# 实验 4：MapReduce 高级编程技术

陈德丹 221220159

## 任务 1：社交网络中的互相关注好友

○ **基本思路**: map 阶段对于输入的每个用户的关注列表，各生成一个用户对，小号在前，大号在后，例如 3-5，这样可以避免重复，同时打上标记，小号关注大号则标记 1，大号关注小号则标记 2，reduce 阶段对于每个用户对，如果两种标记都存在，则说明互相关注并输出

○ **mapper** 输出的 key 的类型为 Text，value 为 IntWritable

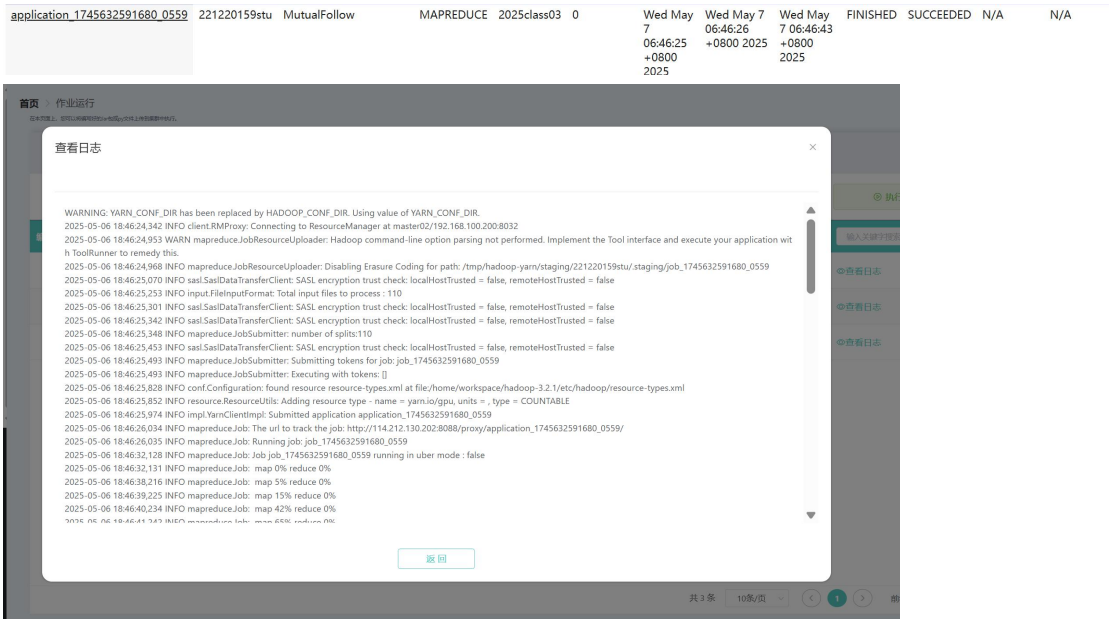
```
private Text userPair = new Text();
private final static IntWritable DIRECTION = new IntWritable();
public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
    // 解析输入行，格式如 "user:follow1 follow2 ..."
    String[] parts = value.toString().split(":");
    if (parts.length != 2) return;
    String userA = parts[0].trim();
    int a = Integer.parseInt(userA);  //将输入的用户A转换为整数

    String[] following = parts[1].trim().split(regex:"\\s+");
    for (String userB : following) {  // 遍历用户A的关注列表
        if (userB.isEmpty()) continue;
        int b = Integer.parseInt(userB);

        // 确保小号在前，大号在后
        int min = Math.min(a, b);
        int max = Math.max(a, b);
        String pair = min + "-" + max;
        userPair.set(pair);

        // 判断关注方向（标记1或2）
        if (a == min) {
            DIRECTION.set(1); // 小号关注大号
        } else {
            DIRECTION.set(2); // 大号关注小号
        }
        context.write(userPair, DIRECTION);
```

○ **reducer** 输出的 key 的类型为 Text，value 为 NullWritable

```java
// Reducer：检查是否存在双向标记
public static class MutualFollowReducer
        extends Reducer<Text, IntWritable, Text, NullWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {
        boolean hasForward = false;
        boolean hasReverse = false;

        for (IntWritable val : values) {
            if (val.get() == 1) hasForward = true;
            else if (val.get() == 2) hasReverse = true;

            // 提前终止循环优化性能
            if (hasForward && hasReverse) break;
        }

        // 输出互相关注的用户对
        if (hasForward && hasReverse) {
            context.write(key, NullWritable.get());
        }
    }
}
```

○ 平台输出结果（路径为/user/221220159stu/output-Exp4/1）


○ 平台运行过程



## 任务 2：社交网络寻找共同关注

○ **基本思路**： 根据任务 1 的输出结果，首先建立 每个用户 与 参与的所有互相关注对的映射，map 阶段对于输入的用户，把它参与的互相关注对分别作为键，它的关注列表作为值输出，这样在 reduce 阶段，对于一个互相关注用户对（如 1-2），能接收到 1 的关注列表和 2 的关注列表，从而计算它们的共同关注并输出

○ **输入**依次为：①.原始关注数据（user:follow1 follow2...）②任务 1 生成的互相关注用户对（如 1-2）③x

**输出**：

共同关注数 ≤x 的结果到 lessX 文件（如 less15）

共同关注数 >x 的结果到 greaterX 文件（如 greater15）

○ **Mapper** 需要加载互相关注用户对数据，并为每个用户对关联其关注列表

首先在 setup 阶段，将每个用户相关联的所有互相关注用户对存到一个 Map 中，例如用户 1 对应 1-2，1-3

```
// 存储格式：<用户ID, 该用户参与的所有互相关注用户对>
// 例如用户1参与的用户对：["1-2", "1-3"]
private Map<String, List<String>> mutualPairs = new HashMap<>();
private BufferedReader fis;

@Override
protected void setup(Context context) throws IOException {
    // 从分布式缓存读取任务1生成的互相关注用户对
    Configuration conf = context.getConfiguration();
    URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();
    for (URI patternsURI : patternsURIs) {
        Path patternsPath = new Path(patternsURI.getPath());
        String fileName = patternsPath.getName().toString();
        fis = new BufferedReader(new FileReader(fileName));
        String line = null;
        while ((line = fis.readLine()) != null) {
            String pair = line.trim(); // 用户对格式如"1-2"
            String[] users = pair.split(regex:"-");
            if (users.length != 2) continue; // 跳过格式错误的数据

            // 将用户对关联到两个用户上，便于后续快速查找
            // 示例：用户对"1-2"会添加到用户1和用户2的关联列表中
            mutualPairs.computeIfAbsent(users[0], k -> new ArrayList<>()).add(pair);
            mutualPairs.computeIfAbsent(users[1], k -> new ArrayList<>()).add(pair);
        }
    }
}
```

map 阶段，读入每个用户的关注列表，遍历它参与的所有互相关注用户对，将关注列表作为值传给 reducer（map 输出的键和值类型均为 Text）

```
public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
    // 输入格式示例："1:2 3 4"
    String[] parts = value.toString().split(":");
    if (parts.length != 2) return; // 跳过格式错误的数据

    String currentUser = parts[0].trim();    // 当前用户ID
    String followsList = parts[1].trim();    // 该用户的关注列表

    // 查找当前用户参与的所有互相关注用户对
    List<String> pairs = mutualPairs.get(currentUser);
    if (pairs != null) {
        for (String pair : pairs) {
            /* 输出格式:
             * Key   : 用户对（如"1-2"）
             * Value : "当前用户:关注列表"（如"1:2 3 4"）
             *
             * 目的：将同一用户对的两个用户的关注列表发送到同一个Reducer
             */
            context.write(new Text(pair), value);
        }
    }
}
```

○ **Reducer** 的 setup 阶段，首先根据传入的 x，动态生成输出文件名

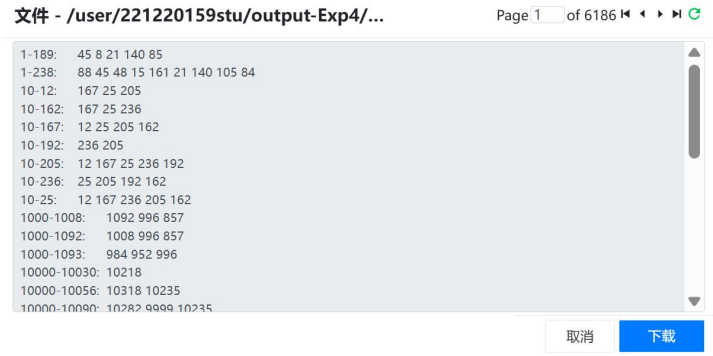
```
public static class CommonFollowersReducer
        extends Reducer<Text, Text, Text, Text> {

    private MultipleOutputs<Text, Text> mos; // 多路输出控制器
    private int x;                           // 阈值参数
    private String lessOutputName, greaterOutputName; // 动态输出文件名

    @Override
    protected void setup(Context context) {
        mos = new MultipleOutputs<>(context);
        // 从作业配置中读取参数x（默认值10）
        x = context.getConfiguration().getInt("x", 10);
        // 动态生成输出文件名（例如x=15 → "less15"和"greater15"）
        lessOutputName = "less" + x;
        greaterOutputName = "greater" + x;
    }
```

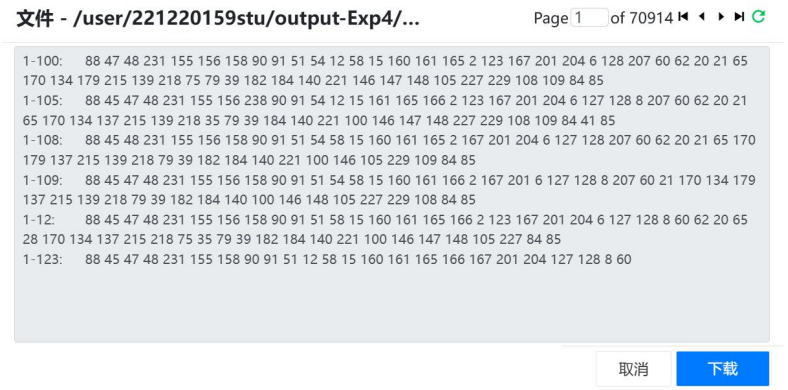
reduce 函数中，分别存储用户对中两个用户的关注列表，通过集合的 retainAll 方法筛选出共同关注，并根据共同关注数量选择输出路径（输出的键和值类型均为 Text）

```
public void reduce(Text key, Iterable<Text> values, Context context)
        throws IOException, InterruptedException {
    // 用户对格式如"1-2"
    String[] users = key.toString().split("-");
    String userA = users[0]; // 用户A（小号）
    String userB = users[1]; // 用户B（大号）

    // 存储两个用户的关注列表
    Set<String> followsA = new HashSet<>();
    Set<String> followsB = new HashSet<>();

    // 解析来自Mapper的值（格式："用户ID:关注列表"）
    for (Text val : values) {
        String[] parts = val.toString().split(":");
        if (parts.length != 2) continue; // 跳过格式错误的数据

        String userId = parts[0];       // 用户ID
        String[] follows = parts[1].trim().split(regex:"\\s+"); // 分割关注列表

        // 将关注列表按用户归属分类
        if (userId.equals(userA)) {
            followsA.addAll(Arrays.asList(follows));
        } else if (userId.equals(userB)) {
            followsB.addAll(Arrays.asList(follows));
        }
    }
```

```
    // 计算交集：共同关注
    Set<String> common = new HashSet<>(followsA);
    common.retainAll(followsB);
    if (common.isEmpty()) return; // 无共同关注则跳过

    System.out.println(common.size());
    System.out.println(common);

    // 将共同关注的交集转化为String
    String outputValue = String.join(delimiter:" ", common);

    // 构造输出键（原用户对追加冒号，如"1-2:"）
    Text outputKey = new Text(key.toString() + ":");

    // 根据共同关注数量选择输出路径
    if (common.size() <= x) {
        mos.write(lessOutputName, outputKey, new Text(outputValue));
    } else {
        mos.write(greaterOutputName, outputKey, new Text(outputValue));
    }
}
```

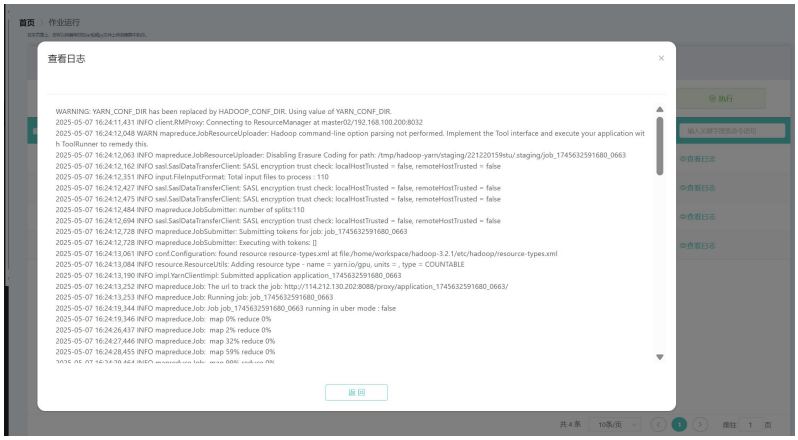
○ 平台输出结果（路径为/user/221220159stu/output-Exp4/2）（x=15）




○ 平台运行过程

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| application_1745632591680_0663 | 221220159stu | CommonFollowers | MAPREDUCE | 2025class03 | 0 | Thu May 8 04:24:13 +0800 2025 | Thu May 8 04:24:13 +0800 2025 | Thu May 8 04:24:59 +0800 2025 | FINISHED | SUCCEEDED | N/A | N/A | N |



### 任务 3：好友推荐

○ **基本思路**：观察示例可知，如果用户 B 和用户 A 互关，用户 A 和用户 C 互关，但 B 没有关注 C，则向 B 推荐 C。因此可以根据任务 1 的输出结果建立每个用户互相关注的用户集合，对于每个用户，遍历它互相关注的互相关注，如果它没有关注则推荐，再将计数前 5 的结果输出即可。

○ **Mapper** 的 setup 阶段，从互相关注用户对中建立每个用户互相关注的用户集合

```
// 建立互相关注列表 存储格式：<用户ID, 与该用户互相关注的所有用户>
// 例如用户1对应 ["2", "3"]
private Map<String, Set<String>> mutualFollowers = new HashMap<>();
private BufferedReader fis;

@Override
protected void setup(Context context) throws IOException {
    // 从分布式缓存读取任务1生成的互相关注用户对
    Configuration conf = context.getConfiguration();
    URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();
    for (URI patternsURI : patternsURIs) {
        Path patternsPath = new Path(patternsURI.getPath());
        String fileName = patternsPath.getName().toString();
        fis = new BufferedReader(new FileReader(fileName));
        String line = null;
        while ((line = fis.readLine()) != null) {
            String pair = line.trim(); // 用户对格式如"1-2"
            String[] users = pair.split(regex:"-");
            if (users.length != 2) continue; // 跳过格式错误的数据

            // 将用户1添加到用户0的互相关注列表中，将用户0添加到用户1的互相关注列表中
            mutualFollowers.computeIfAbsent(users[0], k -> new HashSet<>()).add(users[1]);
            mutualFollowers.computeIfAbsent(users[1], k -> new HashSet<>()).add(users[0]);
        }
    }
}
```

map 函数，首先存储输入用户的关注列表，再遍历它互相关注的互相关注，如果没有关注则输出（map 的 key 和 value 的类型均为 Text）

```
public void map(Object key, Text value, Context context)
    throws IOException, InterruptedException {
    // 输入格式: user:follow1 follow2...
    String[] parts = value.toString().split(":");
    String userB = parts[0].trim();

    //将用户B的关注列表转换为集合
    Set<String> userB_following = new HashSet<>(
        Arrays.asList(parts[1].trim().split(regex:" "))
    );
    // 获取用户B的互相关注列表
    Set<String> userB_mutual_follow = mutualFollowers.get(userB);
    if (userB_mutual_follow == null) return; // 如果没有互相关注的用户，跳过
    // 遍历用户B的所有互相关注用户
    for (String userA : userB_mutual_follow) {
        // 对于与用户B互相关注的用户A，获取其互相关注列表
        Set<String> userA_mutual_follow = mutualFollowers.get(userA);
        for (String userC : userA_mutual_follow) {
            if (userB_following.contains(userC)) continue;
            if (userB.equals(userC)) continue; // 跳过自己

            // 此时A和B、A和C都互相关注，但B不关注C
            // 生成推荐关系：给B推荐C
            // Key: 被推荐人(userB), Value: 候选用户
            context.write(new Text(userB), new Text(userC));
        }
    }
```

○ **Reducer** 对于每个用户的候选被推荐用户，用一个 Map 记录推荐次数，排序后输出推荐次数前 5 的用户（键和值类型均为 Text）

```
public static class RecommendationReducer
        extends Reducer<Text, Text, Text, Text> {

    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
        // 存储候选用户的推荐次数
        Map<String, Integer> recommendations = new HashMap<>();

        // 统计推荐次数
        for (Text val : values) {
            String candidate = val.toString();
            recommendations.put(candidate, recommendations.getOrDefault(candidate, defaultValue:0) + 1);
        }

        // 构造前5推荐结果
        // 排序并取前 5 个 key，拼接成字符串
        String top5 = recommendations.entrySet().stream()
                .sorted((e1, e2) -> e2.getValue().compareTo(e1.getValue())) // 按 value 倒序
                .limit(maxSize:5)
                .map(Map.Entry::getKey) // 只保留 key
                .collect(Collectors.joining(delimiter:" ")); // 用空格拼接

        context.write(key, new Text(top5));
    }
}
```

○ 平台输出结果（路径为/user/221220159stu/output-Exp4/3）


○ 平台运行过程

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| application_1745632591680_0647 | 221220159stu | FriendRecommendation | MAPREDUCE | 2025class03 | 0 | Thu May 8 03:10:11 +0800 2025 | Thu May 8 03:10:11 +0800 2025 | Thu May 8 03:10:52 +0800 2025 | FINISHED | SUCCEEDED | N/A | N/A | N/A |