第5章MapReduce进阶编程

(源自:https://biglab.site)

(版本:Ver1.2-20230828)

任务前言

学习目标

1. 掌握Hadoop Java API的使用。
2. 理解Combiner的工作原理。
3. 掌握使用Combiner对MapReduce工作流程进行优化。
4. 了解Hadoop内置数据类型。
5. 掌握编写和使用自定义数据类型。
6. 掌握编写和使用Partitioner设置分区。
7. 掌握MapReduce参数传递方式。
8. 学会使用ToolRunner提交MapReduce任务。
9. 掌握使用Eclipse提交MapReduce任务。

任务背景

2021年3月，网站运营方提出了新的需求，为了比较今年与去年同期的用户访问数据，要求分别统计出2021年1月与2月的用户访问次数，并输出到不同的目录中。

本章结合竞赛网站每日访问次数的统计任务。

1. 首先介绍MapReduce中输入输出文件格式的设置过程。
2. 接着介绍Hadoop Java API的使用，方便HDFS系统的文件及目录操作。
3. 重点讲解了自定义键值类型、Combiner、Partitioner分区器、自定义计数器的使用，对竞赛网站每日访问次数的MapReduce程序进行优化，提高程序运行的效率。
4. 最后介绍在IDEA中直接提交并运行MapReduce任务的操作过程，简便程序运行的操作。

任务5.1筛选日志文件并生成序列化文件

任务描述

在大数据文件的分析处理中，尤其是在处理逻辑比较复杂的情况下，需要使用多个MapReduce程序连续进行处理，因此需要在HDFS上保存大量的中间结果。如何提高中间结果的存取效率，对于整个数据处理流程具有重要意义。

Hadoop序列化具有紧凑、快速、可扩展以及互操作的特点，十分适合作为MapReduce任务的输入与输出数据格式。

本小节的任务是从原始日志数据中筛选出1月与2月的数据，以序列化文件的格式保存，为后续的数据处理任务做准备

一般而言，HDFS的一个文件对应多个数据块，每个数据块对应一个输入分片InputSplit，而每个Map任务只处理一个输入分片，每个分片包含一批记录，每条记录是一个键值对。

编写Mapper模块时只关注输入的键值对，而没有关注输入数据块。输入数据块主要是由Hadoop自带的输入格式进行处理的。

获取测试数据

说明:使用crt进入集群的master或slave节点,继续上一章，使用raceData.csv作为数据分析对象,如hdfs已存在则可以不用重传

```
cd /root/hadoop
tar -zxvf ./hadoop_data.tar.gz
hdfs dfs -put ./hadoop_data/raceData.csv  /user/myname/
```

MapReduce输入格式

常用输入格式：

输入格式	描述	键类型	值类型
TextInputFormat	默认格式，读取文件的行	行的字节偏移量（LongWriable）	行的内容（Text）
SequenceFileInputFormat	Hadoop定义的高性能二进制格式	用户自定义	
KeyValueInputFormat	将行解析为键值对	第一个tab字符前的所有字符（Text）	行剩下的内容（Text）

MapReduce输出格式

常用的输出格式：

输出格式	描述
TextOutputFormat	默认的输出格式，以“key \t value”的方式输出行
SequenceFileOutputFormat	输出二进制文件，适合作为子MapReduce作业的输入
NullOutputFormat	忽略收到的数据，即没有输出

筛选日志代码实现

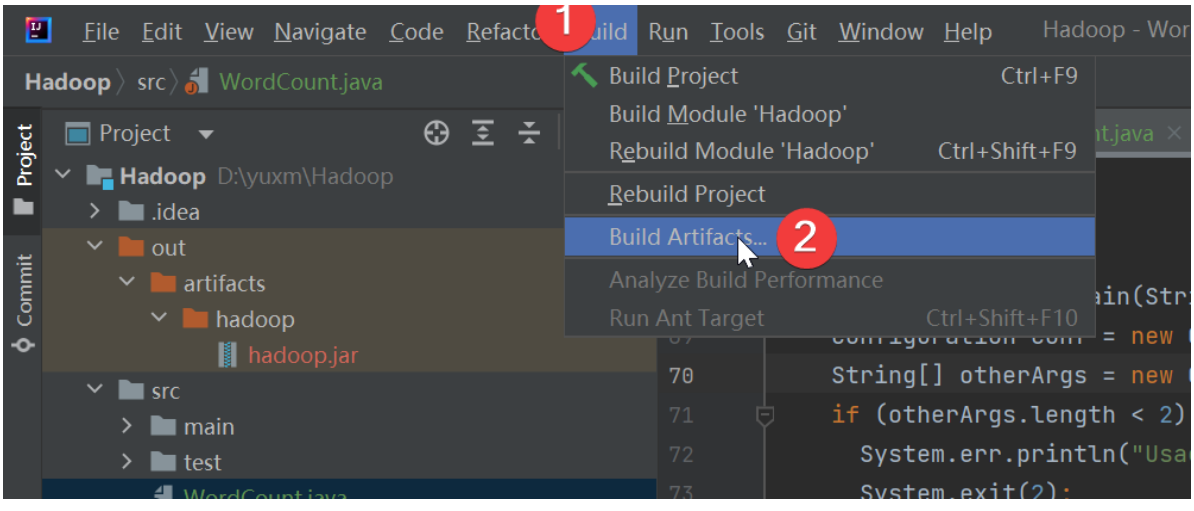
源码参考：

```
package chap5_select;

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.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

import java.io.IOException;

public class SelectData {
    public static class MyMap extends Mapper<Object, Text, Text, Text> {
        public void map(Object key, Text value, Context context)
                throws IOException, InterruptedException {
            String line = value.toString();
            String arr[] = line.split(",");
```

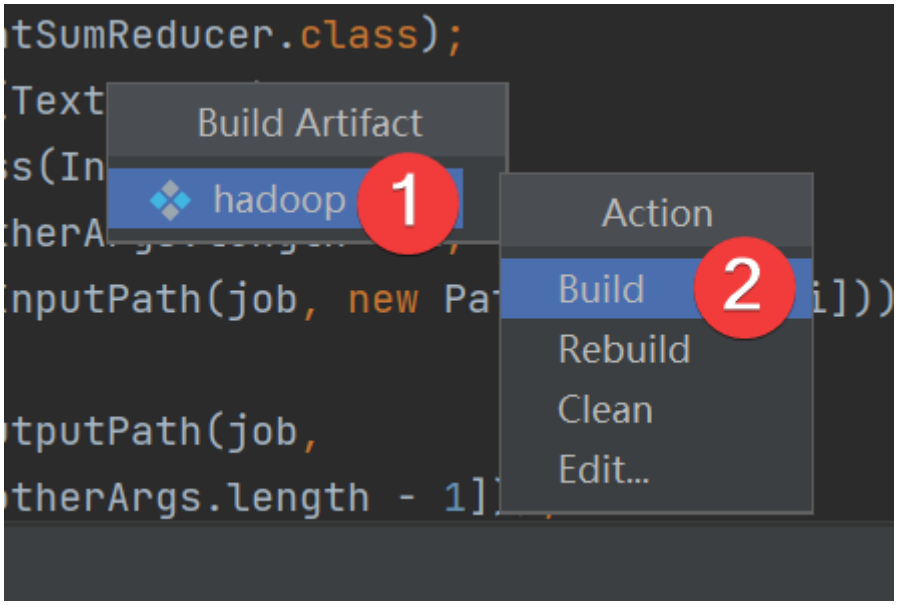
```
            if (arr[4].contains("2021/1") || arr[4].contains("2021/2")) {
                context.write(new Text(arr[2]),
                        new Text(arr[4].substring(0, 9)));
            }
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args)
                .getRemainingArgs();
        if (otherArgs.length < 2) {
            System.err.println("必须输入读取文件路径和输出路径");
            System.exit(2);
        }
        Job job = Job.getInstance(conf, "Select Data");
        job.setJarByClass(SelectData.class);
        job.setMapperClass(MyMap.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        // 设置输入格式
        job.setInputFormatClass(TextInputFormat.class);
        // 设置输出格式
        job.setOutputFormatClass(SequenceFileOutputFormat.class);
        // 设置Reducer任务数为0
        job.setNumReduceTasks(0);
        for (int i = 0; i < otherArgs.length - 1; ++i) {
            FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
        }
        FileOutputFormat.setOutputPath(job,
                new Path(otherArgs[otherArgs.length - 1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }

}
```

编译项目

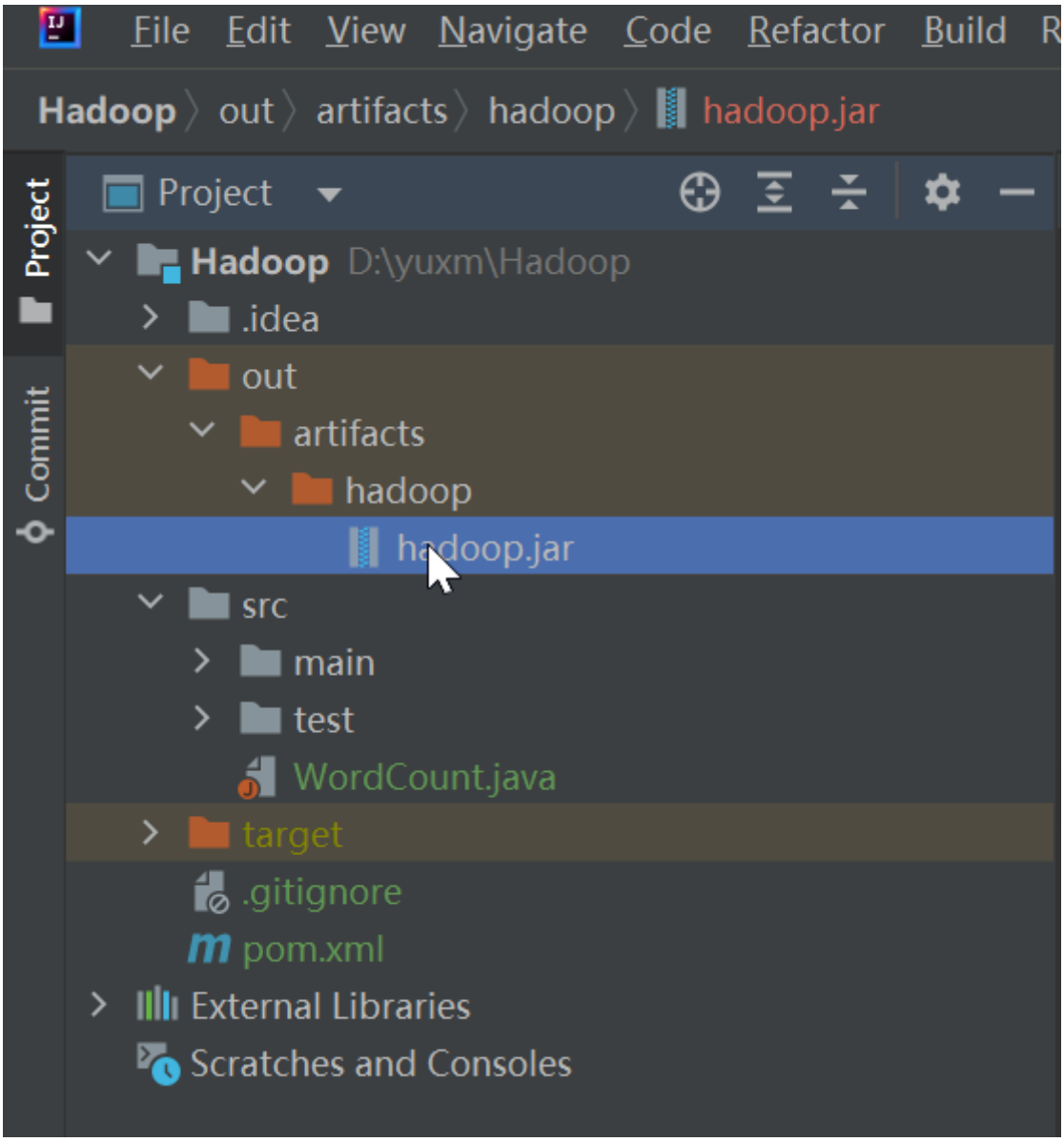
点菜单build->BuildArtifacts...



弹出菜单：



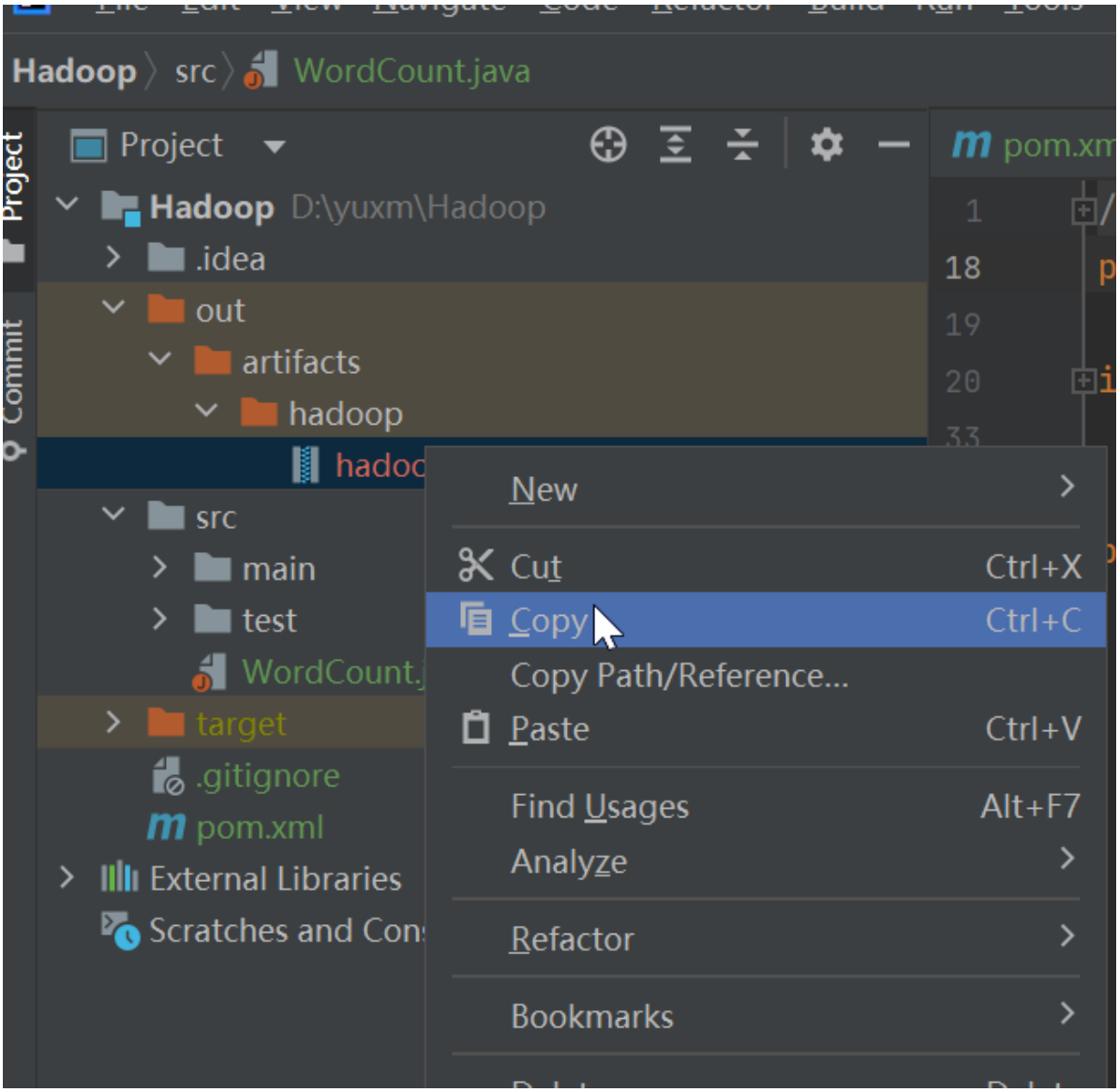
编译完成后，会在左侧源码树图上：out->artifacts-> hadoop下出现hadoop.jar文件，这就是编译后的应用程序：



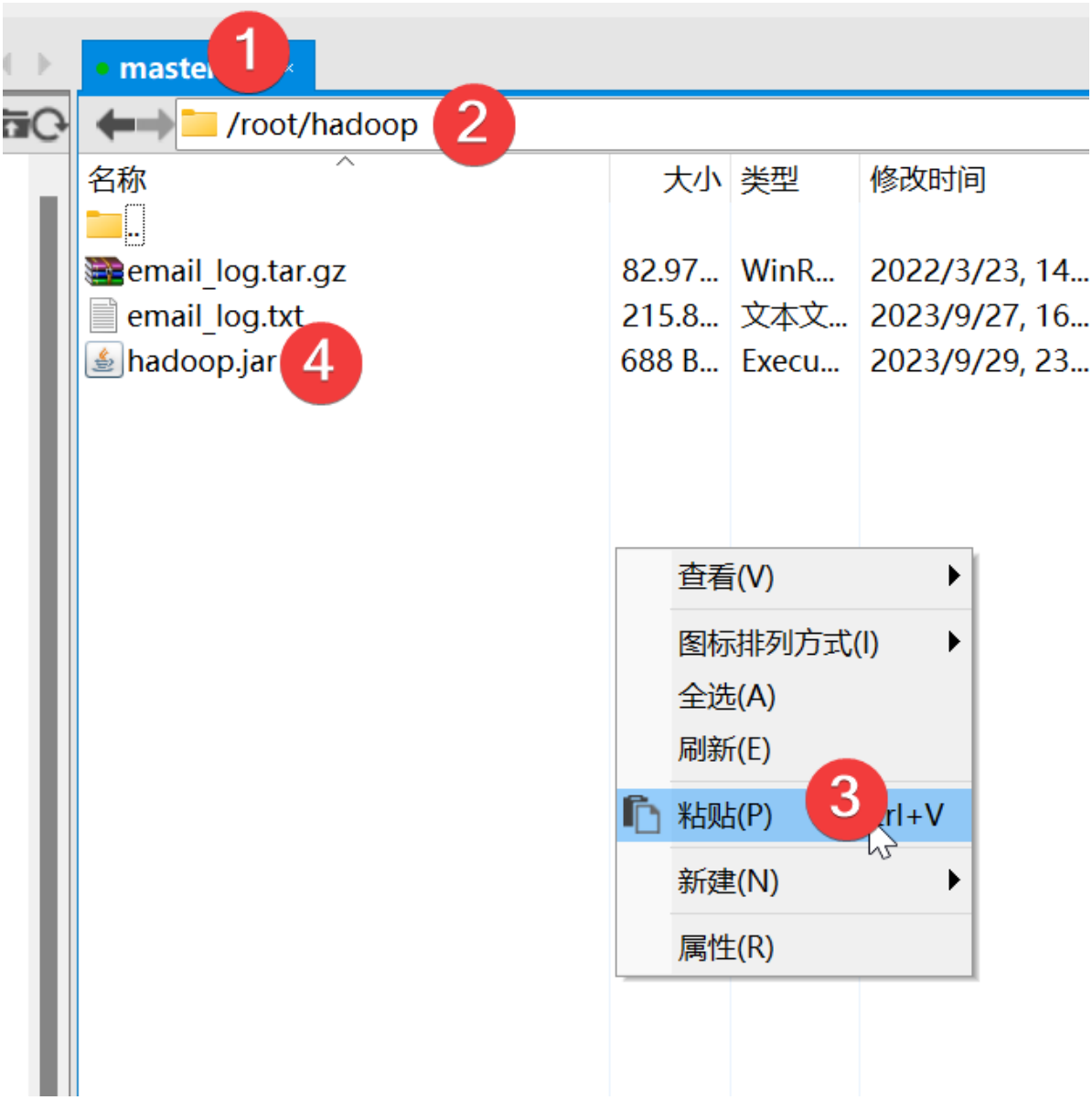
上传编译后文件

上传编译后的文件到集群

右击上图的hadoop.jar,选择菜单“复制”



然后打开xftp工具，连接到master虚拟主机上，进入 /root/hadoop目录，右击空白的地方，在弹出的菜单中选择“粘贴”



运行编译后文件

使用xshell或crt打开master,进入/root/hadoop目录，执行编译后的文件hadoop.jar：

```
cd /root/hadoop
hadoop jar ./hadoop.jar chap5_select.SelectData /user/myname/raceData.csv
/user/myname/output_raceData
```

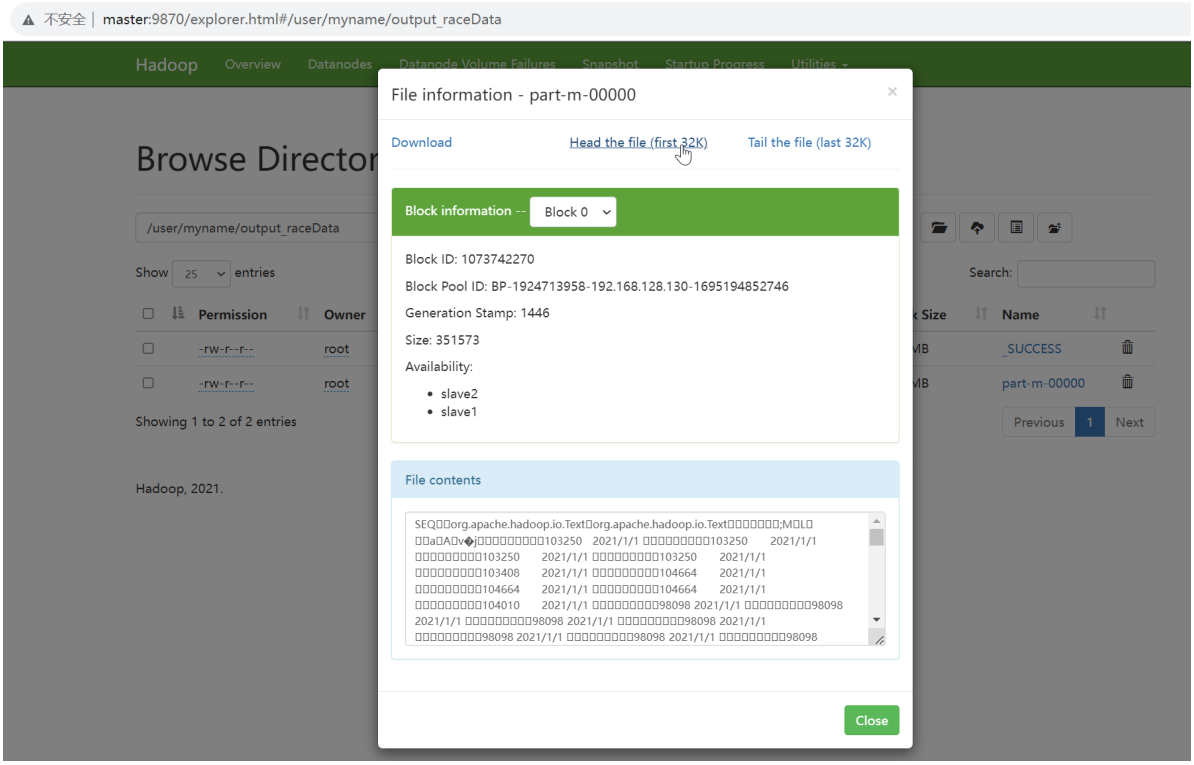
运行结果如：

```
[root@master hadoop_data]# cd /root/hadoop
[root@master hadoop]# hadoop jar ./hadoop.jar main.java.SelectData
/user/myname/raceData.csv  /user/myname/output_raceData
2023-10-11 09:43:50,902 INFO client.RMProxy: Connecting to ResourceManager at
master/192.168.128.130:8032
2023-10-11 09:43:52,092 INFO mapreduce.JobResourceUploader: Disabling Erasure
Coding for path: /tmp/hadoop-
yarn/staging/root/.staging/job_1696986888826_0001
2023-10-11 09:43:53,048 INFO input.FileInputFormat: Total input files to
process : 1
```

```
2023-10-11 09:43:53,233 INFO mapreduce.JobSubmitter: number of splits:1
2023-10-11 09:43:53,569 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1696986888826_0001
2023-10-11 09:43:53,572 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-10-11 09:43:53,884 INFO conf.Configuration: resource-types.xml not found
2023-10-11 09:43:53,885 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-10-11 09:43:54,586 INFO impl.YarnClientImpl: Submitted application application_1696986888826_0001
2023-10-11 09:43:54,658 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/application_1696986888826_0001/
2023-10-11 09:43:54,659 INFO mapreduce.Job: Running job: job_1696986888826_0001
2023-10-11 09:44:07,222 INFO mapreduce.Job: Job job_1696986888826_0001 running in uber mode : false
2023-10-11 09:44:07,224 INFO mapreduce.Job:  map 0% reduce 0%
2023-10-11 09:44:17,403 INFO mapreduce.Job:  map 100% reduce 0%
2023-10-11 09:44:17,419 INFO mapreduce.Job: Job job_1696986888826_0001 completed successfully
2023-10-11 09:44:17,535 INFO mapreduce.Job: Counters: 32
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=221811
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=51641009
                HDFS: Number of bytes written=351573
                HDFS: Number of read operations=7
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=29212
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=7303
                Total vcore-milliseconds taken by all map tasks=7303
                Total megabyte-milliseconds taken by all map tasks=14956544
        Map-Reduce Framework
                Map input records=389603
                Map output records=14336
                Input split bytes=108
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=271
                CPU time spent (ms)=3090
                Physical memory (bytes) snapshot=105512960
                Virtual memory (bytes) snapshot=3596816384
                Total committed heap usage (bytes)=18059264
                Peak Map Physical memory (bytes)=105512960
                Peak Map Virtual memory (bytes)=3596816384
```

```
        File Input Format Counters
                Bytes Read=51640901
        File Output Format Counters
                Bytes Written=351573
[root@master hadoop]#
```

查看web上输出结果：/user/myname/output_raceData/part-m-00000



任务5.2Hadoop Java API读取序列化日志文件

任务描述

针对HDFS上的文件操作，大多数是通过在终端执行HDFS命令进行的。从本地计算机上传文件至HDFS时，还需要经过集群节点这个中转站，操作过程不但烦琐，而且也占用了不必要的存储空间。

通过Hadoop Java API对HDFS中的文件进行操作，不但能够轻松进行各类常规的文件操作，而且提供了多种文件类型接口，能够轻松处理文本、键值对、序列化等多种文件格式。本章的第2个任务将简要讲解Java API的基本操作与应用。

本小节的任务是使用Hadoop Java API的方式读取上一小节的用户日志序列化文件，并将读取的数据保存到本地文件系统中。

获取测试数据

在本地d:盘创建tmp目录，作为上下传文件的临时目录：D:\tmp

本小节使用上节应用生成的结果文件:/user/myname/output_raceData/part-m-00000

FileSystemAPI获取实例

接口官方文档

https://hadoop.apache.org/docs/stable/api/index.html

获取FileSystem

get(Configuration conf)
Returns the configured FileSystem implementation.

get(URI uri, Configuration conf)
Get a FileSystem for this URI's scheme and authority.

get(URI uri, Configuration conf, String user)
Get a FileSystem instance based on the uri, the passed in configuration and the user.

FileSystemAPI实例

设置目录的创建和删除权限:

```
[root@master hadoop]# hdfs dfs -chmod -R 777 /user
[root@master hadoop]#
```

源码参考:

```
package chap5_hdfsfs;

import java.io.*;
import java.net.URI;
import java.net.URISyntaxException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;

public class HdfsApi {
    public static void main(String[] args) throws IOException {
//        S1_ListDir列出文件夹
        ListDir(args);
//        S2_CreateDir创建目录
        CreateDir(args);
//        S3_CopyToLocal下载文件
        CopyToLocal(args);
//      S4_CopyFromLocal上传文件
        CopyFromLocal(args);
//      S5_CatFile读写文件
        CatFile(args);
//      S6_ListFile列出文件
        ListFile(args);
//      S7_DelFile删除文件
        DelFile(args);
//      S8_DelPath删除目录
        DelPath(args);

    }
```

```
30
31      /***
32       * S1_ListDir列出文件夹
33       * @param args
34       * @throws IOException
35       */
36      public static void ListDir(String[] args) throws IOException {
37          //获取配置
38          Configuration conf=new Configuration();
39          conf.set("fs.defaultFS", "hdfs://master:8020/");
40          //获取文件系统
41          FileSystem fs=FileSystem.get(conf);
42          //声明文件路径
43          Path path=new Path("/user/limm/");
44          //获取文件列表
45          FileStatus[] fileStatuses=fs.listStatus(path);
46          //遍历文件列表
47          for (FileStatus file : fileStatuses) {
48              //判断是否是文件夹还是文件
49              if(file.isDirectory()){
50                  System.out.println("Dir:" +file.getPath().toString());
51              }
52              else if (file.isFile())
53              {
54                  System.out.println("File:" +file.getPath().toString());
55              }
56              else
57              {
58                  System.out.println("Other:" +file.getPath().toString());
59              }
60          }
61      }
62
63      /***
64       * S2_CreateDir创建目录
65       * @param args
66       * @throws IOException
67       */
68      public static void CreateDir(String[] args) throws IOException {
69          //获取配置
70          Configuration conf=new Configuration();
71          //获取文件系统
72          FileSystem fs= null;
73          try {
74              fs = FileSystem.get(new
URI("hdfs://master:8020/"),conf,"root");
75          } catch (InterruptedException e) {
76              throw new RuntimeException(e);
77          } catch (URISyntaxException e) {
78              throw new RuntimeException(e);
79          }
80          ////声明创建的目录
81          Path path=new Path("/user/limm/temp"); //limm改为本人
82          //调用mkdirs函数创建目录
83          fs.mkdirs(path);
```

```
        //关闭文件
        fs.close();
    }

    /***
     * S3_CopyToLocal下载文件
     * @param args
     * @throws IOException
     */
    public static void CopyToLocal(String[] args) throws IOException {
        //获取配置
        Configuration conf=new Configuration();
        conf.set("fs.defaultFS", "hdfs://master:8020/");
        //获取文件系统
        FileSystem fs=FileSystem.get(conf);
        //声明源文件路径和目标路径
        Path fromPath=new Path("/user/limm/raceData.csv");
        Path toPath=new Path("D:/tmp");
        //调用copyToLocalFile方法下载文件到本地
        fs.copyToLocalFile(false, fromPath, toPath, true);
        //关闭文件系统
        fs.close();
    }

    /***
     * S4_CopyFromLocal上传文件
     * @param args
     * @throws IOException
     */
    public static void CopyFromLocal(String[] args) throws IOException {
        //获取配置
        Configuration conf = new Configuration();
        //获取文件系统
        FileSystem fs= null;
        try {
            fs = FileSystem.get(new
URI("hdfs://master:8020/"),conf,"root");
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        } catch (URISyntaxException e) {
            throw new RuntimeException(e);
        }
        //声明源文件路径和目标路径
        Path fromPath = new Path("D:/tmp/raceData.csv");
        Path toPath = new Path("/user/limm/temp/user_log.txt");
        //调用copyFromLocalFile方法上传文件
        fs.copyFromLocalFile(fromPath,toPath);
        //关闭文件系统
        fs.close();
    }

    /***
     * S5_CatFile读写文件
     * @param args
     * @throws IOException
```

```
     */
    public static void CatFile(String[] args) throws IOException {
        //获取配置
        Configuration conf=new Configuration();
        //获取文件系统
        FileSystem fs= null;
        try {
            fs = FileSystem.get(new
URI("hdfs://master:8020/"),conf,"root");
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        } catch (URISyntaxException e) {
            throw new RuntimeException(e);
        }
        //声明查看的路径
        Path path=new Path("/user/limm/temp/user_log.txt");
        //创建新文件
        Path newPath=new Path("/user/limm/temp/new_user_log.txt");
        fs.delete(newPath,true);
        FSDataOutputStream os=fs.create(newPath);
        //获取指定文件的数据字节流
        FSDataInputStream is=fs.open(path);
        //读取文件内容并写入到新文件
        BufferedReader br=new BufferedReader(new InputStreamReader(is,"utf-
8"));
        BufferedWriter bw=new BufferedWriter(new
OutputStreamWriter(os,"utf-8"));
        String line="";
        while((line=br.readLine())!=null){
            bw.write(line);
            bw.newLine();
        }
        //关闭数据字节流
        bw.close();
        os.close();
        br.close();
        is.close();
        //关闭文件系统
        fs.close();
    }

    /***
     * S6_ListFile列出文件
     * @param args
     * @throws IOException
     */
    public static void ListFile(String[] args) throws IOException {
        //获取配置
        Configuration conf=new Configuration();
        conf.set("fs.defaultFS", "hdfs://master:8020/");
        //获取文件系统
        FileSystem fs=FileSystem.get(conf);
        //声明文件路径
        Path path=new Path("/user/limm");
        //获取文件列表
```

```
        FileStatus[] fileStatuses=fs.listStatus(path);
        //遍历文件列表
        for (FileStatus file : fileStatuses) {
            //判断是否是文件夹
            if(file.isFile()){
                System.out.println(file.getPath().toString());
            }
        }
        //关闭文件系统
        fs.close();
    }

    /***
     * S7_DelFile删除文件
     * @param args
     * @throws IOException
     */
    public static void DelFile(String[] args) throws IOException {
        //获取配置
        Configuration conf=new Configuration();
        //获取文件系统
        FileSystem fs= null;
        try {
            fs = FileSystem.get(new
URI("hdfs://master:8020/"),conf,"root");
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        } catch (URISyntaxException e) {
            throw new RuntimeException(e);
        }
        //声明文件路径
        Path path=new Path("/user/limm/temp/user_log.txt");
        //删除文件
        fs.delete(path, true);
        //关闭文件系统
        fs.close();
    }

    /***
     * S8_DelPath删除目录
     * @param args
     * @throws IOException
     */
    public static void DelPath(String[] args) throws IOException {
        //获取配置
        Configuration conf=new Configuration();
        //获取文件系统
        FileSystem fs= null;
        try {
            fs = FileSystem.get(new
URI("hdfs://master:8020/"),conf,"root");
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        } catch (URISyntaxException e) {
            throw new RuntimeException(e);
```

```
        }
        //声明文件路径
        Path path=new Path("/user/limm/temp/");
        //删除文件
        fs.delete(path, true);
        //关闭文件系统
        fs.close();
    }
}

```

运行结果

```
"C:\Program Files\Java\jdk1.8.0_281\bin\java.exe" "-javaagent:D:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.1\lib\idea_rt.jar=56067:D:\Program Files\JetBrains\IntelliJ IDEA Community ...
File:hdfs://master:8020/user/myname/1.txt
File:hdfs://master:8020/user/myname/11.txt
File:hdfs://master:8020/user/myname/2.txt
File:hdfs://master:8020/user/myname/email_log.txt
File:hdfs://master:8020/user/myname/hadoop-root-namenode-master.log
Dir:hdfs://master:8020/user/myname/output111
Dir:hdfs://master:8020/user/myname/output_AccessCount
Dir:hdfs://master:8020/user/myname/output_AccessTimeSort
Dir:hdfs://master:8020/user/myname/output_email_log
Dir:hdfs://master:8020/user/myname/output_namenode_wordmean
Dir:hdfs://master:8020/user/myname/output_raceData
File:hdfs://master:8020/user/myname/raceData.csv
Dir:hdfs://master:8020/user/myname/temp
hdfs://master:8020/user/myname/1.txt
hdfs://master:8020/user/myname/11.txt
hdfs://master:8020/user/myname/2.txt
hdfs://master:8020/user/myname/email_log.txt
hdfs://master:8020/user/myname/hadoop-root-namenode-master.log
hdfs://master:8020/user/myname/raceData.csv

Process finished with exit code 0

```

读取序列化日志任务实现

源码参考

```
package main.java;

import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;

import org.apache.hadoop.conf.Configuration;
```

```java
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;
public class DownloadFile {
    public static void main(String[] args) throws IOException {
        //获取配置
        Configuration conf = new Configuration();
        conf.set("fs.defaultFS", "hdfs://master:8020/");
        //获取文件系统
        FileSystem fs = FileSystem.get(conf);
        //获取SequenceFile.Reader对象
        SequenceFile.Reader reader = new SequenceFile.Reader(fs,
                new Path("/user/myname/output_raceData/part-m-00000"), conf);
        //获取序列化文件中使用的键值类型
        Text key = new Text();
        Text value = new Text();
        BufferedWriter out = new BufferedWriter(
                new OutputStreamWriter(new
FileOutputStream("d:\\tmp\\selectdata.txt", true)));
        while (reader.next(key, value)) {
            out.write(key.toString() + "\t" + value.toString() + "\r\n");
        }
        out.close();
        reader.close();
        System.out.println("end");
    }
}

```

运行结果

```
"C:\Program Files\Java\jdk1.8.0_281\bin\java.exe" "-javaagent:D:\Program
Files\JetBrains\IntelliJ IDEA Community Edition
2023.1\lib\idea_rt.jar=56667:D:\Program Files\JetBrains\IntelliJ IDEA
Community ......" main.java.DownloadFile
log4j:WARN No appenders could be found for logger
(org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more
info.
end

Process finished with exit code 0

```

结果文件

打开d:\tmp\selectdata.txt文件，预期内容无乱码，内容如：

```
103250  2021/1/1
103250  2021/1/1
103250  2021/1/1
103250  2021/1/1
103408  2021/1/1
104664  2021/1/1
104664  2021/1/1
104664  2021/1/1
104010  2021/1/1
......
```

任务5.3优化日志文件统计程序

任务描述

在Hadoop中，Mapper和Reducer处理的具体对象是键值对。

键值对的类型有Text、IntWritable、DoubleWritable等，在实际业务中，数据构成与逻辑更加复杂，键或值可能是由多个元素组成的，需要用户根据实际情况自定义键值对的类型。

Map端的输出结果是经过网络传输到对应的Reduce端的。当Map端的输出数据量特别大时，网络传输可能成为影响处理效率的一大因素。为了提高整体处理效率，Hadoop提供了用于优化的组件Combiner与Partitioner，可以帮助数据在进入Reduce阶段前进行一系列的合并与分区处理。另外，Hadoop提供了在执行MapReduce程序过程中的计数功能，用户也可以根据需要进行个性化的计数设置。

本小节的任务如下。

1. 使用MapReduce读取序列化文件，统计竞赛网站用户在2021年1月份和2月份每日的登录次数；
2. 要求最终的输出结果根据月份分别保存到两个不同的文件中；
3. 同时要求分别统计输入记录中1月份和2月份的记录数以及输出结果中1月份和2月份的记录数。

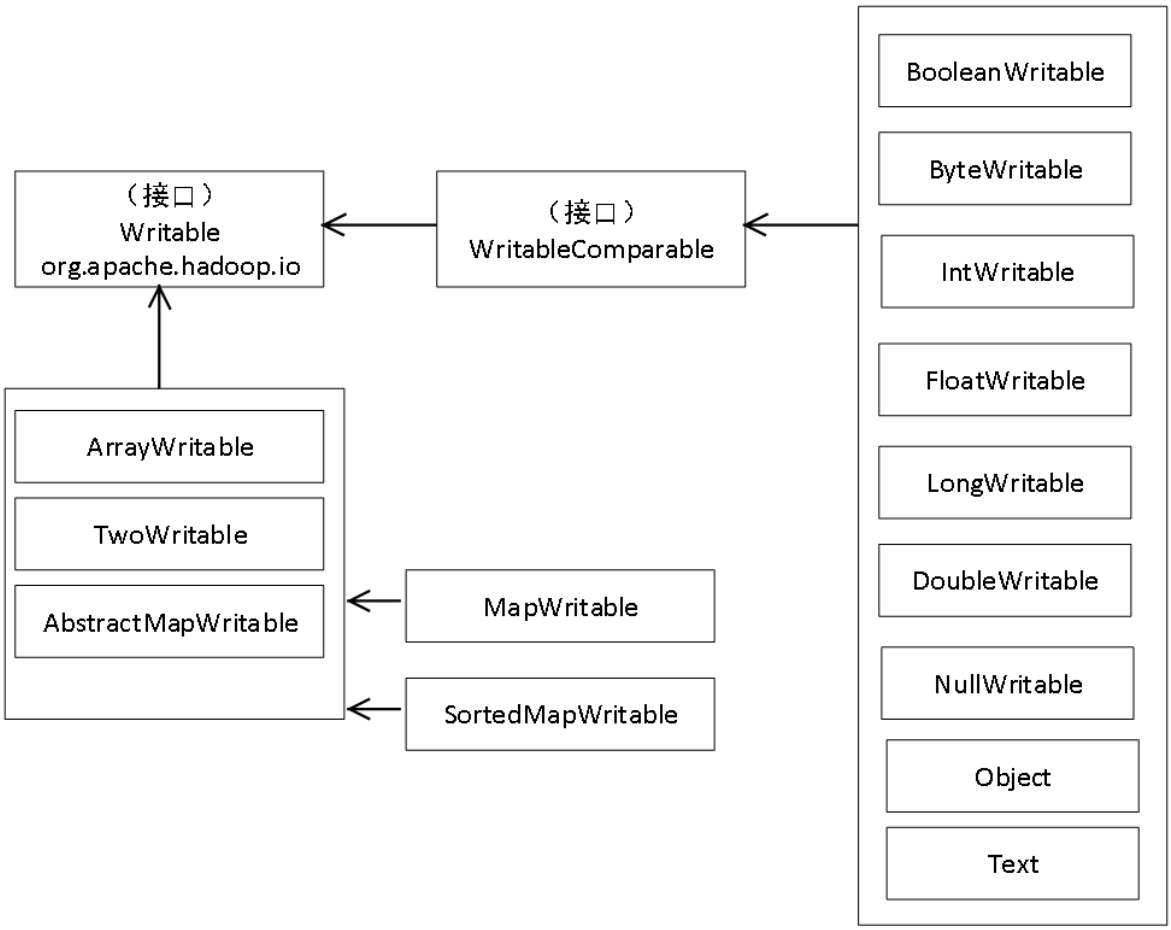
MR优化关键因素：自定义键值、Combiner、Partitioner、记数器

自定义键值类型

内置数据类型继承关系

内置类型->WritableComparable->Writable

常用内置数据类型



类型	解释
BooleanWritable	标准布尔型，相当于Java数据类型里面的boolean，当<key,value>中的key或value为布尔型时使用
ByteWritable	单字节型，相当于Java数据类型里面的byte，当<key,value>中的key或value为单字节类型时使用
DoubleWritable	双精度浮点型，相当于Java数据类型里面的double，当<key,value>中的key或value为双字节 类型时使用
FloatWritable	单精度浮点型，相当于Java数据类型里面的float，当<key,value>中的key或value为浮点类型时使用
IntWritable	整型，相当于Java数据类型里面的int，当<key,value>中的key或value为整型时使用
LongWritable	长整型，相当于Java数据类型里面的long，当<key,value>中的key或value为长整型时使用
Text	使用UTF-8格式存储的文本，在Java数据类型中主要针对String类型
NullWritable	空值，当<key,value>中的key或value为空时使用
Object	超级数据类型，是所有数据类型的根类

自定义键类型

需实现WritableComarable接口(WritableComarable自身又实现了Writable接口)，比自定义值类型要多实现compareTo方法

实现Writable接口

- 需实现方法readFields
- 需实现方法write
- 需实现方法compareTo
- 继承自Object类
    - 需重写方法toString()

自定义值类型

需实现Writable接口

- 需实现方法readFields
- 需实现方法write
- 继承自Object类
    - 需重写方法toString()

Combiner:

Combiner定义

在MapReduce作业运行过程中，节点上的每一个Map可能会产生大量的本地输出。如果有几十亿条记录，那么Map阶段可能输出几十亿个键值对，传输至Reduce端的数据量是巨大的，对集群的带宽要求非常高，Map和Reduce任务之间的数据传输代价也会增大，从而对整个工作的效率产生影响。为了提高MapReudce作业的工作效率，Hadoop允许用户声明一个Combiner。Combiner是一个运行在Map端的“迷你Reduce”过程，它只处理单台机器生成的数据。

声明的Combiner继承的是Reducer类，其方法实现原理和Reduce的实现原理基本相同。不同的是，Combiner操作发生在Map端，或说Combiner运行在每一个运行Map任务的节点上。Combiner会接收特定节点上的Map输出作为输入，对Map输出的数据先进行一次合并，再将输出结果发送至Reducer端。需要注意的是，Combiner的加入不影响原逻辑，即Combiner不影响最终运行结果，影响的只是效率。

迷你Reduce

Combiner使用条件

- 仅当Reducer输入的键值对类型与Reducer输出的键值对类型一样时，
  并且计算逻辑不影响最终计算结果时，才可以用
- 如求和或者求最大值 时可以使用
- 如求平均值不可使用
    - 如数据:
        - Map1:4,5
        - Map2:3,6,2
    - 使用Combiner:
        - Map1:4,5=>(4+5)/2=4.5

        - Map2:3,6,2=>(3+6+2)/3=3.66667
        - Reducer:=(4.5+3.66667)/2=4.08333
    - 不使用Combiner
        - Reducer:=(4+5+3+6+2)/5=4

Combiner使用方法

- 申明Combiner
    - public class LogCountCombiner extends Reducer<MemberLogTime,IntWritable,MemberLogTime,IntWritable>{}
- 在main驱动类里配置Combiner
    - job.setCombinerClass(LogCombiner.class)
- 当Combiner逻辑与Reducer相同时配置Combiner
    - job.setCombinerClass(LogCountReducer.class)

Partitioner

Partitioner定义

- 功能是让Map对Key进行分区，从而将不同的key分发到不同的Reducer中进行处理
- 分区阶段发生在Map阶段之后，在Reduce阶段之前
- 分区的数量等于Reducer的个数。个数=job.setNumReduceTasks

Partitioner使用条件

- 一般情况，使用默认的HashPartitioner
    - 源码
        - HasPartitioner<K2,V2> implements Partitioner<K2,V2>{}
        - void configue(JobConf job){}
        - int getPartition(K2 key,V2 value,int numRecudeTasks){}
- 比如要求按月统计登录次数并且输出到不同的文件里

Partitioner使用方法

- 自定义Partitioner，并继续自Partitioner<K2,V2>,并重写getPartition方法
- 如要实现统计结果根据不同的月份分发到不同的输出文件里，在getPartition的实现方法里，分别使用0、1与numPartitions相除取余
- 在驱动类main方法中设置Partitioner类和Reducer个数

记数器

记数器定义

计数器是Hadoop框架使用的一种对统计信息收集的手段、主要用于对数据的控制及收集统计信息

记数器分类

MapReduce任务计数器

org.apache.hadoop.mapreduce.TaskCounter

文件系统 计数器

org.apache.hadoop.mapreduce.FileSystemCounter

输入文件计数器

输出文件计数器

作业计数器

任务实现将日志按月份统计

任务目标:

本小节的任务是通过MapReduce编程实现用户在2021年1月份和2月份每天的登录次数统计

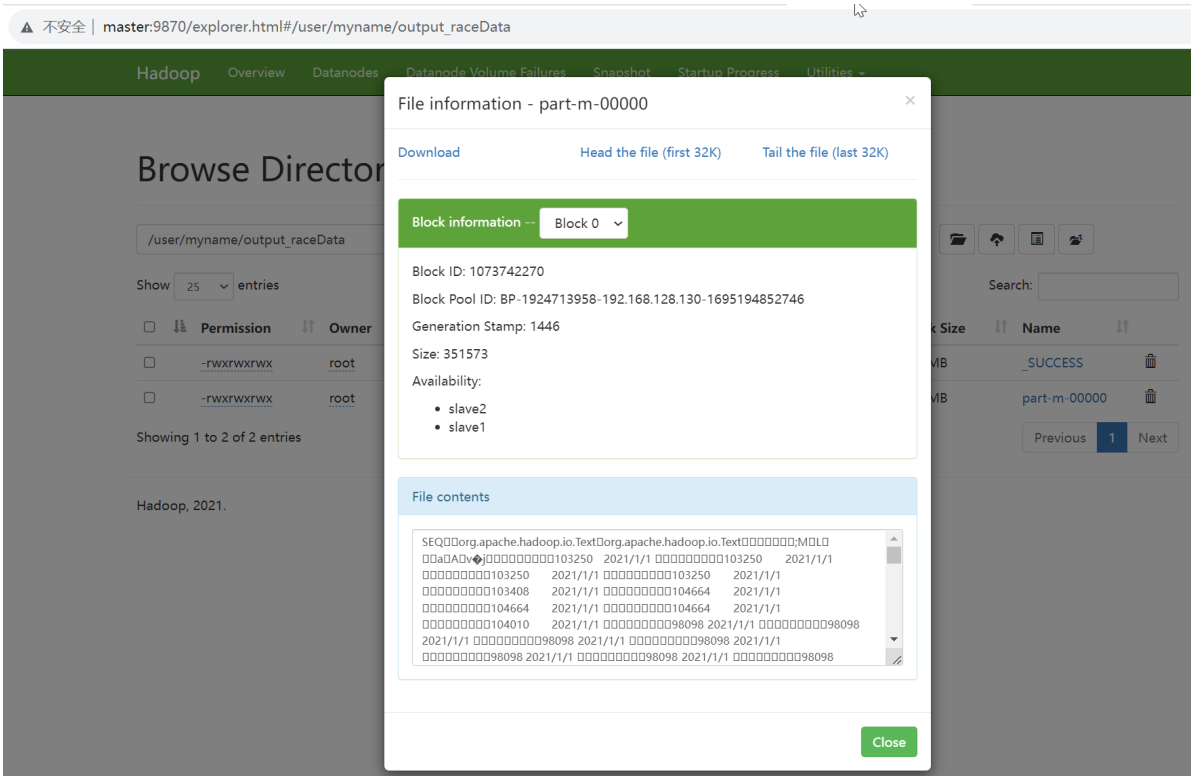
任务实现步骤

- 1,读取HDFS中任务5.1生成的序列化文件
  - /user/myname/output_raceData/part-m-00000
- 2,InputFormat读入分片数据
- 3,定义MemberLogTime自定义键类型
- 4,定义Map,并使用自定义键类型MemberLogTime,同时还应用自定义计数量来表示1月或2月的登录次数
- 5,加入Combiner组件来减少Reducer的负担
- 6,任务结束前将结果按月份输出到两个文件,用到了Partitioner
- 7,最后编写驱动类Main方法

数据准备

基于5.1的输出结果,请确认以下hdfs上的文件是否有内容
/user/myname/output_raceData/part-m-00000

如图:



源码实现

在wordcount工程下创建包logcount,以下类都在该包下创建

自定义类型MemberLogTime

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

import org.apache.hadoop.io.WritableComparable;

public class MemberLogTime implements WritableComparable<MemberLogTime>{
private String member_name;
private String logTime;
public MemberLogTime() {

}
public MemberLogTime(String member_name,String logTime){
    this.member_name=member_name;
    this.logTime=logTime;
}
public String getMember_name() {
    return member_name;
}
public void setMember_name(String member_name) {
    this.member_name = member_name;
}
public String getLogTime() {
    return logTime;
}
public void setLogTime(String logTime) {
    this.logTime = logTime;
}
@Override
public void readFields(DataInput in) throws IOException {
    this.member_name=in.readUTF();
    this.logTime=in.readUTF();
}

@Override
public void write(DataOutput out) throws IOException {
    out.writeUTF(member_name);
    out.writeUTF(logTime);
}
@Override
public int compareTo(MemberLogTime o) {
    return this.getMember_name().compareTo(o.getMember_name());
}
@Override
public String toString() {
    return this.member_name+","+this.logTime;
}
}
```

LogCountMapper实现

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class LogCountMapper extends Mapper<Text, Text, MemberLogTime,
IntWritable> {
private MemberLogTime mt=new MemberLogTime();
private IntWritable one=new IntWritable(1);
enum LogCounter{
    January,
    February
}
@Override
protected void map(Text key, Text value, Mapper<Text, Text, MemberLogTime,
IntWritable>.Context context)
        throws IOException, InterruptedException {
    String member_name=key.toString();
    String logTime=value.toString();
    if(logTime.contains("2021-01")){
        context.getCounter(LogCounter.January).increment(1);;
    }else if(logTime.contains("2021-02")){
        context.getCounter(LogCounter.February).increment(1);;
    }
    mt.setMember_name(member_name);
    mt.setLogTime(logTime);
    context.write(mt, one);
}
}
```

LogCountCombiner实现

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class LogCountCombiner extends Reducer<MemberLogTime, IntWritable,
MemberLogTime, IntWritable> {
@Override
protected void reduce(MemberLogTime key, Iterable<IntWritable> value,
        Reducer<MemberLogTime, IntWritable, MemberLogTime,
IntWritable>.Context context)
                throws IOException, InterruptedException {
    int sum=0;
    for (IntWritable val : value) {
        sum+=val.get();
    }
    context.write(key, new IntWritable(sum));
}
```

```
}
```

LogCountPartitioner实现

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Partitioner;
public class LogCountPartitioner extends Partitioner<MemberLogTime,
IntWritable> {
@Override
public int getPartition(MemberLogTime key, IntWritable value, int
numPartitions) {
    String date=key.getLogTime();
    if(date.contains("2016-01")){
        return 0%numPartitions;
    }else{
        return 1%numPartitions;
    }
}
}
```

LogCountReducer实现

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;
public class LogCountReducer extends Reducer<MemberLogTime, IntWritable,
MemberLogTime, IntWritable> {
@Override
protected void reduce(MemberLogTime key, Iterable<IntWritable> value,
        Reducer<MemberLogTime, IntWritable, MemberLogTime,
IntWritable>.Context context)
                throws IOException, InterruptedException {
    if(key.getLogTime().contains("2016-01")){
        context.getCounter("OutputCounter","JanuaryResult").increment(1);;
    }else if(key.getLogTime().contains("2016-02")){
        context.getCounter("OutputCounter", "FebruaryResult").increment(1);
    }
    int sum=0;
    for (IntWritable val : value) {
        sum+=val.get();
    }
    context.write(key, new IntWritable(sum));
}
}
```

LogCount驱动类实现

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.SequenceFileAsTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class LogCount {

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.set("fs.defaultFS", "hdfs://master:9864");
   String[] otherArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();
   if (otherArgs.length < 2) {
   otherArgs= new String[]{"/user/myname/output_SelectData/part-m-
00000","/user/myname/output_MonthData"};
   //注意myname处改为自己姓名全拼
   }
    Job job = Job.getInstance(conf, "logcount");
    job.setJarByClass(LogCount.class);
    job.setMapperClass(LogCountMapper.class);
    job.setReducerClass(LogCountReducer.class);
    job.setCombinerClass(LogCountCombiner.class);
    job.setPartitionerClass(LogCountPartitioner.class);
    job.setNumReduceTasks(2);

•

job.setOutputKeyClass(MemberLogTime.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(SequenceFileAsTextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);

FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileSystem.get(conf).delete(new Path(otherArgs[1]), true);
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

System.err.println(job.waitForCompletion(true) ? -1 : 1);

}

}
```

调试运行

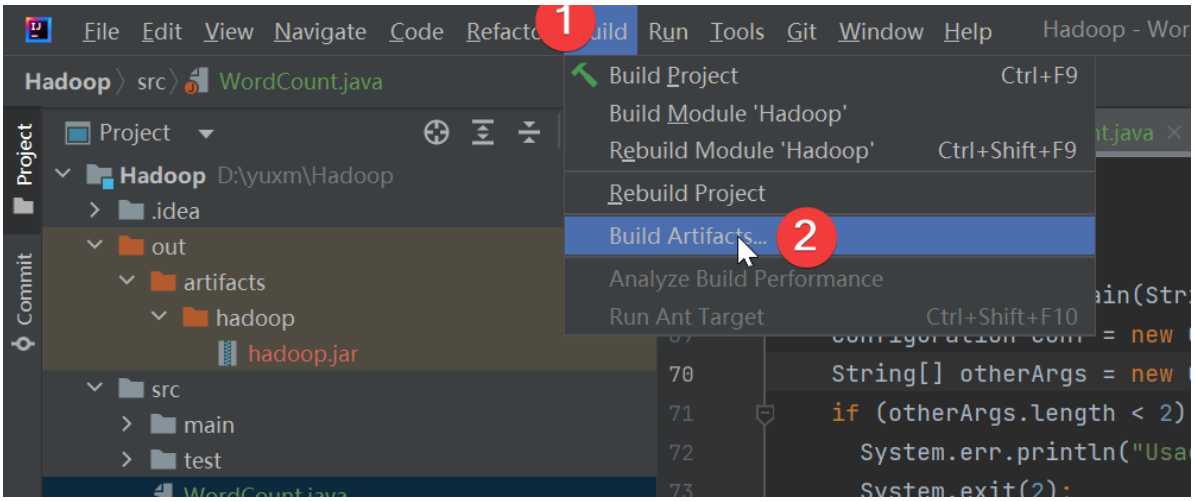
运行LogCount->main()

验证结果

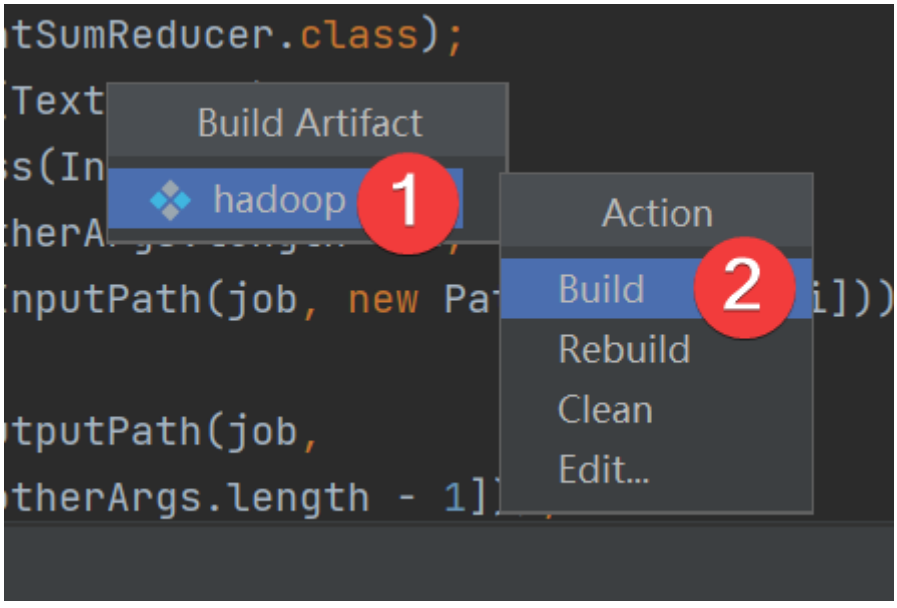
- 1,查看/user/myname/output_MonthData目录下是否有两文件
  - part-r-00000
  - part-r-00001
- 2,查看part-r-00000和part-r-00001内容
  - 如:Aaron,2016-01-04　99

编译项目

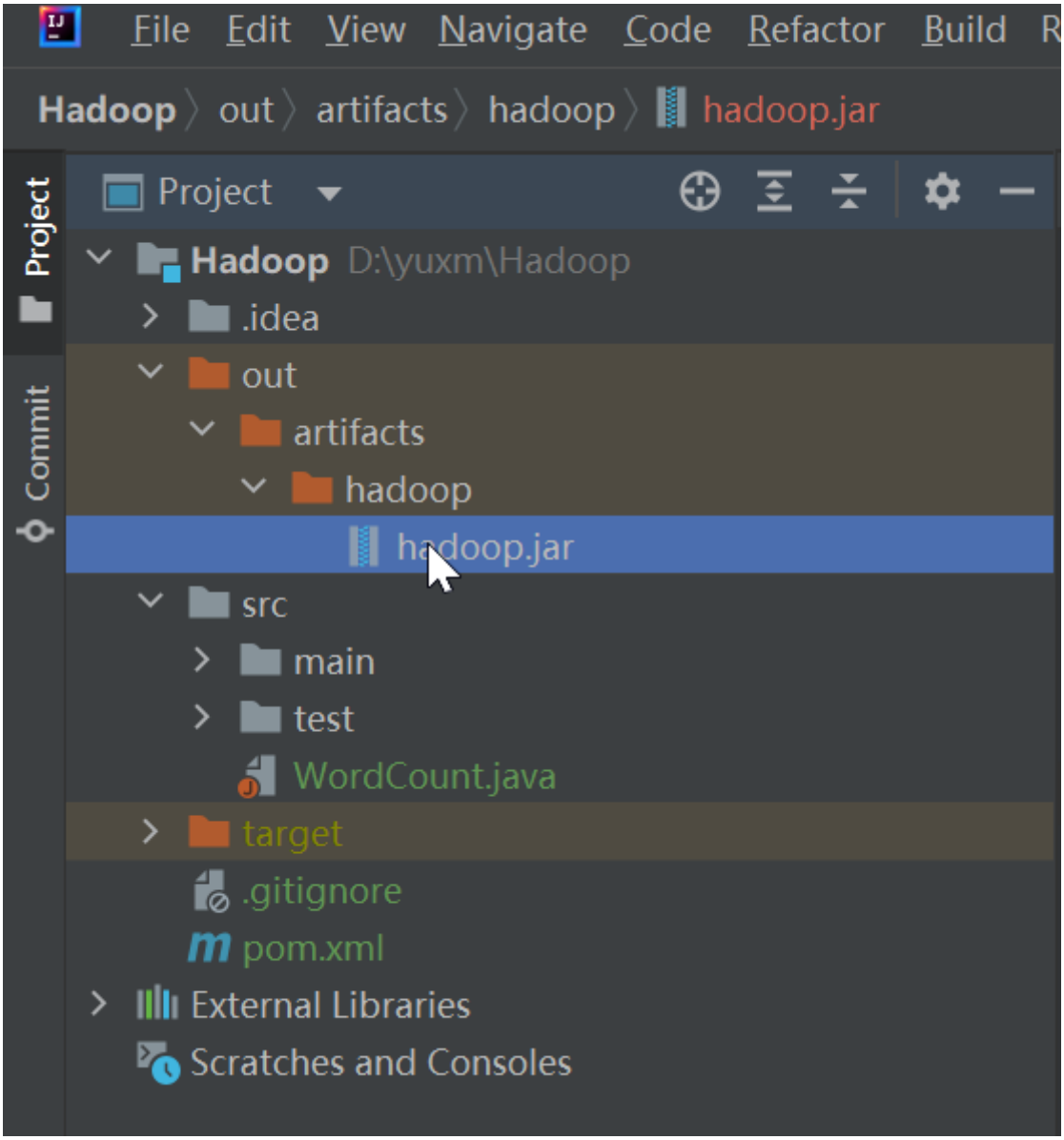
点菜单build->BuildArtifacts...



弹出菜单：



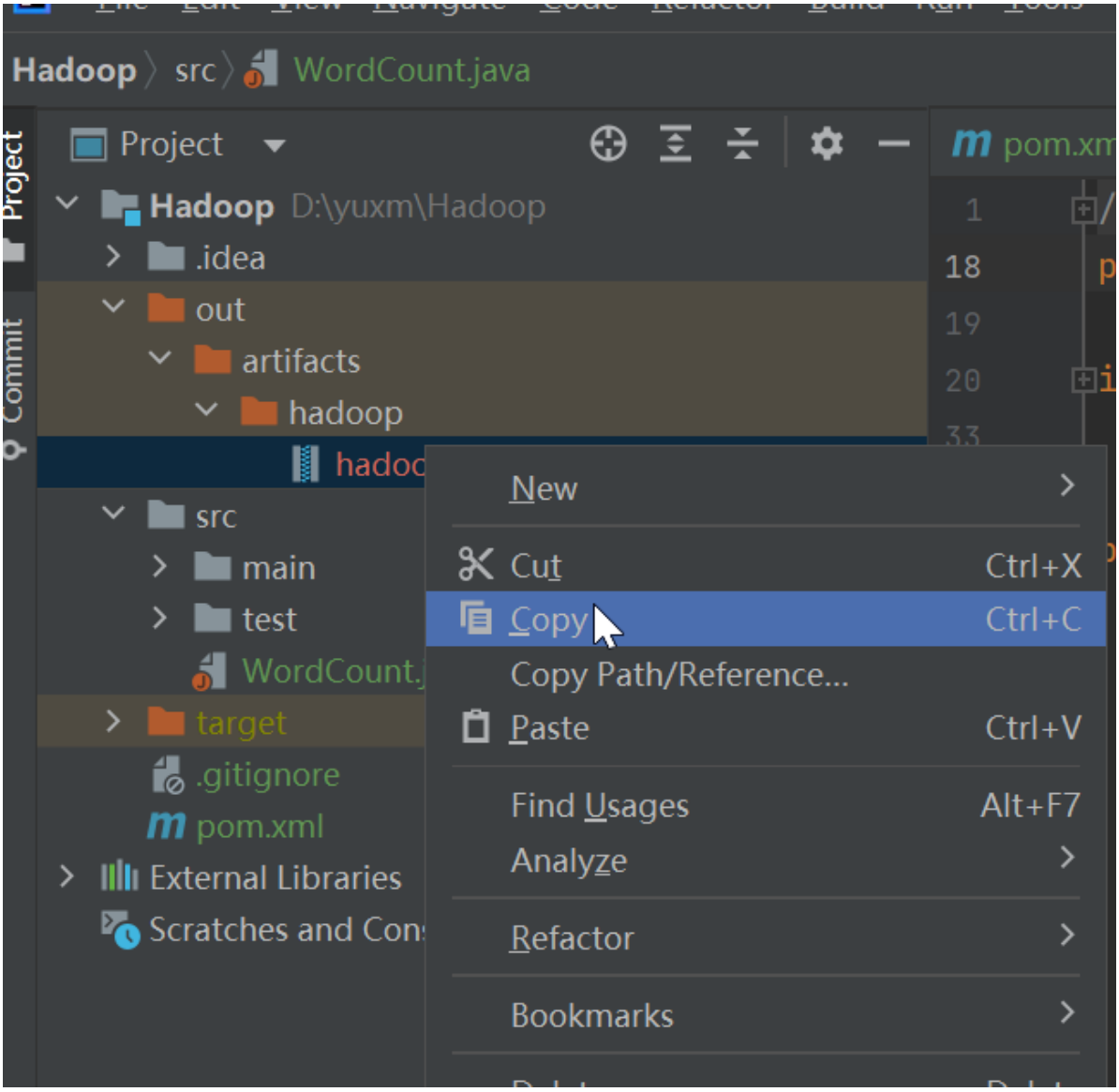
编译完成后，会在左侧源码树图上：out->artifacts-> hadoop下出现hadoop.jar文件，这就是编译后的应用程序：



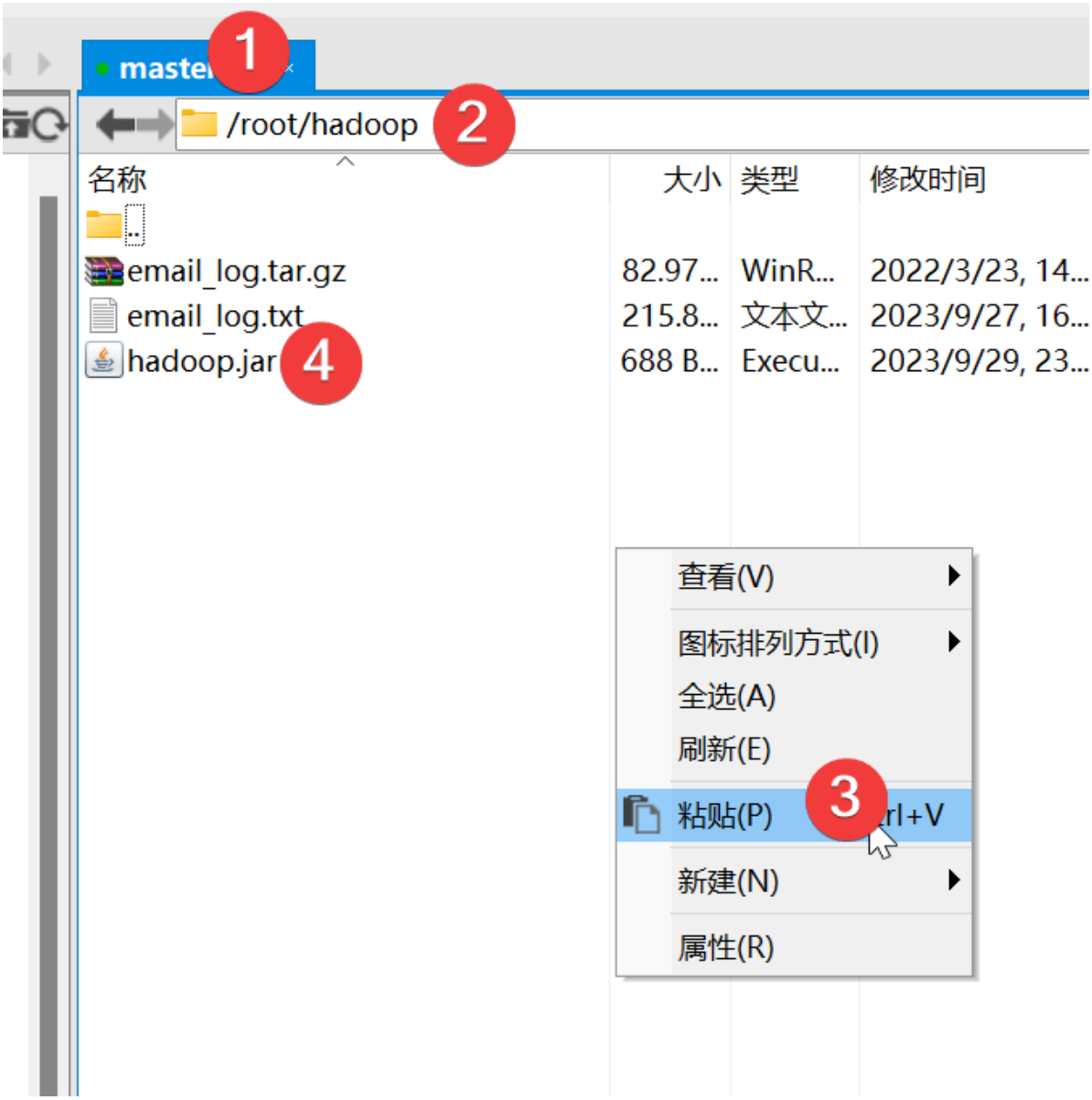
上传编译后文件

上传编译后的文件到集群

右击上图的hadoop.jar,选择菜单“复制”



然后打开xftp工具，连接到master虚拟主机上，进入 /root/hadoop目录，右击空白的地方，在弹出的菜单中选择“粘贴”



运行编译后文件

使用xshell或crt打开master,进入/root/hadoop目录，执行编译后的文件hadoop.jar：

```
cd /root/hadoop
hadoop jar ./hadoop.jar chap5_logcount.LogCount \
/user/myname/output_raceData/part-m-00000 \
/user/myname/output_MonthDataa
```

运行结果如：

```

```

任务5.4.IDEA中打包并提交MR程序

任务描述

在IDEA中编写MapReduce程序后，一般先编译生成JAR包，再上传到集群服节点，通过hadoop jar命令提交至集群运行，需要在不同平台系统间切换，并且各个环节需要手动处理，处理流程比较烦琐且效率低。

为了更简便、更高效地提交并运行MapReduce任务，本小节的任务如下。

1. 介绍MapReduce任务在工作环境中更为简便地提交流程。
2. 在IDEA中实现自动打包MapReduce任务并远程连接Hadoop集群。
3. 直接提交并运行上一小节编写的MapReduce程序，并显示执行过程中的输出日志。

5.4.1.MR中进行传递参数

在本章前几节的MapReduce编程中，除了输入路径和输出路径是提交MapReduce需要输入的参数外，其他参数（如分隔符）均是直接写进程序中。如果输入数据的分隔符有所改动，那么程序里使用的分隔符也要进行相应地修改，再重新编译打包程序，再次提交至集群运行，操作十分繁琐。解决方案是将分隔符作为参数传递到程序中，当输入数据的分隔符改变时，直接修改传递的参数即可。

Hadoop的Configuration配置类提供了许多设置传递参数属性名称的方法，Configuration配置类中设置参数方法的参数说明如下。name：为参数设置的属性名称。value：参数值。source：该配置值的来源（用于调试）。pattern：模式值。

修饰符和返回类型	方法
void	set(String name,String value)
void	set(String name,String value,String source)
void	setBoolean(String name,boolean value)
void	setDouble(String name,double value)
void	setFloat(String name,float value)
void	setInt(String name,int value)
void	setLong(String name,long value)
void	setPattern(String name,Pattern pattern)

set方法使用

```
Configuration conf= getConf();
conf.set("inputPath",args[0]);
conf.set("outputPath",args[1]);
conf.set("splitter",args[2]);
```

get方法使用

- String inputPath=context.getConfiguration().get("inputPath")
- String outputPath=context.getConfiguration().get("outputPath")
- String splitter=context.getConfiguration().get("splitter")

5.4.2.使用Hadoop辅助类ToolRunner

MapReduce程序的Driver驱动类中，main()方法进行了MapReduce作业的相关配置，再通过命令行的方式运行作业。为了简化命令行方式，Hadoop自带了一些辅助类，其中GenericOptionsParser是一个解释常用的Hadoop命令行选项类，GenericOptionsParser类可以根据需要为Configuration对象设置相应的取值。通常不直接使用GenericOptionsParser类，更方便的方式是实现Tool接口，通过ToolRunner运行应用程序，ToolRunner内部调用了GenericOptionsParser类。

以统计某竞赛网站用户在2012年1月份和2月份每天的登录次数为例。首先上一小节的任务实现中实现的驱动类，使驱动类继承Hadoop配置类Configured，并实现Tool接口以及Tool接口中的run(String[] args)方法，在run(String[] args)方法中进行MapReduce作业的相关配置，再编写一个main()方法，调用ToolRunner中的run(Configuration conf,Tool tool,String[] args)方法，传递相关参数。main()方法中使用了ToolRunner的run()方法，并且将参数值写在main()方法中，因此使用hadoop jar命令运行程序时就不需要写参数值。打包程序并命名为logcount.jar，在master节点中输入hadoop jar命令即可成功提交MapReduce任务。

5.4.3.自动打包并提交MapReduce任务

当每一次运行MapReduce程序时都要先将程序编译打成JAR包，再上传至Linux文件系统中，再通过hadoop jar命令提交并运行MapReduce程序，这种提交MapReduce任务的方式是比较繁琐的。为了提高开发效率，开发者可以通过IDEA将MapReduce任务直接提交至Hadoop集群中。

在IDEA直接将MapReduce任务提交到Hadoop集群中，需要设置连接Hadoop集群的配置，包括使用跨平台提交任务、指定NameNode、指定使用YARN框架、指定ResourceManager、指定资源分配器、指定HistoryServer以及指定JAR包的存放路径。

以统计某竞赛网站用户在2012年1月份和2月份每天的登录次数为例，在驱动类中添加一个连接Hadoop集群配置的getMyConfiguration()方法。在定义了获取集群配置的方法后，需要在驱动类的run()方法中调用定义的获取集群配置的getMyConfiguration()方法。

添加了集群配置代码后，只需将程序打包，在代码中输入jar包本地路径，无需上传至Linux系统中，右键单击驱动类，选择“Run 'LogCount.main()'”命令即可运行MapReduce程序。

在工程的/src/main/resources目录下添加log4j.properties日志文件，该文件的目的是将运行程序的日志打印至控制台，方便用户查看日志。LogCount 程序在运行过程中将在IDEA的控制台打印输出日志信息。

虽然这样操作简便了一些步骤，但还是要将程序编译生成JAR包。因此为了省略程序打包这一步骤，用户可以自行编写一个将程序打成JAR包的工具类，通过调用这个工具类中的方法对程序进行打包，而不用手动打包。在编写好自动打包工具类之后，只需在设置Hadoop集群配置时将添加对LogCount调用自动打包的类作为JaAR包路径即可，无须将程序打包即可在IDEA中直接提交MapReduce任务。

5.4.4.任务实现

本小节的任务是在IDEA中实现直接提交并运行5.3小节编写的MapReduce程序，实现步骤如下。

1. 首先改写驱动类LogCount，在驱动类中继承Configured类中的Tool接口，重写Tool接口中的run()方法，并在main()方法里设置传递参数，参数包括输入数据路径、输出数据路径。在驱动类中编写连接Hadoop集群配置的getMyConfiguration()方法，声明配置，指定NameNode、YARN框架、ResourceManager和资源分配器，并在run()方法里调用getMyConfiguration()方法。
2. 编写一个自动打包的工具类，通过调用这个工具类中的方法对程序进行自动打包。
3. 在IDEA中直接运行驱动类，执行MapReduce程序。

数据准备

基于5.1的输出结果:

```
hadoop jar ./hadoop.jar main.java.SelectData /user/myname/raceData.csv
/user/myname/output_raceData
```

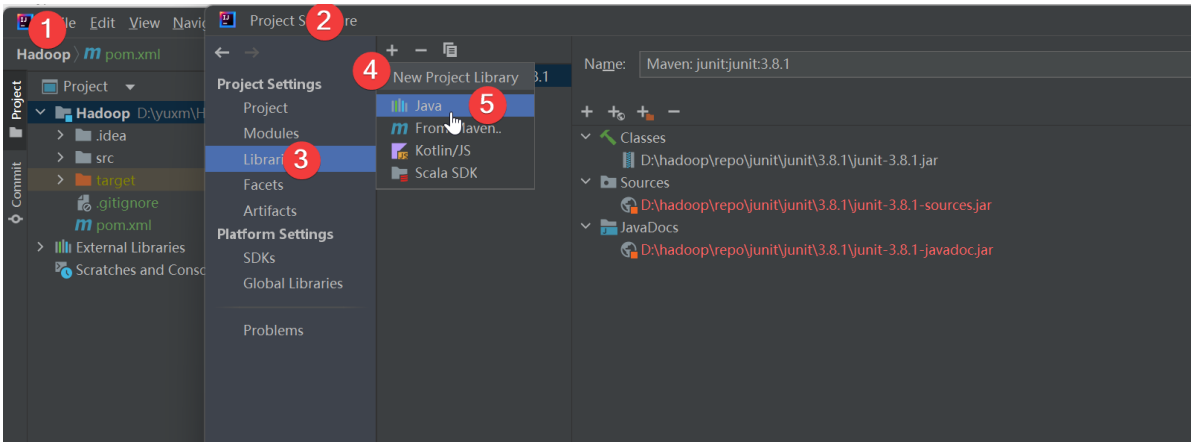
请确认以下hdfs上的文件是否有内容：/user/myname/output_raceData/part-m-00000，使用命令行查看：

```
hdfs dfs -cat /user/myname/output_raceData/part-m-00000
```

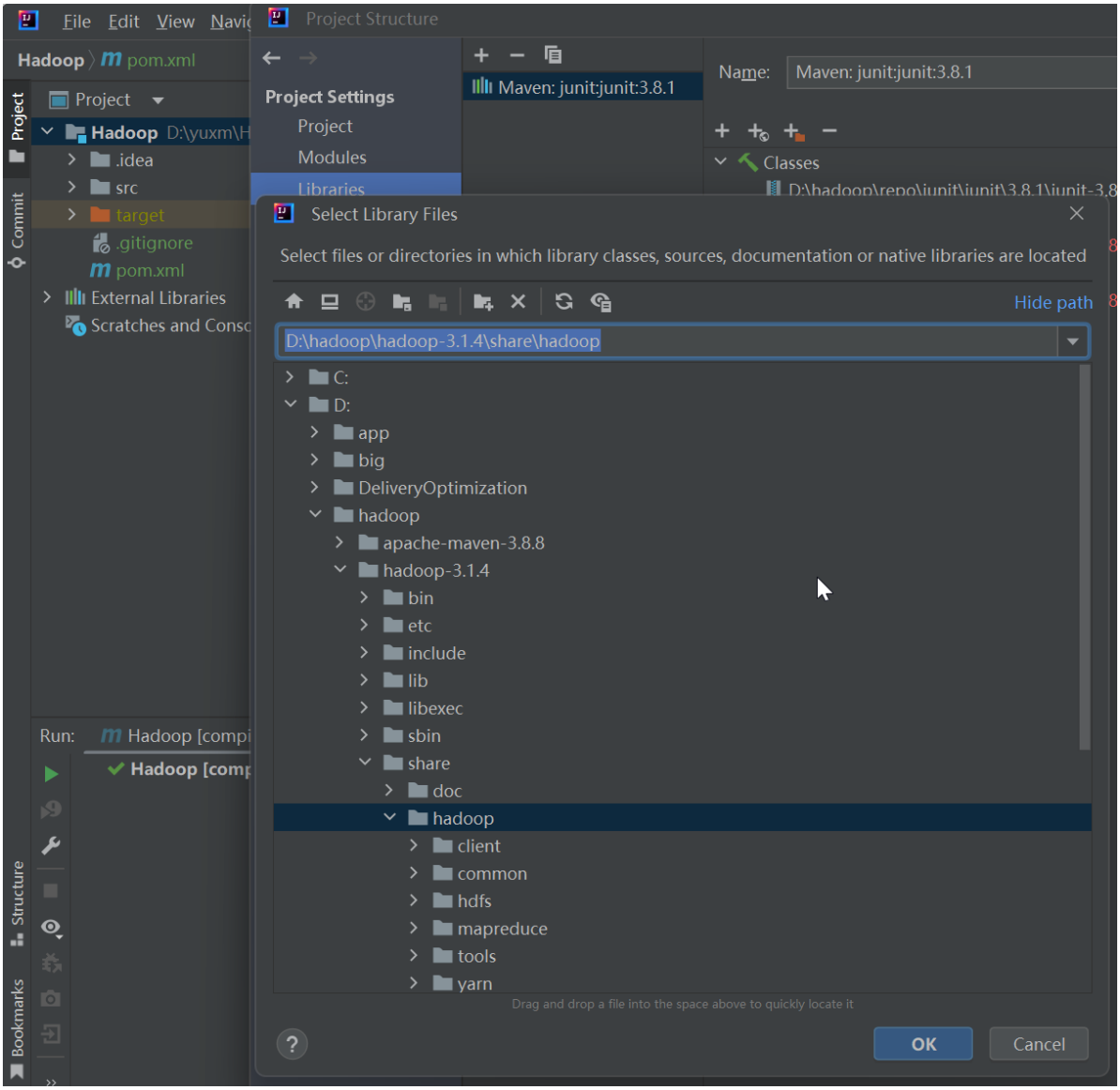
查看的内容比较多，可以使用Ctrl+C终止查看。

依赖包准备

单击“Project Structure”的对话框左侧的“Libraries ”选项，再右侧单击“+ ”选项，在弹出的选项栏中单击“Java ”选项



在弹出的对话框，选择要添加的jar包，选择我们前面已经解压的Hadoop安装目录:D:\hadoop\hadoop-3.1.4 下的\share\hadoop目录：D:\hadoop\hadoop-3.1.4\share\hadoop ,则将该目录下的全部JAR包导入项目，单击“OK ”按钮进入下一步



需要导入hadoop->share->hadoop下面的各子目录中，需要补充引用以下这些jar包：

```
引用目录：./hadoop/share/hadoop/common/lib/
avro-1.7.7.jar
woodstox-core-5.0.3.jar
slf4j-api-1.7.252.jar
stax2-api-3.1.4.jar
guava-27.0-jre.jar
commons-collections-3.2.2.jar
hadoop-auth-3.1.4.jar
htrace-core4-4.0.1-incubating.jar
commons-io-2.5.jar
commons-lang3-3.4.jar
jackson-annotations-2.9.10.jar
jackson-core-2.9.10.jar
jackson-databind-2.9.10.4.jar
jackson-core-asl-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
protobuf-java-2.5.0.jar
re2j-1.1.jar

引用目录：./hadoop/share/hadoop/hdfs/lib/所有jar
hadoop-annotations-3.1.3.jar
hadoop-auth-3.1.3.jar
```

```
htrace-core4-4.1.0-incubating.jar
hadoop-hdfs-3.1.1-tests.jar
hadoop-hdfs-client-3.1.1.jar
hadoop-hdfs-client-3.1.1-tests.jar
hadoop-hdfs-httpfs-3.1.1.jar
hadoop-hdfs-native-client-3.1.1.jar
hadoop-hdfs-native-client-3.1.1-tests.jar
hadoop-hdfs-nfs-3.1.1.jar
hadoop-hdfs-rbf-3.1.1.jar
hadoop-hdfs-rbf-3.1.1-tests.ja

```

C:\Program Files\Java\jdk1.8.0_281\bin

C:\Program~1\Java\jdk1.8.0_281\bin

在IDEA项目-》

源码实现

在已有的工程下创建logcount包,以下类都在该包下创建

自定义类型MemberLogTime

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

import org.apache.hadoop.io.WritableComparable;

public class MemberLogTime implements WritableComparable<MemberLogTime>{
private String member_name;
private String logTime;
public MemberLogTime() {

}
public MemberLogTime(String member_name,String logTime){
    this.member_name=member_name;
    this.logTime=logTime;
}
public String getMember_name() {
    return member_name;
}
public void setMember_name(String member_name) {
    this.member_name = member_name;
}
public String getLogTime() {
    return logTime;
}
public void setLogTime(String logTime) {
    this.logTime = logTime;
}
@Override
public void readFields(DataInput in) throws IOException {
    this.member_name=in.readUTF();
    this.logTime=in.readUTF();

```

```
}

@Override
public void write(DataOutput out) throws IOException {
    out.writeUTF(member_name);
    out.writeUTF(logTime);
}
@Override
public int compareTo(MemberLogTime o) {
    return this.getMember_name().compareTo(o.getMember_name());
}
@Override
public String toString() {
    return this.member_name+","+this.logTime;
}
}
```

LogCountMapper实现

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class LogCountMapper extends Mapper<Text, Text, MemberLogTime,
IntWritable> {
private MemberLogTime mt=new MemberLogTime();
private IntWritable one=new IntWritable(1);
enum LogCounter{
    January,
    February
}
@Override
protected void map(Text key, Text value, Mapper<Text, Text, MemberLogTime,
IntWritable>.Context context)
        throws IOException, InterruptedException {
    String member_name=key.toString();
    String logTime=value.toString();
    if(logTime.contains("2016-01")){
        context.getCounter(LogCounter.January).increment(1);;
    }else if(logTime.contains("2016-02")){
        context.getCounter(LogCounter.February).increment(1);;
    }
    mt.setMember_name(member_name);
    mt.setLogTime(logTime);
    context.write(mt, one);
}
}
```

LogCountCombiner实现

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class LogCountCombiner extends Reducer<MemberLogTime, IntWritable,
MemberLogTime, IntWritable> {
@Override
protected void reduce(MemberLogTime key, Iterable<IntWritable> value,
        Reducer<MemberLogTime, IntWritable, MemberLogTime,
IntWritable>.Context context)
                throws IOException, InterruptedException {
    int sum=0;
    for (IntWritable val : value) {
        sum+=val.get();
    }
    context.write(key, new IntWritable(sum));
}
}
```

LogCountPartitioner实现

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Partitioner;
public class LogCountPartitioner extends Partitioner<MemberLogTime,
IntWritable> {
@Override
public int getPartition(MemberLogTime key, IntWritable value, int
numPartitions) {
    String date=key.getLogTime();
    if(date.contains("2016-01")){
        return 0%numPartitions;
    }else{
        return 1%numPartitions;
    }
}
}
```

LogCountReducer实现

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;
public class LogCountReducer extends Reducer<MemberLogTime, IntWritable,
MemberLogTime, IntWritable> {
@Override
protected void reduce(MemberLogTime key, Iterable<IntWritable> value,
        Reducer<MemberLogTime, IntWritable, MemberLogTime,
IntWritable>.Context context)
```

```
                throws IOException, InterruptedException {
    if(key.getLogTime().contains("2016-01")){
        context.getCounter("OutputCounter","JanuaryResult").increment(1);;
    }else if(key.getLogTime().contains("2016-02")){
        context.getCounter("OutputCounter", "FebruaryResult").increment(1);
    }
    int sum=0;
    for (IntWritable val : value) {
        sum+=val.get();
    }
    context.write(key, new IntWritable(sum));
}
}
```

LogCount驱动类实现

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.SequenceFileAsTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
public class LogCount extends Configured implements Tool{
public static void main(String[] args){
    String[] myArgs={
            "/uer/myname/output_SelectData/part-m-00000",
            "/uer/myname/output_logcount"
    };


try {
    ToolRunner.run(LogCount.getMyConfiguration(), new LogCount(), myArgs);
} catch (Exception e) {
    e.printStackTrace();
}

}
@Override
public int run(String[] args) throws Exception {
    Configuration conf=LogCount.getMyConfiguration();
    Job job=Job.getInstance(conf, "logcount");
    job.setJarByClass(LogCount.class);
    job.setMapperClass(LogCountMapper.class);
    job.setReducerClass(LogCountReducer.class);
    job.setCombinerClass(LogCountCombiner.class);
    job.setPartitionerClass(LogCountPartitioner.class);
    job.setNumReduceTasks(2);
```

```
    job.setOutputKeyClass(MemberLogTime.class);
    job.setOutputValueClass(IntWritable.class);
    job.setInputFormatClass(SequenceFileAsTextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileSystem.get(conf).delete(new Path(args[1]), true);
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    return job.waitForCompletion(true)?-1:1;
}
public static Configuration getMyConfiguration(){
    //声明配置
    Configuration conf = new Configuration();
    conf.setBoolean("mapreduce.app-submission.cross-platform",true);
    conf.set("fs.defaultFS", "hdfs://master:9864");// 指定namenode
    conf.set("mapreduce.framework.name","yarn"); // 指定使用yarn框架
    String resourcenode="master";
    conf.set("yarn.resourcemanager.address", resourcenode+":8032"); // 指定
resourcemanager

conf.set("yarn.resourcemanager.scheduler.address",resourcenode+":8030");// 指
定资源分配器
    conf.set("mapreduce.jobhistory.address",resourcenode+":10020");
    conf.set("mapreduce.job.jar",JarUtil.jar(LogCount.class));
    return conf;
}
}
```

JarUtil类实现

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.jar.JarEntry;
import java.util.jar.JarOutputStream;
public class JarUtil {
  public static String jar(Class<?> cls){// 验证ok
    String outputJar =cls.getName()+".jar";
    String input = cls.getClassLoader().getResource("").getFile();
    input= input.substring(0,input.length()-1);
    input = input.substring(0,input.lastIndexOf("/")+1);
    input =input +"bin/";
    jar(input,outputJar);
    return outputJar;
  }
  private static void jar(String inputFileName, String outputFileName){
    JarOutputStream out = null;
    try{
      out = new JarOutputStream(new FileOutputStream(outputFileName));
      File f = new File(inputFileName);
      jar(out, f, "");
    }catch (Exception e){
      e.printStackTrace();
```

```
    }finally{
      try {
        out.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }

  }
  private static void jar(JarOutputStream out, File f, String base) throws
Exception {
    if (f.isDirectory()) {
      File[] fl = f.listFiles();
      base = base.length() == 0 ? "" : base + "/"; // 注意，这里用左斜杠
      for (int i = 0; i < fl.length; i++) {
        jar(out, fl[ i], base + fl[ i].getName());
      }
    } else {
      out.putNextEntry(new JarEntry(base));
      FileInputStream in = new FileInputStream(f);
      byte[] buffer = new byte[1024];
      int n = in.read(buffer);
      while (n != -1) {
        out.write(buffer, 0, n);
        n = in.read(buffer);
      }
      in.close();
    }
  }
}
```

配置文件

说明:配置文件放在src目录下

log4j.properties

```
log4j.rootLogger = INFO,stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy/MM/dd HH:mm:ss,SSS}-
%c{1}: %m%n
```

版本历史

- Ver1.1-20220121
  - 初始版本
- Ver1.2-20230523

    - 增加了常见问题,因为java版本未指定为1.8
- Ver1.2-20230828
    - 增加markdown版本

上一章节 下一章节