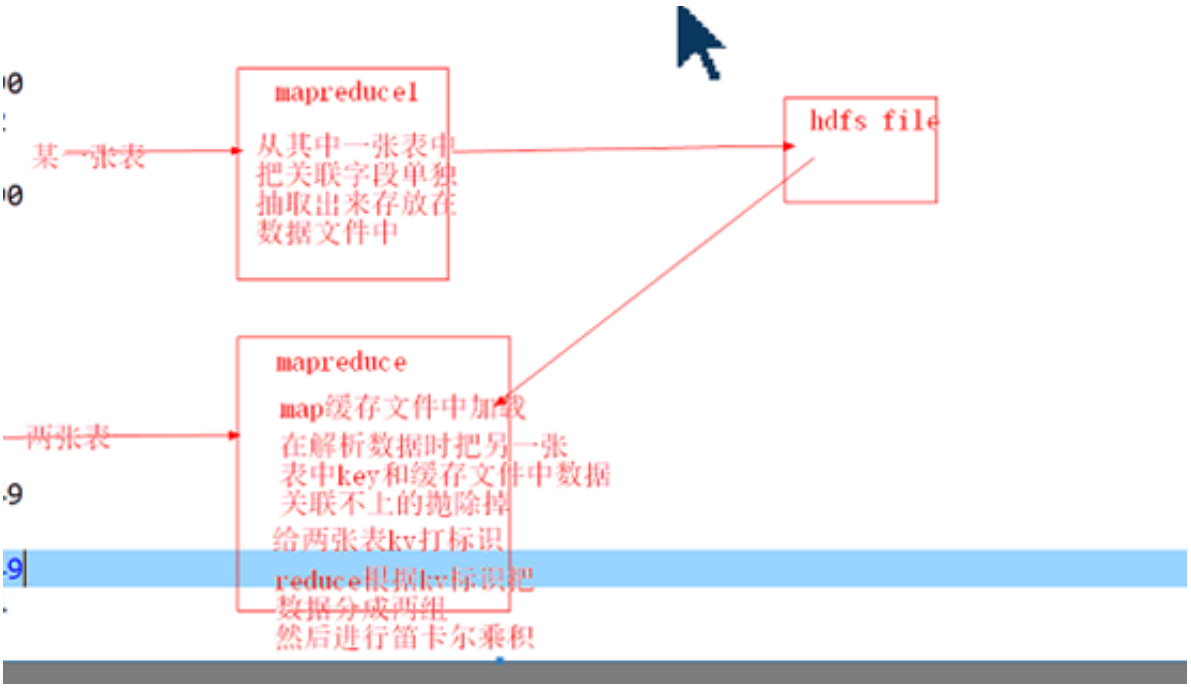
```
        public void write(DataOutput out) throws IOException {
            out.writeUTF(value);
            out.writeUTF(flag);
        }
        @Override
        public void readFields(DataInput in) throws IOException {
            this.value = in.readUTF();
            this.flag = in.readUTF();
        }
    }

    /**
    * 项目名称：mapreeduce
    * 类名称：ReduceJoinMap
    * 类描述：读取两个文件，根据来源把每一个kv打上标签输出给reduce，key必须是关联字段
    * @version
    */
    public static class ReduceJoinMap extends Mapper<LongWritable, Text, Text, ValueWithFlag>{
        private String[] infos;
        private Text outkey = new Text();
        private ValueWithFlag outValue = new ValueWithFlag();
        private FileSplit inputSplit;
        private String fileName;

        // 获取文件名，给文件添加自定义文件名标记,setup方法在执行map工作时执行
        @Override
        protected void setup(Mapper<LongWritable, Text, Text, ValueWithFlag>.Context context)
                throws IOException, InterruptedException {
            inputSplit = (FileSplit) context.getInputSplit();
            if(inputSplit.getPath().toString().contains("user-logs-large.txt")){
                fileName = "userLogsLarge";
            }else if(inputSplit.getPath().toString().contains("user_info.txt")){
                fileName = "userInfo";
            }
        }

        @Override
        protected void map(LongWritable key, Text value,
                Mapper<LongWritable, Text, Text, ValueWithFlag>.Context context)
                throws IOException, InterruptedException {

            outValue.setFlag(fileName);
```

```
            infos = value.toString().split("\\s");
            if(fileName.equals("userLogsLarge")){
                // 解析user-logs-large.txt的过程(用户名，行为类型，IP地
址)
                outkey.set(infos[0]);
                outValue.setValue(infos[1] + "\t" + infos[2]);
            }else if(fileName.equals("userInfo")){
                // 解析user_info.txt的过程(用户名，性别，省份)
                outkey.set(infos[0]);
                outValue.setValue(infos[1] + "\t" + infos[2]);
            }
            context.write(outkey, outValue);
        }
    }

    /**
    * 项目名称：mapreeduce
    * 类名称：ReduceJoinReducer
    * 类描述：接收map发送过来的数据，根据value中的flag来把相同key对应的value
分成两组
    * 那么两组中的数据就是来自两张表中的数据，对这两组数据做笛卡尔乘积即完成关联
    * @version
    */
    public static class ReduceJoinReducer extends Reducer<Text, Val
ueWithFlag, Text,Text>{
        private List<String> userLogsLargeList;
        private List<String> userInfoList;
        private Text outValue = new Text();

        @Override
        protected void reduce(Text key, Iterable<ValueWithFlag> val
ues,
                Reducer<Text, ValueWithFlag, Text, Text>.Context co
ntext) throws IOException, InterruptedException {
            userLogsLargeList = new ArrayList<>();
            userInfoList = new ArrayList<>();
            for(ValueWithFlag value : values){
                if(value.getFlag().equals("userLogsLarge")){
                    userLogsLargeList.add(value.getValue());
                }else if (value.getFlag().equals("userInfo")) {
                    userInfoList.add(value.getValue());
                }
            }
            // 对两组中的数据进行笛卡尔乘积
            for(String userLogsLarge : userLogsLargeList){
                for (String userInfo : userInfoList) {
                    outValue.set(userLogsLarge + "\t" + userInfo);
                    context.write(key, outValue);
                }
```

```
            }
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);
        job.setJarByClass(ReduceJoin.class);
        job.setJobName("Reduce端关联");

        job.setMapperClass(ReduceJoinMap.class);
        job.setReducerClass(ReduceJoinReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(ValueWithFlag.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path("/user-logs-large.txt"));
        FileInputFormat.addInputPath(job, new Path("/user_info.txt"));
        Path outputDir = new Path("/bd14/ReduceJoin");
        outputDir.getFileSystem(configuration).delete(outputDir,true);
        FileOutputFormat.setOutputPath(job, outputDir);

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

半关联(semijoin)

> 半关联是对reduce端的一种优化。它适用于两张表关联程度比较低的情况，关联程度不高时，等于在map端给文件做了一次赛选，减少了reduce端的压力。它的基本原理是这样的，它先从一个相对比较小的文件中，把关联字段提取出来，存放在数据文件中，Map将关联字段读取到缓存中，加载另外一个文件时，如果缓存中存在，则可以关联，如果不存在，则不可以进行关联，并且将这两个表kv打上标记，将数据发送到reduce端，reduce接收数据，根据标识将数据分成两个部分，然后做笛卡尔乘积，关联的结果就是做完笛卡尔乘积的结果


mapreduce1
从其中一张表中
把关联字段单独
抽取出来存放在
数据文件中
某一张表
hdfs file
mapreduce
map缓存文件中加载
在解析数据时把另一张
表中key和缓存文件中数据
关联不上的抛除掉
两张表
给两张表kv打标识
reduce根据kv标识把
数据分成两组
然后进行笛卡尔乘积

```java
// 代码一，抽取关联字段

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
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.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


/**
* 项目名称：mapreeduce
* 类名称：ReduceJoin
* 类描述：计算每个省份的用户对系统的访问次数
* @version
*/
public class ReduceJoin {
    /**
    * 项目名称：mapreeduce
    * 类名称：ValueWithFlag
    * 类描述：定义表示类型,为了序列化
    * @version
    */
    public static class ValueWithFlag implements Writable{
        private String value;
        private String flag;
        public String getValue() {
            return value;
        }
        public void setValue(String value) {
            this.value = value;
        }
        public String getFlag() {
            return flag;
        }
        public void setFlag(String flag) {
            this.flag = flag;
        }
```

```
        @Override
        public void write(DataOutput out) throws IOException {
            out.writeUTF(value);
            out.writeUTF(flag);
        }
        @Override
        public void readFields(DataInput in) throws IOException {
            this.value = in.readUTF();
            this.flag = in.readUTF();
        }
    }

    /**
    * 项目名称：mapreeduce
    * 类名称：ReduceJoinMap
    * 类描述：读取两个文件，根据来源把每一个kv打上标签输出给reduce，key必须是关联字段
    * @version
    */
    public static class ReduceJoinMap extends Mapper<LongWritable, Text, Text, ValueWithFlag>{
        private String[] infos;
        private Text outkey = new Text();
        private ValueWithFlag outValue = new ValueWithFlag();
        private FileSplit inputSplit;
        private String fileName;

        // 获取文件名，给文件添加自定义文件名标记,setup方法在执行map工作时执行
        @Override
        protected void setup(Mapper<LongWritable, Text, Text, ValueWithFlag>.Context context)
                throws IOException, InterruptedException {
            inputSplit = (FileSplit) context.getInputSplit();
            if(inputSplit.getPath().toString().contains("user-logs-large.txt")){
                fileName = "userLogsLarge";
            }else if(inputSplit.getPath().toString().contains("user_info.txt")){
                fileName = "userInfo";
            }
        }

        @Override
        protected void map(LongWritable key, Text value,
                Mapper<LongWritable, Text, Text, ValueWithFlag>.Context context)
                throws IOException, InterruptedException {
```

```
            outValue.setFlag(fileName);
            infos = value.toString().split("\\s");
            if(fileName.equals("userLogsLarge")){
                // 解析user-logs-large.txt的过程(用户名，行为类型，IP地
址)
                outkey.set(infos[0]);
                outValue.setValue(infos[1] + "\t" + infos[2]);
            }else if(fileName.equals("userInfo")){
                // 解析user_info.txt的过程(用户名，性别，省份)
                outkey.set(infos[0]);
                outValue.setValue(infos[1] + "\t" + infos[2]);
            }
            context.write(outkey, outValue);
        }
    }

    /**
    * 项目名称：mapreeduce
    * 类名称：ReduceJoinReducer
    * 类描述：接收map发送过来的数据，根据value中的flag来把相同key对应的value
分成两组
    * 那么两组中的数据就是来自两张表中的数据，对这两组数据做笛卡尔乘积即完成关联
    * @version
    */
    public static class ReduceJoinReducer extends Reducer<Text, Val
ueWithFlag, Text,Text>{
        private List<String> userLogsLargeList;
        private List<String> userInfoList;
        private Text outValue = new Text();

        @Override
        protected void reduce(Text key, Iterable<ValueWithFlag> val
ues,
                Reducer<Text, ValueWithFlag, Text, Text>.Context co
ntext) throws IOException, InterruptedException {
            userLogsLargeList = new ArrayList<>();
            userInfoList = new ArrayList<>();
            for(ValueWithFlag value : values){
                if(value.getFlag().equals("userLogsLarge")){
                    userLogsLargeList.add(value.getValue());
                }else if (value.getFlag().equals("userInfo")) {
                    userInfoList.add(value.getValue());
                }
            }
            // 对两组中的数据进行笛卡尔乘积
            for(String userLogsLarge : userLogsLargeList){
                for (String userInfo : userInfoList) {
                    outValue.set(userLogsLarge + "\t" + userInfo);
                    context.write(key, outValue);
```

```
                }
            }
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);
        job.setJarByClass(ReduceJoin.class);
        job.setJobName("Reduce端关联");

        job.setMapperClass(ReduceJoinMap.class);
        job.setReducerClass(ReduceJoinReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(ValueWithFlag.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path("/user-logs-large.txt"));
        FileInputFormat.addInputPath(job, new Path("/user_info.txt"));
        Path outputDir = new Path("/bd14/ReduceJoin");
        outputDir.getFileSystem(configuration).delete(outputDir,true);
        FileOutputFormat.setOutputPath(job, outputDir);

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

```java
// 代码二 半连接

import java.io.BufferedReader;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
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.io.Writable;
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.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class SeniTest {
    public static class ValueWithFlag implements Writable {
        private String value;
        private String flag;

        public String getValue() {
            return value;
        }

        public void setValue(String value) {
            this.value = value;
        }

        public String getFlag() {
            return flag;
        }

        public void setFlag(String flag) {
            this.flag = flag;
        }
```

```
        @Override
        public void write(DataOutput out) throws IOException {
            out.writeUTF(value);
            out.writeUTF(flag);
        }

        @Override
        public void readFields(DataInput in) throws IOException {
            this.value = in.readUTF();
            this.flag = in.readUTF();
        }
    }

    public static class SeniTestMap extends Mapper<LongWritable, Te
xt, Text, ValueWithFlag> {
        private String[] infos;
        private HashMap<String, String> userInfos = new HashMap<>
();
        private Text outkey = new Text();
        private FileSplit inputSplit;
        private String fileName;
        private ValueWithFlag outValue = new ValueWithFlag();

        @Override
        protected void setup(Mapper<LongWritable, Text, Text, Value
WithFlag>.Context context)
                throws IOException, InterruptedException {
            // 获取分布式缓存文件路径
            URI[] cacheFiles = context.getCacheFiles();
            FileSystem fileSystem = FileSystem.get(context.getConfi
guration());
            for (URI uri : cacheFiles) {
                if (uri.toString().contains("/bd14/ExtractData/par
t-r-00000")) {
                    FSDataInputStream inputStream = fileSystem.ope
n(new Path(uri));
                    InputStreamReader inputStreamReader = new Input
StreamReader(inputStream, "UTF-8");
                    BufferedReader bufferedReader = new BufferedRea
der(inputStreamReader);
                    String line = bufferedReader.readLine();
                    while (line != null) {
                        infos = line.split("\\s");
                        userInfos.put(infos[0], infos[1]);
                        line = bufferedReader.readLine();
                    }
                }
            }
```

```
            inputSplit = (FileSplit) context.getInputSplit();
            if (inputSplit.getPath().toString().contains("user-log
s-large.txt")) {
                fileName = "userLogsLarge";
            } else if (inputSplit.getPath().toString().contains("us
er_info.txt")) {
                fileName = "userInfo";
            }

        }

        @Override
        protected void map(LongWritable key, Text value,
                Mapper<LongWritable, Text, Text, ValueWithFlag>.Con
text context)
                throws IOException, InterruptedException {
            infos = value.toString().split("\\s");

            if (userInfos.containsKey(infos[0])) {
                outValue.setFlag(fileName);
                infos = value.toString().split("\\s");
                if (fileName.equals("userLogsLarge")) {
                    // 解析user-logs-large.txt的过程(用户名，行为类型，I
P地址)
                    outkey.set(infos[0]);
                    outValue.setValue(infos[1] + "\t" + infos[2]);
                } else if (fileName.equals("userInfo")) {
                    // 解析user_info.txt的过程(用户名，性别，省份)
                    outkey.set(infos[0]);
                    outValue.setValue(infos[1] + "\t" + infos[2]);
                }
                context.write(outkey, outValue);

            }
        }

    }

    public static class SeniTestReducer extends Reducer<Text, Value
WithFlag, Text, Text> {

        private List<String> userLogsLargeList;
        private List<String> userInfoList;
        private Text outValue = new Text();

        @Override
        protected void reduce(Text key, Iterable<ValueWithFlag> val
ues,
                Reducer<Text, ValueWithFlag, Text, Text>.Context co
```

```
ntext) throws IOException, InterruptedException {
            userLogsLargeList = new ArrayList<>();
            userInfoList = new ArrayList<>();
            for (ValueWithFlag value : values) {
                if (value.getFlag().equals("userLogsLarge")) {
                    userLogsLargeList.add(value.getValue());
                } else if (value.getFlag().equals("userInfo")) {
                    userInfoList.add(value.getValue());
                }
            }
            // 对两组中的数据进行笛卡尔乘积
            for (String userLogsLarge : userLogsLargeList) {
                for (String userInfo : userInfoList) {
                    outValue.set(userLogsLarge + "\t" + userInfo);
                    context.write(key, outValue);
                }
            }
        }

    }

    public static void main(String[] args) throws Exception {
        Configuration configuration = new Configuration();
        Job job = Job.getInstance(configuration);
        job.setJarByClass(SeniTest.class);
        job.setJobName("Reduce端关联SeniTest");

        job.setMapperClass(SeniTestMap.class);
        job.setReducerClass(SeniTestReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(ValueWithFlag.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        // 设置分布式缓存文件(小表)
        Path cacheFilePath = new Path("/bd14/ExtractData/part-r-000
00");
        job.addCacheFile(cacheFilePath.toUri());

        FileInputFormat.addInputPath(job, new Path("/user-logs-larg
e.txt"));
        FileInputFormat.addInputPath(job, new Path("/user_info.tx
t"));
        Path outputDir = new Path("/bd14/SeniTest");
        outputDir.getFileSystem(configuration).delete(outputDir, tr
ue);
        FileOutputFormat.setOutputPath(job, outputDir);
```

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