实验六　Spark 编程

6.1 实验目的

- 学习编写简单的基于 RDD API 的 Spark 程序
- 掌握在 IDEA 中调试 Spark 相关程序，以及在单机伪分布式、分布式部署方式下运行 Spark 相关程序的方法

6.2 实验任务

- 完成 WordCount 示例程序的编写
- 在单机伪分布式和分布式部署方式下运行 WordCount 示例程序

6.3 实验环境

- 操作系统：Ubuntu 18.04
- JDK 版本：jdk 1.8
- Hadoop 版本：2.10.1
- Spark 版本: 2.4.7
- Scala 版本：2.11.12
- IDEA 版本：2020.2.3 （Ultimate 版）

6.4 实验步骤

6.4.1 编写 Spark 应用程序

(1) 新建 Maven 项目并添加依赖

- 新建名为“SparkDemo”的 Maven 项目
- 编辑项目根目录下的 pom.xml 文件，分别在 <dependencies> 和 <build> 标签中添加 Spark 相关的插件 maven-compiler-plugin 和依赖包 spark-core

```
<dependencies>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_2.11</artifactId>
        <version>2.4.7</version>
    </dependency>
```

```
</dependencies>
```

修改完成后，在菜单界面选择 View->Tool Windows->Maven，在弹出的界面中点击 Reload All Maven Projects 加载依赖文件，第一次加载此过程可能耗时较长。

(2) 编写代码

• Java 版: 新建 Java 类 *WordCount*

在项目的 *src/main/java* 目录下，选择 New -> Package，输入名称 *cn.edu.ecnu.spark.example.java.wordcount*，之后右键单击新建好的包，选择 New -> Java Class，输入名称 *WordCount*。

```
package cn.edu.ecnu.spark.example.java.wordcount;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.*;
import scala.Tuple2;

import java.util.Arrays;
import java.util.Iterator;

public class WordCount {

  public static void run(String[] args) {
    /* 步骤1: 通过SparkConf设置配置信息，并创建SparkContext */
    SparkConf conf = new SparkConf();
    conf.setAppName("WordCountJava");
    conf.setMaster("local"); // 仅用于本地进行调试，如在集群中运行则删除本行
    JavaSparkContext sc = new JavaSparkContext(conf);

    /* 步骤2: 按应用逻辑使用操作算子编写DAG，其中包括RDD的创建、转换和行动等 */
    // 读入文本数据，创建名为lines的RDD
    JavaRDD<String> lines = sc.textFile(args[0]);

    // 将lines中的每一个文本行按空格分割成单个单词
    JavaRDD<String> words =
        lines.flatMap(
            new FlatMapFunction<String, String>() {
              @Override
              public Iterator<String> call(String line) throws Exception {
```

```
            return Arrays.asList(line.split(" ")).iterator();
          }
        });
    // 将每个单词的频数设置为1，即将每个单词映射为[单词，1]
    JavaPairRDD<String, Integer> pairs =
        words.mapToPair(
            new PairFunction<String, String, Integer>() {
              @Override
              public Tuple2<String, Integer> call(String word) throws Exception
                {
                return new Tuple2<String, Integer>(word, 1);
              }
            });
    // 按单词聚合，并对相同单词的频数使用sum进行累计
    JavaPairRDD<String, Integer> wordCounts =
        pairs
            .groupByKey()
            .mapToPair(
                new PairFunction<Tuple2<String, Iterable<Integer>>, String,
                    Integer>() {
                  @Override
                  public Tuple2<String, Integer> call(Tuple2<String,
                      Iterable<Integer>> t)
                      throws Exception {
                    Integer sum = Integer.valueOf(0);
                    for (Integer i : t._2) {
                      sum += i;
                    }
                    return new Tuple2<String, Integer>(t._1, sum);
                  }
                });
    // 合并机制
    /*JavaPairRDD<String, Integer> wordCounts =
    pairs.reduceByKey(
        new Function2<Integer, Integer, Integer>() {
          @Override
          public Integer call(Integer t1, Integer t2) throws Exception {
            return t1 + t2;
          }
        });*/

    // 输出词频统计结果到文件
```

```
        wordCounts.saveAsTextFile(args[1]);
    /* 步骤3：关闭SparkContext */
    sc.stop();
  }

  public static void main(String[] args) {
    run(args);
  }
}
```

• Scala 版：新建 Scala 类 *WordCountScala*

在项目的 *src/main* 目录下，选择 New -> Directory，输入 scala，接着右键单击目录 scala，选择 Mark Directory as -> Sources Root。在 scala 目录下选择 New->Package，输入名称 *cn.edu.ecnu.spark. example.scala.wordcount*，之后右键单击新建好的包，选择 New -> Scala Class，输入名称 *WordCount*。

```
package cn.edu.ecnu.spark.example.scala.wordcount

import org.apache.spark.{SparkConf, SparkContext}

object WordCount {

  def run(args: Array[String]): Unit = {
    /* 步骤1：通过SparkConf设置配置信息，并创建SparkContext */
    val conf = new SparkConf
    conf.setAppName("WordCount")
    conf.setMaster("local") // 仅用于本地进行调试，如在集群中运行则删除本行
    val sc = new SparkContext(conf)

    /* 步骤2：按应用逻辑使用操作算子编写DAG，其中包括RDD的创建、转换和行动等 */
    // 读入文本数据，创建名为lines的RDD
    val lines = sc.textFile(args(0))
    // 将lines中的每一个文本行按空格分割成单个单词
    val words = lines.flatMap { line => line.split(" ") }
    // 将每个单词的频数设置为1，即将每个单词映射为[单词，1]
    val pairs = words.map { word => (word, 1) }

    // 按单词聚合，并对相同单词的频数使用sum进行累计
    val wordCounts = pairs.groupByKey().map(t => (t._1, t._2.sum))
    // 如需使用合并机制则将第上一行替换为下行
    // val wordCounts = pairs.reduceByKey(_+_)

```

```
    // 输出词频统计结果到文件
    wordCounts.saveAsTextFile(args(1))

    /* 步骤3：关闭SparkContext */
    sc.stop()
  }

  def main(args: Array[String]): Unit = {
    run(args)
  }
}
```

6.4.2 调试 Spark 应用程序

使用 IDEA 调试“WordCount”应用程序。

(1) 准备数据

```
cd ~
mkdir spark_input
cp ~/Chinese-Cloze-RC-master/people_daily/pd/pd.test ~/spark_input/pd.test
# 之前实验的数据集，如果删除了可以通过wget
    https://github.com/ymcui/Chinese-Cloze-RC/archive/master.zip下载
```

(2) 配置运行环境

点击菜单栏 Run -> Edit Configuration, 在弹出的界面中点击 + 号选择 Application，新建 Application 配置，Name 为 WordCountJava（Scala 版则为 WordCountScala），配置界面如图6.1所示。

- 配置 Main Class 为 *cn.edu.ecnu.spark.example.java.wordcount.WordCount*
  (Scala 版则配置为 *cn.edu.ecnu.spark.example.scala.wordcount.WordCount*)
- 配置 Program arguments 为 *输入路径 输出路径*
  如 */home/dase-local/spark_input /home/dase-local/IdeaProjects/SparkDemo/spark_output/*，其中前者表示输入路径，后者表示输出路径。

(3) 运行应用程序

在菜单栏点击 Run->Run ‘WordCountJava’ (Scala 版为 Run ‘WordCountScala’)，运行结果如图6.2所示，程序运行结束在 IDEA 中产生的文件如图6.3所示。

注意：运行前删除目录*/home/dase-local/IdeaProjects/SparkDemo/spark_output/*。

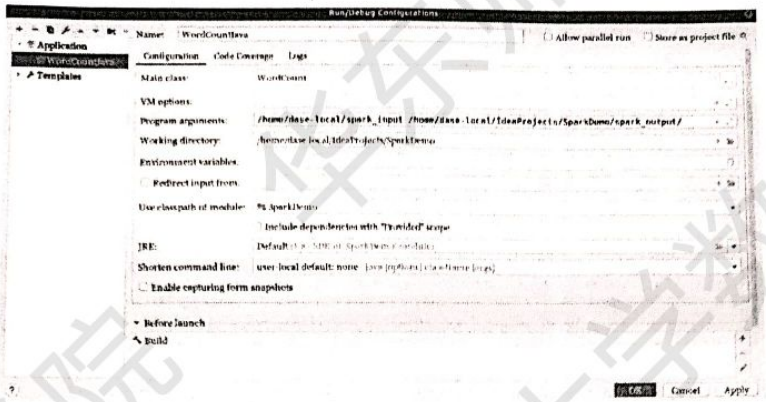

图 6.1 运行应用程序配置界面

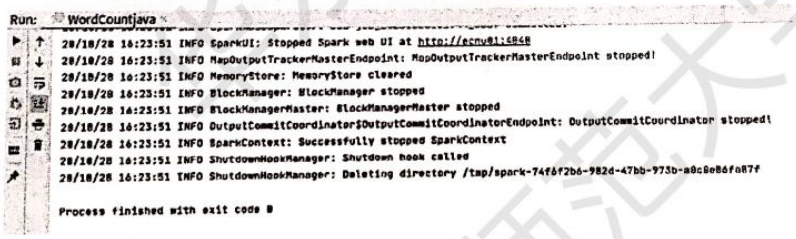

图 6.2 在 IDEA 中的运行结果

图 6.3 程序运行产生的文件

6.4.3 运行 Spark 应用程序

以 Java 版本为例，展示如何运行 Spark 应用程序。

(1) 准备工作

以下操作在各节点均以 dase-local 用户身份进行。

- 使用 IDEA 将程序打成 jar 包

  注意：打包前注释掉代码中的 *conf.setMastr("local");*)。

  jar 包名称为 RddWordCountJava.jar，打包路径为*/home/dase-local/IdeaProjects/SparkDemo/out/artifacts/RddWordCountJava/*，配置界面如图6.4所示。

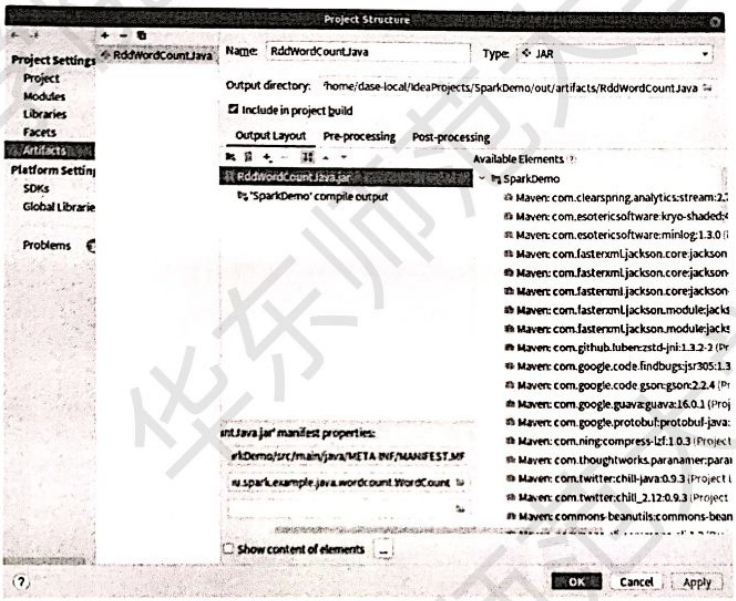

图 6.4 jar 包配置界面

- 复制 jar 包到指定目录

```
mkdir  ~/spark-2.4.7/myApp
cp ~/IdeaProjects/SparkDemo/out/artifacts/RddWordCountJava/RddWordCountJava.j
    ar ~/spark-2.4.7/myApp/
```

(2) 单机伪分布式部署方式下运行应用程序

以下操作在各节点均以 dase-local 用户身份进行。

- 启动服务

```
su dase-local
cd ~
~/hadoop-2.10.1/sbin/start-dfs.sh
~/spark-2.4.7/sbin/start-all.sh
~/spark-2.4.7/sbin/start-history-server.sh
```

• 准备数据

```
~/hadoop-2.10.1/bin/hdfs dfs -cp ./input/pd.test ./spark_input
#该命令将hdfs://user/dase-local/input/pd.test拷贝到hdfs://user/dase-local/spark_input中
#其中，pd.test数据文件之前已经上传至hdfs://user/dase-local/input
#spark_input目录之前也已创建完成
```

• 通过提交 jar 包运行应用程序

```
~/hadoop-2.10.1/bin/hdfs dfs -rm -r spark_output #删除 HDFS 上的输出路径
~/spark-2.4.7/bin/spark-submit \
 --master spark://localhost:7077 \
 --class cn.edu.ecnu.spark.example.java.wordcount.WordCount \
 /home/dase-local/spark-2.4.7/myApp/RddWordCountJava.jar
     hdfs://localhost:9000/user/dase-local/spark_input
     hdfs://localhost:9000/user/dase-local/spark_output
     #将之前实验中上传到HDFS上的文件作为输入
```

```
21/02/28 15:00:17 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHandler@62fe6067{/jobs/job,null,UNAVAILABLE,@Spark}
21/02/28 15:00:17 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHandler@3e34ace1{/jobs/json,null,UNAVAILABLE,@Spark}
21/02/28 15:00:17 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHandler@6e1d8f9e{/jobs,null,UNAVAILABLE,@Spark}
21/02/28 15:00:17 INFO ui.SparkUI: Stopped Spark web UI at http://219.228.148.154:4040
21/02/28 15:00:17 INFO cluster.StandaloneSchedulerBackend: Shutting down all executors
21/02/28 15:00:18 INFO cluster.CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
21/02/28 15:00:18 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
21/02/28 15:00:18 INFO memory.MemoryStore: MemoryStore cleared
21/02/28 15:00:18 INFO storage.BlockManager: BlockManager stopped
21/02/28 15:00:18 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
21/02/28 15:00:18 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
21/02/28 15:00:18 INFO spark.SparkContext: Successfully stopped SparkContext
21/02/28 15:00:18 INFO util.ShutdownHookManager: Shutdown hook called
21/02/28 15:00:18 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-ad27bbda-9833-421c-8b34-52e2f6e4e1c2
```

图 6.5 在伪分布式部署方式下应用程序的运行结果

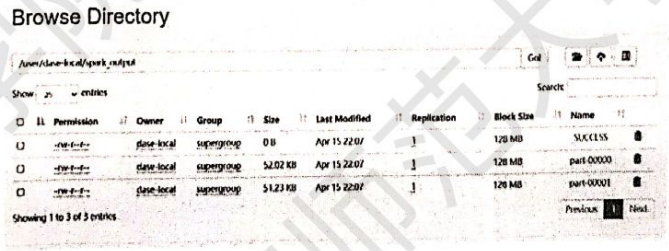

图 6.6 程序运行产生的文件

执行结果如图6.5所示，程序运行产生的文件如图6.6所示。

• 停止服务

```
~/hadoop-2.10.1/sbin/stop-dfs.sh
~/spark-2.4.7/sbin/stop-all.sh
~/spark-2.4.7/sbin/stop-history-server.sh
```

(3) 分布式部署方式下运行应用程序

以下操作在各节点均以 dase-dis 用户身份进行。

• 复制 jar 包到指定目录

在客户端节点 (ecnu04) 执行以下命令，将之前打好的包拷贝至 dase-dis 用户下的 ~/spark-2.4.7/myApp 路径中。

```
su dase-dis
mkdir ~/spark-2.4.7/myApp/ #在hadoop目录下新建myApp/目录
scp dase-local@localhost:/home/dase-local/spark-2.4.7/myApp/
    /home/dase-dis/hadoop-2.10.1/myApp/
```

• 启动服务

在主节点执行以下命令:

```
 su dase-dis
~/hadoop-2.10.1/sbin/start-dfs.sh
~/spark-2.4.7/sbin/start-all.sh
~/spark-2.4.7/sbin/start-history-server.sh
```

• 准备数据

在客户端节点 (ecnu04) 执行以下命令:

```
~/hadoop-2.10.1/bin/hdfs dfs -cp ./input/pd.test ./spark_input
#pd.test数据文件之前已经上传至hdfs:///user/dase-dis/input
#spark_input目录之前也已创建完成
```

• 通过提交 jar 包运行应用程序

在客户端节点 (ecnu04) 执行以下命令:

```
~hadoop-2.10.1/bin/hdfs dfs -rm -r spark_output
~/spark-2.4.7/bin/spark-submit \
 --master spark://ecnu01:7077 \
 --class cn.edu.ecnu.spark.example.java.wordcount.WordCount \
 /home/dase-dis/spark-2.4.7/myApp/RddWordCountJava.jar
     hdfs://ecnu01:9000/user/dase-dis/spark_input
     hdfs://ecnu01:9000/user/dase-dis/spark_output
```

分布式环境下程序运行产生的文件如图6.7所示。

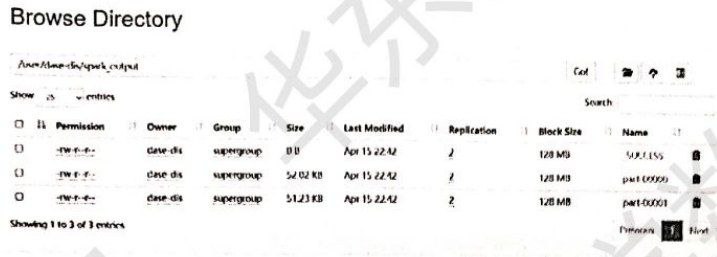

图 6.7 程序运行产生的文件

- 停止服务

在主节点执行以下命令:

```
~/hadoop-2.10.1/sbin/stop-dfs.sh
~/spark-2.4.7/sbin/stop-all.sh
~/spark-2.4.7/sbin/stop-history-server.sh
```

6.5 思考题

1 第6.4.3节中编写的 WordCount 应用程序由多少个 Stage 组成？请结合 Spark 的 Web UI 进行说明。

2 对于第6.4.3节中编写的 WordCount 应用程序，根据实际运行情况，结合 Spark 的 Web UI 说明运行过程中 Shuffle 的数据量有多大。