

实验名称

MapReduce分析数据然后再求数据的topN 问题

实验目的

熟练掌握MapReduce的编程

掌握setup,cleanup方法的使用

实验背景

我们都知道MapReduce的局限性是map读一行，执行一次，reduce是每一组执行一次，,但是当我们想全部得到数据之后，按照需求删选然后再输出怎么办呢？这时候只使用map和reduce显然是达不到目的的，此时我们想到了 setUp和cleanUp的特性，只执行一次。这样我们对于最终数据的过滤，然后输出要放在cleanUp中。这样就能实现对数据，不一组一组输出，而是全部拿到，最后过滤输出。经典运用常见，mapreduce分析数据然后再求数据的topN 问题。

实验原理

hadoop中的MapReduce框架里已经预定义了相关的接口，其中如Mapper类下的方法setup()和cleanup()

- setup(), 此方法被MapReduce框架仅且执行一次，在执行Map任务前，进行相关变量或者资源的集中初始化工作。若是将资源初始化工作放在方法map()中，导致Mapper任务在解析每一行输入时都会进行资源初始化工作，导致重复，程序运行效率不高！
- cleanup(),此方法被MapReduce框架仅且执行一次，在执行完毕Map任务后，进行相关变量或资源的释放工作。若是将释放资源工作放入方法map()中，也会导致Mapper任务在解析、处理每一行文本后释放资源，而且在下一行文本解析前还要重复初始化，导致反复重复，程序运行效率不高！

实验环境

ubuntu 22.10

hadoop 3.1.3

jdk 1.8

建议课时

4课时

实验步骤

一、环境准备

本实验在idea进行开发。

首先启动Hadoop环境：

```
start-all.sh
```

当看下以下进程，则Hadoop启动成功

```
jps
```

```
ubuntu@079b2e0d54d1:~$ jps
978 ResourceManager
1076 NodeManager
1370 Jps
827 SecondaryNameNode
667 DataNode
543 NameNode
```

二、数据准备

打开命令行窗口，创建文件word.txt

```
cd ~
vi word.txt
```

添加内容

```
love you do
you like me
me like you do
love you do
you like me
me like you do
love you do
you like me
me like you do
love you do
you like me
```

上传至HDFS/{学号}/shiyang6/input, 此实验{学号}为001

```
hdfs dfs -mkdir -p /001/shiyang6/input
hdfs dfs -put ~/word.txt /001/shiyang6/input
```

三、代码编写

1、创建WordMapper类并继承 Mapper类，步骤请参考前面的实验，然后重写map()

完整代码如下：

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
```

```

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

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

        //根据空格切割数据
        String[] split = value.toString().split(" ");

        //循环遍历字符串数据，将数据以 (word,1)形式输出至reduce端
        for (String word : split) {
            context.write(new Text(word), new IntWritable(1));
        }
    }
}

```

2、创建WordReducer类并继承Reducer类，步骤请参考前面的实验，然后重写reduce()和cleanup()

reduce计算的数据先放入全局变量map中，所有的reduce执行完毕后，在cleanup中再次处理全局变量map，进行排序并取得top3

```

@Override
protected void cleanup(Reducer<Text, IntWritable, Text, IntWritable>.Context context)
    throws IOException, InterruptedException {

    //将Map中的所有元素转换为列表
    List<Map.Entry<String, Integer>> list = new LinkedList<Map.Entry<String, Integer>>(map.entrySet());

    //使用Collections.sort方法对列表里的数据进行降序排序。|
    //Comparator为对比函数
    Collections.sort(list, new Comparator<Map.Entry<String, Integer>>() {
        public int compare(Entry<String, Integer> arg0, Entry<String, Integer> arg1) {
            return (int) (arg1.getValue() - arg0.getValue());
        }
    });

    //循环遍历排序后的列表元素，获取前三个输出到文件
    for (int i = 0; i < 3; i++) {
        context.write(new Text(list.get(i).getKey()), new IntWritable(list.get(i).getValue()));
    }
}

```

完整代码如下：

```

import java.io.IOException;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

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

```

```

public class wordReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    Map<String, Integer> map = new HashMap<String, Integer>();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> iter, Context context)
        throws IOException, InterruptedException {
        int count = 0;

        //统计每个单词出现的次数
        for (IntWritable wordCount : iter) {
            count += wordCount.get();
        }
        String name = key.toString();

        //将统计好的数据保存到map集合中
        map.put(name, count);
    }

    //reduce计算的数据先放入全局变量map中，所有的reduce执行完毕后，在cleanup中再次处理全局
    变量map，进行排序并取得top3
    @Override
    protected void cleanup(Reducer<Text, IntWritable, Text, IntWritable>.Context
        context)
        throws IOException, InterruptedException {

        //将Map中的所有元素转换为列表
        List<Map.Entry<String, Integer>> list = new LinkedList<Map.Entry<String,
        Integer>>(map.entrySet());

        //使用Collections.sort方法对列表里的数据进行降序排序。
        //Comparator为对比函数
        Collections.sort(list, new Comparator<Map.Entry<String, Integer>>() {
            public int compare(Entry<String, Integer> arg0, Entry<String,
        Integer> arg1) {
                return (int) (arg1.getValue() - arg0.getValue());
            }
        });

        //循环遍历排序后的列表元素，获取前三个输出到文件
        for (int i = 0; i < 3; i++) {
            context.write(new Text(list.get(i).getKey()), new
        IntWritable(list.get(i).getValue()));
        }
    }
}

```

3、创建WordMain类并编写main(), 步骤请参考前面的实验

完整代码如下:

```

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

```

```

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class wordMain {
    public static void main(String[] args) throws Exception{
        // 配置hadoop运行环境
        Configuration conf = new Configuration();

        //创建Job实例与MapReduce驱动类
        Job job = Job.getInstance(conf);
        job.setJarByClass(wordMain.class);

        //设置Mapper运行类与输出格式
        job.setMapperClass(wordMapper.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        //设置Reducer运行类与输出格式
        job.setReducerClass(wordReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        //设置读取、输出数据的路径
        FileInputFormat.setInputPaths(job, new
Path("hdfs://localhost:9000/001/shiyan6/input/word.txt"));
        FileOutputFormat.setOutputPath(job, new
Path("hdfs://localhost:9000/001/shiyan6/output"));

        job.waitForCompletion(true);
    }
}

```

4、执行程序并查看结果

执行程序

```

3 import org.apache.hadoop.io.IntWritable;
4 import org.apache.hadoop.io.Text;
5 import org.apache.hadoop.mapreduce.Job;
6 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
7 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
8
9 1个用法
10 public class WordMain {
11     public static void main(String[] args) throws Exception{
12         // 配置hadoop运行环境
13         Configuration conf = new Configuration();
14
15         // 创建Job实例与MapReduce驱动类
16         Job job = Job.getInstance(conf);
17         job.setJarByClass(WordMain.class);
18
19         // 设置Mapper运行类与输出格式
20         job.setMapperClass(WordMapper.class);
21         job.setMapOutputKeyClass(Text.class);
22         job.setMapOutputValueClass(IntWritable.class);
23
24         // 设置Reducer运行类与输出格式
25         job.setReducerClass(WordReducer.class);
26         job.setOutputKeyClass(Text.class);
27         job.setOutputValueClass(IntWritable.class);

```

使用hadoop shell查看结果

```

hdfs dfs -ls /001/shiyan6/output
hdfs dfs -cat /001/shiyan6/output/part-r-00000

```

```

ubuntu@b14e25701497:~$ hdfs dfs -ls /output
Found 2 items
-rw-r--r--  3 ubuntu supergroup    0 2022-02-19 07:11 /output/_SUCCESS
-rw-r--r--  3 ubuntu supergroup   19 2022-02-19 07:11 /output/part-r-00000
ubuntu@b14e25701497:~$ hdfs dfs -cat /output/part-r-00000
you      11
like     7
me       7

```

实验总结

该实验是需要先通过mapreduce计算各单词次数，然后将结果排序取得前三，我们都知道reduce是每组执行一次，所以我们将每组计算的结果放入全局变量中，然后通过cleanup()将全局变量中的所有结果值进行排序并取得前三。