《大数据综合处理实验课程设计报告》

181860076 荣毅 952713875@qq.com
171830591 胡庆国 1935035757@qq.com
date: 2021.07.13

《大数据综合处理实验课程设计报告》

一、分工说明

荣毅：任务一、任务二、任务三和任务四的实现，程序的本地测试，相应部分实验报告的撰写

胡庆国：任务五的实现，程序的在线调试，相应部分实验报告的撰写

二、课程设计题目——哈利波特的魔法世界

2.1 任务一：数据预处理

2.1.1 任务要求

要求从给定的哈利波特文集中切分出人名，屏蔽与人名无关的信息。

输入：哈利波特全集、人名列表文件

输出：哈利波特全集中的人名文件

```
样例输入：
哈利凝视着邓布利多那双浅蓝色的眼睛，他真正想问的话一下子脱口而出。

样例输出：
哈利,邓布利多 1
```

2.1.2 任务分析

首先进行可并行性分析。由于任务实际上是由非常多的小任务构成（这样的小任务：从一个段落中挑出人名并输出），这些小任务间彼此没有依赖，因此十分适合采用并行框架进行计算。而根据后续的文本分析任务得出，我们对于段落中相同的人名列表只需要知道数量就可以，因此可以进行相同的人名列表的合并。

接下来进行map和reduce接口的设计，map接受一段文本，输出其中的人名列表，并计数为1；reduce接受人名列表，输出人名列表和对应数量。

接口	**map**	**reduce**
输入	<Object,Text>	<namelist1,1>、<namelist2,1>……
输出	<namelist1,1>、<namelist2,1>……	<namelist1,n_1>、<namelist2,n_2>……

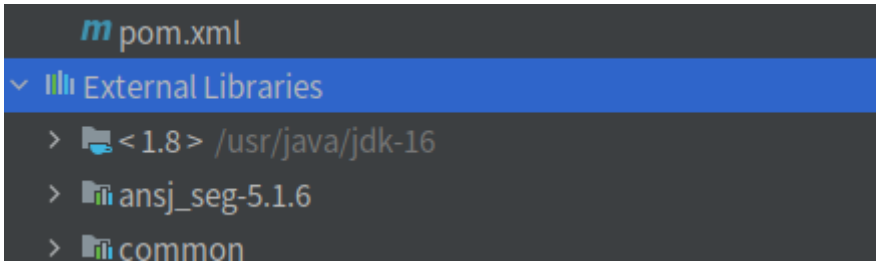
例如：

一个map输出为：<[哈利,罗恩,赫敏],1>、<[哈利,罗恩,赫敏],1>、<[哈利,赫敏],1>

那么对应地在reduce阶段输出：<[哈利,罗恩,赫敏],2>、<[哈利,赫敏],1>

2.1.3 任务过程

根据提示，选择ansj_seg工具对文本进行分词处理。我们选择ansj_seg5.1.6版本，在项目文件中导入jar包：




```
import org.ansj.domain.Term;
import org.ansj.library.DicLibrary;
```

首先我们要进行人名词库的构建，阅读ansj_seq的接口要求，我们只需要向默认的dictionary中循环添加我们的人名词汇即可：

```
DicLibrary.insert(DicLibrary.DEFAULT, line); //其中line为从文件中循环读入的人名，共计800+人名
```

然后进行分词处理，分词的语句参考ansj_seq的接口要求：

```
List<Term> terms = DicAnalysis.parse(line).getTerms();//这一行将会把段落转为一个个的词
```

对于这个terms的list，我们把不需要的词进行过滤，留下我们要的人名就可以了。ansj_seq工具会对我们人工定义的词库进行一个标记userDefine，我们需要将这些terms依次判断是否带有userDefine标记，留下带有标记的就是人名了。

```
//用于过滤无关分词，留下人名
for(Term tmp:terms){
    if(tmp.getNatureStr().equals("userDefine")){
        termSet.add(tmp.getName());
    }
}
```

2.1.4 任务的难点及攻克

上述描述中有一处隐藏了一个优化问题，描述如下：

对于 <[哈利,罗恩,赫敏],1>、<[哈利,赫敏,罗恩],1>，在reduce阶段应当被合并为 <[哈利,赫敏,罗恩],2>，而**进行集合的比较是一个不容忽视的开销**，因此我们不得不找到一个算法可以避免进行集合的比较。

一个简答且有效的做法是，在map阶段的namelist构建阶段采取一种策略使得“ [哈利,罗恩,赫敏]”的等价类均表现为“ [哈利,罗恩,赫敏]”而非其他。因此需要引入一个Hashset，我们知道Hashset的内部是红黑树维持序的，因此先向Hashset中置入元素，然后依次取出，这样互为等价类的集合表现顺序就是相同的。

2.1.5 代码展示

map

```
public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
    String line = value.toString();
    List<Term> terms = DicAnalysis.parse(line).getTerms();
    HashSet<String> termSet = new HashSet<>();
    for (Term tmp : terms) {
```

```
            String word = terms.get(i).getName(); //拿到词
            String natureStr = terms.get(i).getNatureStr(); //拿到词性
            if (natureStr.equals("userDefine")) {
                termSet.add(word)
            }
        }
        StringBuilder namelist = new StringBuilder();
        for (String tmp : termSet) {
            namelist.append(tmp);
            namelist.append(" ");
        }
        if (namelist.length() > 0) {
            String nl = namelist.toString().trim();
            context.write(new Text(nl), one);
        }
    }
```

reduce

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

2.1.6 部分结果展示

2.2 任务二：人物同现统计

2.2.1 任务要求

对人物之间的同现关系次数进行统计。

输入：任务1的输出

输出：同现关系矩阵

```
样例输入：
哈利,邓布利多 2
哈利,邓布利多,罗恩,赫敏 1

样例输出：
<哈利,邓布利多> 3
<哈利,罗恩> 1
<哈利,赫敏> 1
<邓布利多,哈利> 3
<邓布利多,罗恩> 1
<邓布利多,赫敏> 1
<赫敏,哈利> 1
<赫敏,邓布利多> 1
<赫敏,罗恩> 1
<罗恩,哈利> 1
<罗恩,邓布利多> 1
<罗恩,赫敏> 1
```

2.2.2 任务分析

首先进行可并行性分析。同现任务是经典的可并行任务，可以拆分成的最小任务单位描述如下：

对任务一的每一行输出结果进行不同人名之间两两组合，将组合结果与该组合一共出现的次数进行统计求和
并输出。

2.2.3 任务过程

只需要在map阶段进行list中元素两两配对计算，然后再reduce阶段进行结果的合并即可。

2.2.4 任务的难点及攻克

这个任务的难点不在于算法，而在于一个细节问题。在作业描述中我们得知，主角团的名字有可能会根据情况产生不同的变化。例如：哈利、哈利波特、哈利·波特。

这里我们对原来的person_name_list进行一个简单的人工处理，将哈利、哈利波特、哈利·波特都映射为同一个数字，在同现矩阵计算的时候，如果我们一行中有同一人物的不同昵称，我们进行一个判断然后将不合理的同现选项删除（例如<哈利, 哈利波特> 1 应当予以删除）

2.2.5 代码展示

setup

```
public void setup(Context context) {
    try {
        Configuration tmpConf = new Configuration();
        FileSystem fs = FileSystem.get(tmpConf);

        Path sameperson = new Path("/home/rycbe/final/sameperson.txt");
        BufferedReader samebr = new BufferedReader(new
InputStreamReader(fs.open(sameperson)));
        String line = samebr.readLine();
        while (line != null) {
            String[] tmp = line.split(",");
            H.put(tmp[0],Integer.parseInt(tmp[1]));
```

```
            line = samebr.readLine();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

map

```
public void map(Object key,Text value,Context context) throws
IOException,InterruptedException{
    String[] tmp = value.toString().split(",");
    IntWritable count = new IntWritable(Integer.parseInt(tmp[1]));
    String[] words = tmp[0].split(" ");

    for(int i=0;i<words.length;++i){
        for(int j=0;j<words.length;++j){
            if(words[i].equals(words[j]) || H.get(words[i])==H.get(words[j]))
continue;
            wordPair.set("<"+words[i]+","+words[j]+">");
            context.write(wordPair,count);
        }
    }
}
```

reduce

```
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);
}
```

2.2.6 部分结果展示

2.3 任务三：人物关系图构建及特征归一化

2.3.1 任务要求

任务实际上的要求是将任务二的共现矩阵转为邻接表，将出现次数按照比例进行一个边权的确定。

任务输入：任务二的输出

任务输出：以共现次数为基础的邻接表

```
样例输入：
<乔治,厄尼·麦克米兰>	1
<乔治,厄恩·普兰>	1
<乔治,史密斯>	1
<厄尼·麦克米兰,乔治>	1
<厄恩·普兰,乔治>	1
<史密斯,乔治>	1

样例输出：
乔治 厄尼·麦克米兰，0.333333;厄恩·普兰0.333333;史密斯,0.333333
厄尼·麦克米兰 乔治，1
厄恩·普兰 乔治，1
史密斯 乔治，1
```

2.3.2 任务分析

首先进行可并行性分析，显然对于每个人物都是一个独立的小任务，完全可以并行。而且算法也很简单，只需要进行求和之后用原共现次数除以和就得到了邻接表边的权重。

2.3.3 任务过程

首先map将同一个人物进行归并。

例如：

<乔治,厄尼·麦克米兰> 1
<乔治,厄恩·普兰> 1
<乔治,史密斯> 1
<厄尼·麦克米兰,乔治> 1
<厄恩·普兰,乔治> 1
<史密斯,乔治> 1

经过map之后应当为：

乔治 厄尼·麦克米兰,1;厄恩·普兰,1;史密斯,1
厄尼·麦克米兰 乔治,1
厄恩·普兰 乔治,1
史密斯 乔治,1

然后再Reduce阶段进行计算：

乔治 厄尼·麦克米兰，0.333333;厄恩·普兰0.333333;史密斯,0.333333
厄尼·麦克米兰 乔治，1
厄恩·普兰 乔治，1
史密斯 乔治，1

2.3.4 任务的难点及攻克

在Reduce阶段出现了一个细节性的问题。也即Iterable对象在进行遍历的时候无法进行第二遍的遍历（根据算法需要，首先进行求和是一次遍历，然后需要计算比例又是一次遍历），因此我们需要在第一遍遍历的时候用字符串记录内容。具体而言就是用StringBuilder记录共现值，最后在第二遍遍历的时候用StringBuilder中的内容而不是用Iterable对象遍历。

2.3.5 代码展示

map

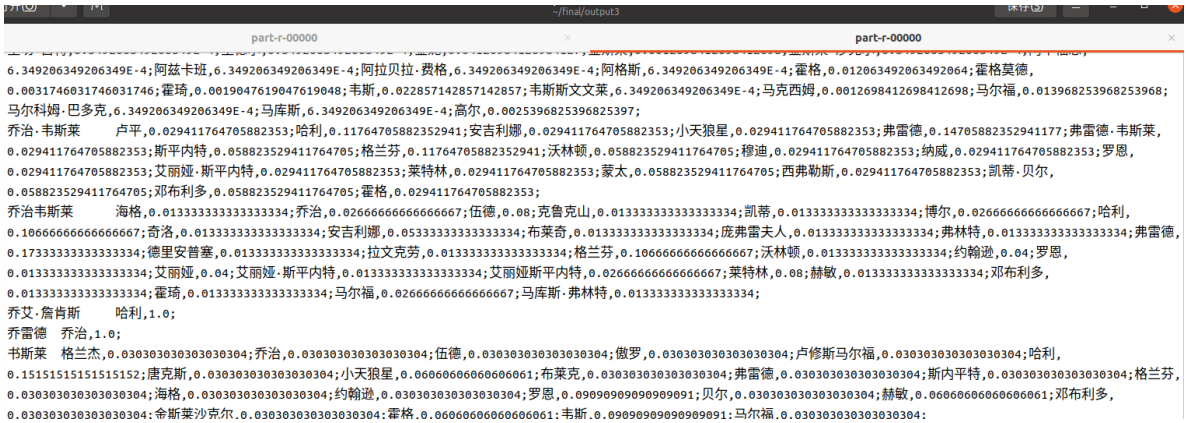
```
public void map(Text key, Text value, Context context)
    throws IOException, InterruptedException{
    String pair = key.toString();
    String val = value.toString();

    String first = pair.substring(1,pair.indexOf(","));
    String second = pair.substring(pair.indexOf(",")+1, pair.indexOf(">"));

    context.write(new Text(first),new Text(second+","+val));

}
```

reduce

```
public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException{
    double sum = 0;
    StringBuilder res = new StringBuilder();
```

```
        for(Text val:values){
            String[] sval = val.toString().split(",");
            sum += Integer.parseInt(sval[1]);
            res.append(val.toString());
            res.append(";");
        }
        String[] tmp = res.toString().split(";");
        StringBuilder rres = new StringBuilder();
        for(String s:tmp){
            String[] t = s.split(",");
            rres.append(t[0]+","+(double)Integer.parseInt(t[1])/sum );
            rres.append(";");
        }
        context.write(new Text(key),new Text(rres.toString()));
    }
```

2.3.6 部分结果展示



2.4 任务四：PageRank计算

2.4.1 任务要求

由任务3得到的图计算PageRank。

任务输入：任务3的输出

任务输出：各个人物PageRank排序后的结果

2.4.2 任务分析

关于PageRank算法的并行性在课程中有详细的介绍，而考虑这次实验具有特殊性，归一化后得到的共现次数实际上就是贡献概率，所以我们修正原来的迭代公式就可以达成目标。具体的修正如下：

$$PR(p_i) = \frac{1-d}{N} + d\sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

将原来的$\frac{1}{L(p_j)}$ 根据本题的意义，实际上这一项应该修改为边的权重：

$$PR(p_i) = \frac{1-d}{N} + d\sum_{p_j \in M(p_i)} PR(p_j) \cdot Edge_{ij}$$

2.4.3 任务过程

任务需要有三个阶段组成

阶段一的任务是初始化建图，也即初始化所有的人物PageRank为1.0

阶段二是迭代，通过map过程生成两种类型的<key,value>，一种是计算得出的某个点的PR值，另一种则是为了传递图的结构。整个迭代过程大约需要10次左右。

阶段三是PR值的排序，实际上我们利用框架自带的排序可以轻松的完成（大概是一行代码，详见下面的代码展示）。

2.4.4 任务的难点及攻克

无

2.4.5 代码展示

```
public class PR {
    public static class BuildGraph extends Mapper<Text,Text,Text, Text>{
        public void map(Text key,Text val,Context context) throws IOException,
InterruptedException{
            context.write(key,new Text("1.0;"+val.toString()));
        }
    }
    public static class PRMapper extends Mapper<Text,Text,Text, Text>{
        public void map(Text key,Text val,Context context) throws IOException,
InterruptedException{
            String sval = val.toString();
            String[] links = sval.split(";");
            double rank = Double.parseDouble(links[0]);
            for(int i = 1;i<links.length;++i){
                String[] tuple = links[i].split(",");
                context.write(new Text(tuple[0]), new
Text(rank*Double.parseDouble(tuple[1])+""));
            }
            context.write(key,new Text(sval.substring(sval.indexOf(";")+1)));
        }
    }

    public static class PRReducer extends Reducer<Text,Text,Text,Text>{
        public void reduce(Text key,Iterable<Text> vals,Context context) throws
IOException, InterruptedException{
            double PR = 0.0;
            String res = "";
            for(Text val:vals){
                String tmp = val.toString();
                if(tmp.indexOf(";")!=-1){
                    res = tmp;
                }
                else {
                    PR += Double.parseDouble(tmp);
                }
            }
            PR = 0.15+0.85*PR;
            context.write(new Text(key),new Text(PR+";"+res));
        }
    }
```

```
    public static class PRviewer extends Mapper<Text,Text, FloatWritable,Text>{
        public void map(Text key,Text val,Context context) throws
IOException,InterruptedException{
            String[] tmp = val.toString().split(";");
            context.write(new FloatWritable(Float.parseFloat(tmp[0])),new
Text(key));
        }
    }
}
```

2.4.6 部分结果展示

4.8540974	布莱克
6.2267485	韦斯
6.388073	乌姆里奇
7.8384147	莱特林
7.881144	卢平
8.647676	纳威
10.540104	格兰芬
10.553443	金妮
10.718002	乔治
11.336994	弗雷德
11.716468	马尔福
11.99584	小天狼星
13.42235	霍格
14.992972	斯内普
16.482944	海格
29.532143	邓布利多
52.43427	赫敏
57.965492	罗恩
123.0439	哈利

2.5 任务五：在人物关系图上的标签传播

2.5.1 任务要求

对人物关系图进行标签传播。

输入：任务3的输出

输出

```
样例输入：
丁沃斯 鲍曼·赖特,0.125;芙蓉,0.125;罗恩,0.125;比尔,0.125;戈德里克·格兰芬多,0.125;戈德里
克,0.125;奥特里－圣卡奇波尔,0.125;上弗莱格利,0.125;

样例输出：
Label_0:
蒙顿格斯， 达芙尼·格林格拉斯， ...
```

2.5.2 任务分析

标签传播算法（Label Propagation Algorithm，LPA）是一种基于图的半监督学习方法，基本思路是用已标记节点的标签去预测未标记节点的标签。具体做法是，对每一个节点，考察其所有邻居节点的标签，以其中出现次数最大者作为当前节点的新的标签。

由于任务中并没有指定初始标签，这里将所有节点的初始标签设置为节点自身（如：100号节点的标签为"100"）。

2.5.3 任务过程

为了便于MapReduce程序计算，在运行Job之前对输入数据进行预处理：

1. 建立person_name_list.txt中的人名和序号的映射，在Mapper和Reducer中全程使用序号计算以避免重复处理字符串。
2. 同样地，为了避免处理字符串，在运行Job之前将输入数据转换为SequenceFile格式，包括人物关系图和初始的标签信息。人物关系图的格式为`<IntWritable, DoubleArray>`，标签信息的格式为`<IntWritable, IntWritable>`。`DoubleArray`是自定义的`Writable`类型，可以看作是`double`类型的数组。逻辑上，人物关系图是一个邻接矩阵。标签信息表示人名序号和标签的对应关系。

处理完成后，开始迭代运行任务。每次任务完成后，将本次的输出结果转换为文本格式写入HDFS，然后用本次的输出结果覆盖上次的输出结果。由于振荡的缘故，无法通过连续两次输出是否相等来判断终止迭代，因此需要指定最大迭代次数。

2.5.4 任务的难点及攻克

在测试数据上，算法不收敛，而是在迭代6次后发生振荡，结果在两个状态之间变化。且这两个结果都出现了大部分节点有着相同标签的现象（与"哈利"相同的标签）。

经过讨论，我们发现如果每次迭代只用了最大的边权进行标签传播，这实际上是一种信息的浪费。事实上不能仅仅依靠最大的边权进行传播，我们需要在传播时保留其余边权信息，具体而言在一次迭代过程中，每个节点需要更新的参数将不再是标签，而是**标签的概率分布**。具体而言，节点将所有邻居节点的标签分布**加权后**作为自己新的标签分布；算法结束时，取标签概率分布中概率最大的标签作为节点最终的标签。具体的算法如下：

```
function weightedLPA(G,maxtimes):
    for v in G:
        v.DecisionMatrix = [0]*lengthofLabel

    repeat
        for v in G:
            for all neighbor w of v:
                v.DecisionMatrix += Edgewv*w.DecisionMatrix  #用边权乘以邻居节点的概率分布
            v.Normalize()
    until maxtimes or convergence

    for v in G:
        v.Label = maxindex(v.DecisionMatrix)

    return v.Label
```

2.5.5 代码展示

map

```
public void map(IntWritable key, DoubleArray value, Context context)
        throws IOException, InterruptedException {
    for(int i = 0; i < value.get().length; i++) {
        double weight = value.get()[i];
        int label = this.labels[i];
        context.write(key, new Weight(label, weight));
    }
}
```

reduce

```
public void reduce(IntWritable key, Iterable<Weight> value, Context context)
        throws IOException, InterruptedException {
    int label = -1;
    double max_weight = -1;
    HashMap<Integer, Double> weighted_labels = new HashMap<Integer, Double>();
    for(Weight w: value) {
        if(!weighted_labels.containsKey(w.get_label())) {
            weighted_labels.put(w.get_label(), w.get_weight());
        }
        else {
            weighted_labels.replace(w.get_label(),
weighted_labels.get(w.get_label()) + w.get_weight());
        }
        if(weighted_labels.get(w.get_label()) > max_weight) {
            label = w.get_label();
            max_weight = w.get_weight();
        }
    }
    context.write(key, new IntWritable(label));
}
```

采用标签概率分布模型进行迭代

```
public static class LPARYMapper extends Mapper<Text, Text, Text, Text> {
    private String[] namelist = new String[885];
    private DoubleArray[] matrix = new DoubleArray[885];

    public void setup(Context context) throws IOException, InterruptedException{
        //get namelist; 已经省略

        //init matrix
        for(int i=0;i<885;++i){
            double[] tmpdata = new double[885];
            matrix[i] = new DoubleArray(tmpdata);
        }
        //get matrix
        try {
            Configuration conf = context.getConfiguration();
            Path pt = new Path("/home/rycbe/final/LPAGraph_"+
                               conf.getInt("LPAInput",-1)   //这个数据代表了实际的轮
次
                               +"/part-r-00000");
            FileSystem fs = FileSystem.get(conf);
```

```
            BufferedReader br = new BufferedReader(new
InputStreamReader(fs.open(pt)));
            String line;
            line = br.readLine();
            while (line != null) {
                String[] kv = line.split("\t");
                int kindex = Integer.parseInt(kv[0]);
```

```
                String[] links = kv[1].split(" ");
                for(int i=0;i<links.length;++i){
                    double s = Double.parseDouble(links[i]);

                    matrix[kindex].set(i,s);
                }
                line = br.readLine();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    public void map(Text key, Text val, Context context)
        throws IOException, InterruptedException{

        int k = Integer.parseInt(key.toString());
        DoubleArray res = new DoubleArray(new double[885]);

        double[] mk = matrix[k].get();
        for(int i=0;i<mk.length;++i){
            if(i==k){
                continue;
            }
            else{
                DoubleArray tmp = matrix[i].mutiply(mk[i]);  //按照概率分布模型进行
计算
                res._add(tmp.get());
            }
        }
        res.normalize();

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

2.5.6 结果展示

```
Result
Label_0[689]
蒙顿格斯，达芙尼·格林格拉斯，赫普兹巴·史密斯，秋·张，斯内平特，布拉德利，
Label_1[1]
阿尔弗雷德·卡特莫尔
Label_2[1]
巴恩斯利
Label_3[1]
尤里克
Label_4[1]
尼古拉斯·德·敏西
Label_5[1]
萨斯·马尔福
Label_6[1]
奥古斯都·派伊
Label_7[1]
墨瑞克
Label_8[2]
戴·卢埃林，希伯克拉特·斯梅缀克
Label_9[1]
阿布拉克
Label_10[1]
莫尼卡·威尔金斯
Label_11[1]
温德尔
Label_12[179]
吉姆，尼古拉斯，布雷司·沙比尼，埃玛·多布斯，格丝尔达·玛奇班，哈罗德·丁戈，
Label_13[1]
波平顿
Label_14[1]
梅齐
Label_15[1]
威廉姆·亚瑟
Label_16[1]
玛丽多尔金
Label_17[1]
芙蓉·伊萨贝尔
```

> 输出结果展示：例如，在输入的关系图中，“阿尔弗雷德·卡特莫尔”和“梅齐”的彼此关系权重都为1.0。因此在计算标签传播时，这两个节点之间的标签发生振荡，其他的单节点的标签也都是因为类似的原因出现振荡而未合并。

2.5.7 jar包的使用方式

Usage: LPA <data> <namelist> <out>

附：参考资料

LPA概率模型算法介绍：

https://blog.csdn.net/weixin_46348799/article/details/108296364

LPA算法基础介绍：

https://blog.csdn.net/bbbeoy/article/details/82666644

关于 hadoop reduce 阶段遍历 Iterable 的 2 个“坑”：

https://blog.csdn.net/xukaics/article/details/48434787