MapReduce 编程入门

侯在钱

目 录

在开发 MapReduce 之前需阅读文档《MapReduce 工作原理》，了解 MapReduce 的执行过程。在 MapReduce 过程中，我们需要编程开发的工作是 Map 程序（第 2 个阶段）和 Reduce 程序（第 4 个阶段）。请参考上一节中的 MapReduce 工作流程图。

本文通过实例介绍使用 MapReduce。本实例的功能是实现对文章中单词出现的个数进行统计。

1. MapReduce 开发过程

（1） Eclipse 中导入开发 MapReduce 依赖的 jar 包。

（2） 编写 Map、Reduect，及提交 MapReduce 任务的程序。

（3） 把程序打成 jar 包。

（4） 把打包的程序上传到 Hadoop 服务器执行。

（5） 查询执行结果，验证结果是否正确。

2. 导入需要的 Jar 包

开发 MapReduce 程序需要如下这几个 jar 包，这些包都可以到 Hadoop 的安装包中查找到。需要的 JAR 包如下：

JAR 文件	位置
commons-cli-1.2.jar	/hadoop-2.6.0/share/hadoop/common/lib/
hadoop-common-2.6.0.jar	/hadoop-2.6.0/share/hadoop/common/
hadoop-mapreduce-client-core-2.6.0.jar	/hadoop-2.6.0/share/hadoop/mapreduce/

需要把这些包加入到 Eclipse 的 Java Build Path 中，才可以编译 MapReduce 程序。

3. 示例程序

要实现 MapReduce 功能，需要编写 3 个程序，一个是 Map 程序，一个是 Reduce 程序，第 3 个是 MapReduce 任务提交程序。

下面的程序实现对文本中单词个数的统计功能。

（1） Map 程序

Map 实现把输入的文本数据，分割成一个一个的单词，使用单词作为 Map 的 Key，使用单词的个数作为 Map 的 Value。

```
package com.hadoop.mp;

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 WordCountMapper extends  Mapper<Object, Text, Text, IntWritable> {

   public void map(LongWritable key, Text value,  Context  context)  throws
IOException,InterruptedException {
       //把value值用空格进行分割
       String s = value.toString();
       String[] words = s.split(" ");
```

```
        //把单词名作为输出Map的Key，输出Map的Value直接写上数字1。
        if (words!=null&&words.length>0) {
            for (int i=0; i<words.length; i++) {
                Text outputKey = new Text(words[i]);
                IntWritable outputValue = new IntWritable(1);
                //把结果写到写到输出Map中
                context.write(outputKey, outputValue);
            }
        }
    }
}
```

（2） Reduce 程序

输入从 Map 阶段得到的结果，通过 Reduce 转换成另一种 Map 数据输出。

```
package com.hadoop.mp;

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

public class WordCountReducer  extends Reducer<Text, IntWritable, Text,
IntWritable> {
     public  void  reduce(Text  key,  Iterable<IntWritable>  values,  Context
context)  throws IOException,InterruptedException {

        //把value值进行累加求和
        int result = 0;
        for (IntWritable val : values) {
            result = result + val.get();
        }

        //输入Map的Key作为输出Map的Key
        //value累加求和后的结果作为输出Map的Value
        context.write(key,new IntWritable(result));

     }

}
```

（3） 提交任务程序

在提交任务时，需要指定几个参数：

（1） 设置任务名称。

（2） 设置使用哪个 Map 程序。

（3） 设置使用哪个 Reduce 程序。

（4） 设置输入的数据存放的目录，及数据输入格式。

（5） 设置输出的数据存放的目录。

（6） 设置输出的 Map 的 Key 和 Value 的数据类型。

```
package com.hadoop.mp;


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.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCountJobSubmit {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();
        //新建Job对象
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCountJobSummit.class);
        //指定Mapper程序
        job.setMapperClass(WordCountMapper.class);
        //指定Combiner程序（和Reduce程序一样）
        job.setCombinerClass(WordCountReducer.class);
        //指定Reducer程序
        job.setReducerClass(WordCountReducer.class);
        //指定输出键值类型
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        //设置输入输出路径
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
        //多子job的类中，可以保证各个子job串行执行
        System.exit(job.waitForCompletion(true) ? 0 : 1);

    }
```

```
}
```

通过 conf.setInputFormat(TextInputFormat.class)指定 TextInputFormat 为数据的数据格式，此格式一次处理一行。

4. 打成 Jar 包

jar cvf MapReduceTest.jar com

5. 上传 Jar 包

上传 **MapReduceTest.jar** 到 **Linux** 的任意目录下。

6. 启动 Hadoop

在**/$HADOOP_HOME/sbin/**目录下执行：
./start-dfs.sh
./start_yarn.sh
如果已经启动，则不需要再启动。

7. 在 HDFS 中创建目录和测试文件

创建 **MapReduce** 程序的数据数据目录，命令如下：
hdfs dfs -mkdir /user/input

从本地目录**(/root/tmp/a.txt)**传送文件到 **HDFS** 的**/user/input/**目录下：
hdfs dfs -put words.txt /user/input

MapReduce 的输出目录不需要创建，**MapReduce** 系统会自动创建。

8. 运行程序

在 **MapReduceTest.jar** 文件所在目录下执行如下命令：

hadoop jar MapReduceTest.jar com.hadoop.mp.WordCountJobSubmit /user/input /user/output

备注：**/user/output/**目录不需要手工创建，由系统自动创建。

9. 查看运行结果

使用如下命令查看**/user/output/**目录下生成的新文件：

hdfs dfs -ls /user/output

查看**/user/output/part-00000** 文件：

hdfs dfs -cat /user/output/part-r-00000