

实验四 Spark 部署与编程 (单机 + 伪分布式)

4.1 实验目的

- 学习 Spark 的部署，简单使用 Spark-Shell
- 查看 Spark 的运行日志，体会与 MapReduce 运行过程中日志的区别
- 通过系统部署理解体系架构，体会 Spark 与 MapReduce 之间的区别
- 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部署方式下运行 Spark 相关程序的方法

4.2 实验任务

- 完成 Spark 的单机集中式部署和单机伪分布式部署
- 在单机伪分布式部署方式下，分别以 Client 和 Cluster 提交方式来运行示例程序
- 完成 WordCount 示例程序的编写
- 在单机伪分布式部署方式下运行 WordCount 示例程序

4.3 实验环境

- 操作系统 (Spark 部署): Ubuntu 18.04 或 Ubuntu 20.04
- 操作系统 (Spark 编程): 任意本地系统
- JDK 版本: 1.8
- Spark 版本: 2.4.7
- Hadoop 版本: 2.10.1

4.4 实验步骤

注意: 本教程以名为 dase-local 的用户为例开展实验，读者在实验过程中需替换为自己的用户名（云主机上的默认用户名为 ubuntu）。

4.4.1 单机集中式部署

- (1) 准备工作
 - 登录用户 dase-local

- 修改.bashrc 文件

说明：若终端配置的 TERM 环境变量为 xterm-256color，则会导致基于 Scala-2.11 构建的 Spark-2.4.7 中的 Spark-Shell 出现错误。对于该问题，需要将 TERM 环境变量设置为 xterm-color。而基于 Scala-2.12 构建的 Spark-2.4.7 不存在此问题，若采用 Scala-2.12 构建的 Spark-2.4.7，则无需修改 TERM 环境变量的配置。

```
1 vi ~/.bashrc
```

在文件结尾添加如下内容：

```
1 export TERM=xterm-color
```

使.bashrc 配置生效：

```
1 source ~/.bashrc
```

- 下载并安装 Spark（云主机镜像中已经下载好了，在/home/ubuntu 目录下，直接解压即可）

```
1 wget http://archive.apache.org/dist/spark/spark-2.4.7/spark-2.4.7-bin-with-hadoop.tgz #下载安装包，可能较慢  
2 tar -zxvf spark-2.4.7-bin-without-hadoop.tgz  
3 mv ~/spark-2.4.7-bin-without-hadoop ~/spark-2.4.7
```

(2) 修改配置

- 修改 spark-env.sh 文件（文件路径：~/dase-local/spark-2.4.7/conf/）

将 spark-env.sh.template 文件重命名为 spark-env.sh：

```
1 mv ~/spark-2.4.7/conf/spark-env.sh.template ~/spark-2.4.7/conf/spark-env.sh
```

在 spark-env.sh 文件末尾添加如下内容：

```
1 # 因为下载的是 Hadoop Free 版本的 Spark，所以需要配置 Hadoop 的路径  
2 HADOOP_HOME=/home/dase-local/hadoop-2.10.1  
3 export SPARK_DIST_CLASSPATH=$($HADOOP_HOME/bin/hadoop classpath)  
4 export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native:$LD_LIBRARY_PATH
```

(3) 运行 Spark 应用程序

- 通过 Spark-Shell 运行应用程序

– 执行以下命令进入 Spark-Shell，进入 Spark-Shell 后如图4.1所示

```
1 ~/spark-2.4.7/bin/spark-shell --master local
```

```
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel  
(newLevel).  
Spark context Web UI available at http://ecnu01:4040  
Spark context available as 'sc' (master = local, app id = local-1612254635858).  
Spark session available as 'spark'.  
Welcome to  
 version 2.4.7  
  
Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_261)  
Type in expressions to have them evaluated.  
Type :help for more information.  
scala> 
```

图 4.1 Spark-Shell

- 在 `scala>` 后输入 Scala 代码

```
1 sc.textFile("file:///home/dase-local/spark-2.4.7/RELEASE").flatMap(_.split  
(" ")).map((_, 1)).reduceByKey(_ + _).collect
```

此处执行的是统计 `/home/dase-local/spark-2.4.7/RELEASE` 文件中的单词数量，执行后应打印出如图 4.2 所示结果。

```
res0: Array[(String, Int)] = Array((-Psparkr,1), (-B,1), (Spark,1), (-Pkubernetes,1),  
(-Pyarn,1), (revision,1), (Build,1), (-DzincPort=3038,1), (-Pflume,1), ((git,1), (2.6.5,1), (flags:,1), (-Mesos,1), (for,1), (-Pkafka-0-8,1),  
(-Phadoop-provided,1), (1421la1,1), (2.4.7,1), (Hadoop,1))
```

图 4.2 统计结果

- 在 `scala>` 后输入 “:q” 来退出 Spark-Shell
- 通过提交 jar 包运行应用程序

```
1 ~/spark-2.4.7/bin/spark-submit --master local --class org.apache.spark.  
examples.SparkPi ~/spark-2.4.7/examples/jars/spark-examples_2.11-  
2.4.7.jar
```

在运行过程中另起一个终端执行 `jps` 查看进程，此时只会出现 `SparkSubmit` 进程，应用程序运行结束后该进程消失，如图 4.3 所示。

```
dase-local@ecnu01:~$ jps  
9484 Jps  
9374 SparkSubmit  
dase-local@ecnu01:~$ jps  
9528 Jps
```

图 4.3 local 部署方式下运行过程中存在的进程

运行结果如图 4.4 所示，可以看到图片中输出的 `Pi` 的近似值。

```
21/02/02 16:43:13 INFO scheduler.DAGScheduler: ResultStage 0 (reduce at SparkPi.  
scala:38) finished in 0.483 s  
21/02/02 16:43:13 INFO scheduler.DAGScheduler: Job 0 finished: reduce at SparkPi.  
.scala:38, took 0.549308 s  
Pi is roughly 3.1452557262786316  
21/02/02 16:43:13 INFO server.AbstractConnector: Stopped Spark@27f9e982{HTTP/1.1  
,[http/1.1]}{0.0.0.0:4041}  
21/02/02 16:43:13 INFO ui.SparkUI: Stopped Spark web UI at http://ecnu01:4041  
21/02/02 16:43:13 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMas-  
terEndpoint stopped!
```

图 4.4 local 部署方式下运行结果

4.4.2 单机伪分布式部署

(1) 准备工作

- 登录用户 dase-local
- 启动已部署的 HDFS 单机伪分布式服务

```
1 ~/hadoop-2.10.1/sbin/start-dfs.sh
```

(2) 修改配置

依次定位到如下文件所在位置，右键单击文件，选择查看所有应用程序，找到文本编辑器并单击，点击选择后开始编辑文件内容。

- 修改 spark-env.sh 文件（文件路径：~/spark-2.4.7/conf/）

定位到文件位置，在末尾添加以下内容：

```
1 export SPARK_MASTER_HOST=localhost # 主节点主机名  
2 export SPARK_MASTER_PORT=7077 # 端口号
```

- 修改 spark-defaults.conf 文件（文件路径：~/spark-2.4.7/conf/）

```
1 cp ~/spark-2.4.7/conf/spark-defaults.conf.template  
~/spark-2.4.7/conf/spark-defaults.conf # 根据模板新建spark-defaults.conf
```

在 spark-defaults.conf 文件末尾添加以下内容：

```
1 # 通过配置以下参数开启日志聚集功能  
2 spark.eventLog.enabled=true  
3 spark.eventLog.dir=hdfs://localhost:9000/tmp/spark_history  
4 spark.history.fs.logDirectory=hdfs://localhost:9000/tmp/spark_history
```

- 修改 spark-config.sh 文件（文件路径：~/spark-2.4.7/sbin/）

在文件末尾添加以下内容：

```
1 export JAVA_HOME=/usr/local/jdk1.8
```

- 在 HDFS 中建立目录 /tmp/spark_history

```
1 ~/hadoop-2.10.1/bin/hdfs dfs -mkdir -p /tmp/spark_history
```

(3) 启动服务

- 启动命令

```
1 ~/spark-2.4.7/sbin/start-all.sh # 启动 Spark  
2 ~/spark-2.4.7/sbin/start-history-server.sh # 启动应用日志服务器
```

(4) 查看 Spark 服务信息

- 使用 jps 查看进程，验证是否成功启动服务

在单机伪分布式部署方式下，该节点既充当 Master，又充当 Worker，故该节点上会出现 Master 和 Worker 进程。如图4.5所示。

```
21040 DataNode  
18753 Master  
21572 HistoryServer  
21687 Jps  
21319 SecondaryNameNode  
20794 NameNode  
18986 Worker
```

图 4.5 启动 Spark 服务后存在的进程

- 查看 Master、Worker、HistoryServer 进程日志

本实验的进程日志记录在 `~/spark-2.4.7/logs/` 路径下，后缀为.log 的文件中。

- Master 进程日志: `spark-*org.apache.spark.deploy.master.Master-1-*.`*log*
- Worker 进程日志: `spark-*org.apache.spark.deploy.worker.Worker-1-*.`*log*
- HistoryServer 进程日志: `spark-*org.apache.spark.deploy.history.HistoryServer-1-*.`*log*

- 访问 Spark Web 界面

通过 `http://localhost:8080`，可看到 Master 和 Worker，如图4.6所示。

(5) 运行 Spark 应用程序

- 通过 Spark-Shell 运行应用程序

- 准备输入文件

```
1 ~/hadoop-2.10.1/bin/hdfs dfs -mkdir -p spark_input  
2 ~/hadoop-2.10.1/bin/hdfs dfs -put ~/spark-2.4.7/RELEASE spark_input/
```

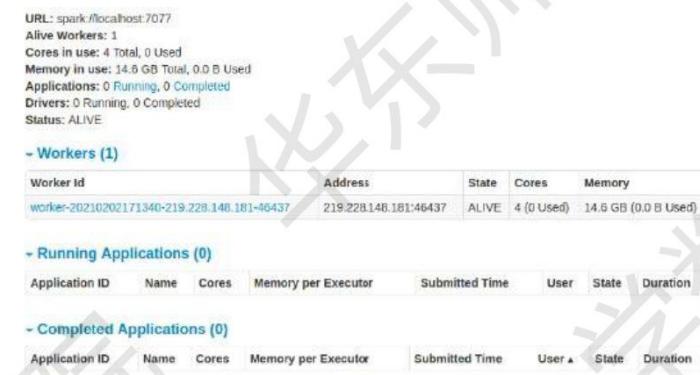


图 4.6 Spark Master 界面

– 进入 Spark-Shell

(注: 如使用 localhost 无法正常启动, 可尝试将 localhost 改为 127.0.0.1)

```
1 ~/spark-2.4.7/bin/spark-shell --master spark://localhost:7077
```

– 在 *scala>* 后输入 Scala 代码

```
1 sc.textFile("spark_input/RELEASE").flatMap(_.split(" ")).map((_,  
1)).reduceByKey(_ + _).collect
```

此处执行的是统计 RELEASE 文件中的单词数量, 执行后应打印出如图4.7所示结果。

```
res0: Array[(String, Int)] = Array((-Psparkr,1), (Build,1), (built,1), (-Pflume,  
1), ((git,1), (-Pmesos,1), (-Phadoop-provided,1), (1421a1),1), (-B,1), (Spark,1)  
) , (-Pkubernetes,1), (-Pyarn,1), (revision,1), (-DzincPort=3038,1), (2.6.5,1),  
(flags:,1), (for,1), (-Pkafka-0-8,1), (2.4.7,1), (Hadoop,1))
```

图 4.7 示例程序运行结果

– 在 *scala>* 后输入 “:q” 来退出 Spark-Shell

• 通过提交 jar 包运行应用程序

(注: 如使用 localhost 无法正常启动, 可尝试将 localhost 改为 127.0.0.1。)

– Client 提交方式 (默认)

该提交方式下 Driver 运行在客户端, 可以在客户端看到应用程序运行过程中的信息。

```
1 ~/spark-2.4.7/bin/spark-submit --deploy-mode client --master  
spark://localhost:7077 --class org.apache.spark.examples.SparkPi  
~/spark-2.4.7/examples/jars/spark-examples_2.11-2.4.7.jar
```

在运行过程中另起一个终端执行 jps 查看进程，如图4.8所示。此时会存在一个 CoarseGrainedExecutorBackend 进程，负责创建及维护 Executor 对象。

```
21040 DataNode
12420 SparkSubmit
21319 SecondaryNameNode
8808 HistoryServer
12537 CoarseGrainedExecutorBackend
8410 Master
20794 NameNode
8604 Worker
12556 Jps
```

图 4.8 Client 提交方式运行过程中存在的进程

运行结果如图4.9所示，可以看到图片中输出的 Pi 的近似值。

```
21/02/02 18:34:27 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
Pi is roughly 3.141475707378537
21/02/02 18:34:27 INFO server.AbstractConnector: Stopped Spark@3c291aad{HTTP/1.1
,[http/1.1]}{0.0.0.0:4040}
21/02/02 18:34:27 INFO ui.SparkUI: Stopped Spark web UI at http://ecnu01:4040
```

图 4.9 Client 提交方式下示例程序运行结果

– Cluster 提交方式

该提交方式下 Master 会随机选取一个 Worker 节点启动 Driver，故在客户端看不到应用程序运行过程中的信息。

```
1 ~/spark-2.4.7/bin/spark-submit --deploy-mode cluster --master
  spark://localhost:7077 --class org.apache.spark.examples.SparkPi
  ~/spark-2.4.7/examples/jars/spark-examples_2.11-2.4.7.jar
```

在运行过程中另起一个终端执行 jps 查看进程，如图4.10所示。在 Cluster 提交方式下，还可以看到一个 DriverWrapper 进程。

(6) 查看 Spark 程序运行信息

- 实时查看应用运行情况

在应用运行过程中（如进入 Spark-Shell 之后），访问 <http://localhost:4040>。

Web 界面如图4.11所示。

- 查看 Spark 应用程序日志

在提交一个应用程序后，在 `~/spark-2.4.7/work` 下会出现应用程序运行日志文件夹。

访问 <http://localhost:8080>，点击 Completed Applications 下的对应应用程序

```
21040 DataNode
15525 SparkSubmit
15622 DriverWrapper
21319 SecondaryNameNode
15767 Jps
15704 CoarseGrainedExecutorBackend
8808 HistoryServer
8410 Master
20794 NameNode
8604 Worker
```

图 4.10 Cluster 提交方式运行过程中存在的进程

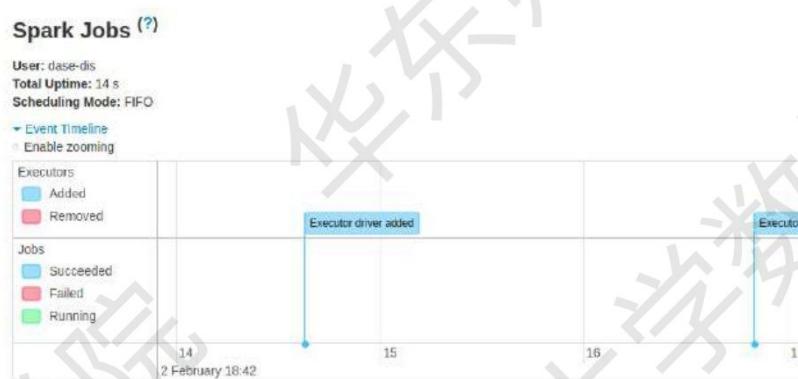


图 4.11 Spark Jobs 界面

的 Application ID，进入到如图4.12所示界面，点击 stdout 或者 stderr 可以查看对应日志信息。

The screenshot shows the Spark Application UI. At the top, it displays application details: ID: app-20210202183853-0002, Name: Spark Pi, User: dase-dis, Cores: Unlimited (3 granted), Executor Limit: Unlimited (1 granted), Executor Memory: 1024.0 MB, Submit Date: 2021/02/02 18:38:53, State: FINISHED.

Below this is the 'Executor Summary (1)' section, which lists one executor: worker_20210202182907-219.228.148.181-40909 with 3 cores, 1024 memory, and a KILLED state. A link to 'stdout.stderr' is provided.

Underneath is the 'Removed Executors (1)' section, which also lists the same executor with the same details.

图 4.12 Spark 某个 Application 的界面

- 查看应用历史记录

在应用运行结束后，访问 <http://localhost:18080>。Web 界面如图4.13所示。

The screenshot shows the Spark History UI. It displays a table of completed applications:

App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
app-20210202183853-0002	Spark Pi	2021-02-02 18:30:53	2021-02-02 18:38:56	3 s	dase-dis	2021-02-02 18:39:56	Download
app-20210202183424-0001	Spark Pi	2021-02-02 18:34:24	2021-02-02 18:34:27	3 s	dase-dis	2021-02-02 18:34:27	Download
app-20210202182931-0000	Spark shell	2021-02-02 18:29:30	2021-02-02 18:33:07	3.6 min	dase-dis	2021-02-02 18:33:07	Download

At the bottom, there are links for 'Showing 1 to 3 of 3 entries' and 'Show incomplete applications'.

图 4.13 Spark History 界面

4.4.3 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，并配置 ScalaSDK（可选）
- 编辑项目根目录下的 pom.xml 文件，在 <dependencies> 标签中添加 Spark 相关的依赖包 spark-core

```

1 <dependencies>
2   <dependency>
3     <groupId>org.apache.spark</groupId>
4     <artifactId>spark-core_2.11</artifactId>
5     <version>2.4.7</version>
6   </dependency>
7 </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*。

```

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

2
3 import org.apache.spark.SparkConf;
4 import org.apache.spark.api.java.JavaPairRDD;
5 import org.apache.spark.api.java.JavaRDD;
6 import org.apache.spark.api.java.JavaSparkContext;
7 import org.apache.spark.api.java.function.*;
8 import scala.Tuple2;
9
10 import java.util.Arrays;
11 import java.util.Iterator;
12
13 public class WordCount {
14
15     public static void run(String[] args) {
16         /* 步骤1：通过SparkConf设置配置信息，并创建SparkContext */
17         SparkConf conf = new SparkConf();
18         conf.setAppName("WordCountJava");
19         conf.setMaster("local"); // 仅用于本地进行调试，如在集群中运行则删除本行
20         JavaSparkContext sc = new JavaSparkContext(conf);
21
22         /* 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 */
23         // 读入文本数据，创建名为lines的RDD
24         JavaRDD<String> lines = sc.textFile(args[0]);
25
26         // 将lines中的每一个文本行按空格分割成单个单词
27         JavaRDD<String> words =
28             lines.flatMap(
29                 new FlatMapFunction<String, String>() {
30                     @Override
31                     public Iterator<String> call(String line) throws Exception {
32                         return Arrays.asList(line.split(" ")).iterator();
33                     }
34                 });
35
36         // 将每个单词的频数设置为1，即将每个单词映射为[单词，1]
37         JavaPairRDD<String, Integer> pairs =
38             words.mapToPair(

```

```
38     new PairFunction<String, String, Integer>() {
39         @Override
40         public Tuple2<String, Integer> call(String word) throws Exception
41             {
42                 return new Tuple2<String, Integer>(word, 1);
43             }
44         });
45     // 按单词聚合，并对相同单词的频数使用sum进行累计
46     JavaPairRDD<String, Integer> wordCounts =
47         pairs
48             .groupByKey()
49             .mapToPair(
50                 new PairFunction<Tuple2<String, Iterable<Integer>>, String,
51                     Integer>() {
52                     @Override
53                     public Tuple2<String, Integer> call(Tuple2<String,
54                         Iterable<Integer> t)
55                         throws Exception {
56                         Integer sum = Integer.valueOf(0);
57                         for (Integer i : t._2) {
58                             sum += i;
59                         }
60                         return new Tuple2<String, Integer>(t._1, sum);
61                     }
62                 });
63     // 合并机制
64     /*JavaPairRDD<String, Integer> wordCounts =
65     pairs.reduceByKey(
66         new Function2<Integer, Integer, Integer>() {
67             @Override
68             public Integer call(Integer t1, Integer t2) throws Exception {
69                 return t1 + t2;
70             }
71         });*/
72     // 输出词频统计结果到文件
73     wordCounts.saveAsTextFile(args[1]);
74     /* 步骤3：关闭SparkContext */
75     sc.stop();
76 }
```

```
77
78 public static void main(String[] args) {
```

```

77     run(args);
78   }
79 }
```

- Scala 版（选做）：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*。

```

1 package cn.edu.ecnu.spark.example.scala.wordcount
2
3 import org.apache.spark.{SparkConf, SparkContext}
4
5 object WordCount {
6
7   def run(args: Array[String]): Unit = {
8     /* 步骤1：通过SparkConf设置配置信息，并创建SparkContext */
9     val conf = new SparkConf
10    conf.setAppName("WordCount")
11    conf.setMaster("local") // 仅用于本地进行调试，如在集群中运行则删除本行
12    val sc = new SparkContext(conf)
13
14    /* 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 */
15    // 读入文本数据，创建名为lines的RDD
16    val lines = sc.textFile(args(0))
17    // 将lines中的每一个文本行按空格分割成单个单词
18    val words = lines.flatMap { line => line.split(" ") }
19    // 将每个单词的频数设置为1，即将每个单词映射为[单词，1]
20    val pairs = words.map { word => (word, 1) }
21
22    // 按单词聚合，并对相同单词的频数使用sum进行累计
23    val wordCounts = pairs.groupByKey().map(t => (t._1, t._2.sum))
24    // 如需使用合并机制则将第上一行替换为下行
25    // val wordCounts = pairs.reduceByKey(_+_)
26
27    // 输出词频统计结果到文件
28    wordCounts.saveAsTextFile(args(1))
29
30    /* 步骤3：关闭SparkContext */
31    sc.stop()
32 }
```

```

33
34     def main(args: Array[String]): Unit = {
35         run(args)
36     }
37 }
```

4.4.4 调试 Spark 应用程序

使用 IDEA 调试“WordCount”应用程序。

(1) 准备数据

使用之前准备的数据集 pd.train。

(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4.14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。

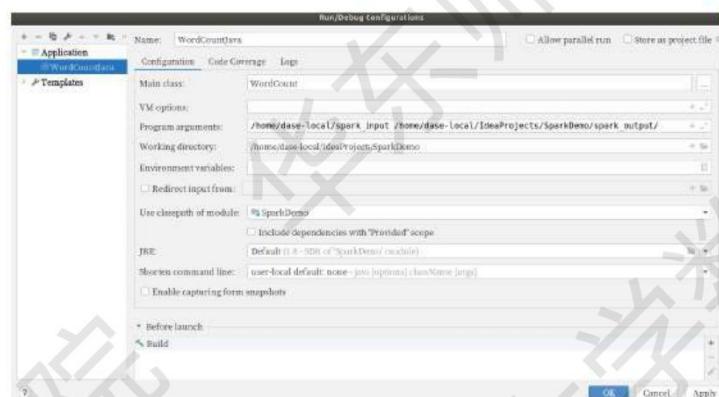


图 4.14 运行应用程序配置界面

(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4.15所示, 程序运行结束在 IDEA 中产生的文件如图4.16所示。

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



```
Run: WordCountJava
28/18/28 16:23:51 INFO SparkUI: Stopped Spark web UI at http://localhost:4040
28/18/28 16:23:51 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
28/18/28 16:23:51 INFO MemoryStore: MemoryStore cleared
28/18/28 16:23:51 INFO BlockManager: BlockManager stopped
28/18/28 16:23:51 INFO BlockManagerMaster: BlockManagerMaster stopped
28/18/28 16:23:51 INFO OutputCommitCoordinator@OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
28/18/28 16:23:51 INFO ShutdownHookManager: Shutdown hook called
28/18/28 16:23:51 INFO ShutdownHookManager: Deleting directory /tmp/spark-74f6f2bb-982d-47bb-973b-a6c8e96fa87f
Process Finished with exit code 0
```

图 4.15 在 IDEA 中的运行结果

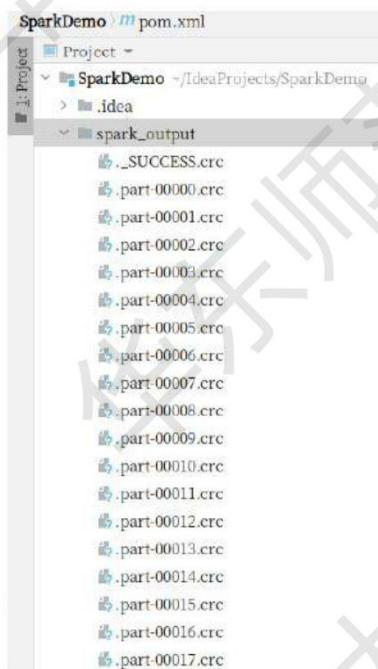


图 4.16 程序运行产生的文件

4.4.5 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.setMaster("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4.17所示。

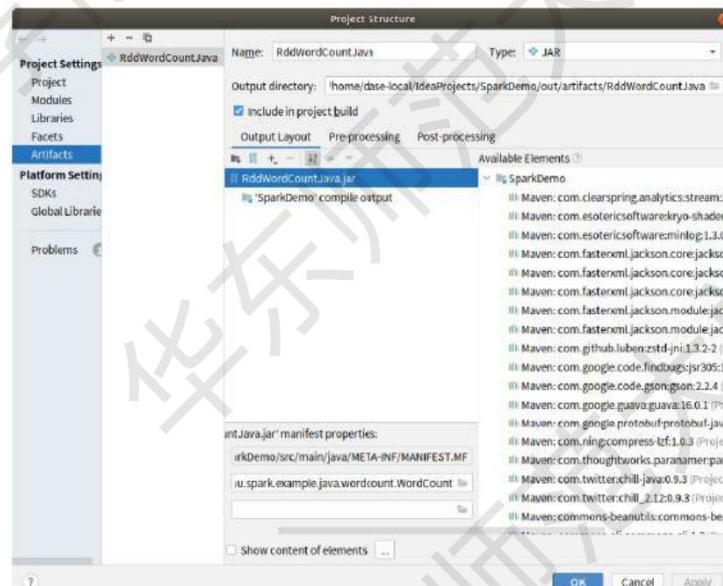


图 4.17 jar 包配置界面

- 复制 jar 包到目录 `/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。

- 准备数据

```

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

```

- 通过提交 jar 包运行应用程序

```

1 ~/hadoop-2.10.1/bin/hdfs dfs -rm -r spark_output #删除 HDFS 上的输出路径
2 ~/spark-2.4.7/bin/spark-submit \

```

```

3   --master spark://localhost:7077 \
4   --class cn.edu.ecnu.spark.example.java.wordcount.WordCount \
5   /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.ServletContextHan
dler@62fe6667{/jobs/job,null,UNAVAILABLE,@Spark}
21/02/28 15:00:17 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHan
dler@3e34ace1{/jobs/json,null,UNAVAILABLE,@Spark}
21/02/28 15:00:17 INFO handler.ContextHandler: Stopped o.s.j.s.ServletContextHan
dler@e1d8f9e{/jobs,null,UNAVAILABLE,@Spark}
21/02/28 15:00:17 INFO ui.SparkUI: Stopped Spark web UI at http://219.228.148.15
4:4040
21/02/28 15:00:17 INFO cluster.StandaloneSchedulerBackend: Shutting down all exe
cutors
21/02/28 15:00:18 INFO cluster.CoarseGrainedSchedulerBackend$DriverEndpoint: Ask
ing each executor to shut down
21/02/28 15:00:18 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMas
terEndpoint stopped!
21/02/28 15:00:18 INFO memory.MemoryStore: MemoryStore cleared
21/02/28 15:00:18 INFO storage.BlockManager: BlockManager stopped
21/02/28 15:00:18 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
21/02/28 15:00:18 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinator
Endpoint: 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-a
d27bba-9833-421c-8b34-52e2f6e4e1c2

```

图 4.18 在伪分布式部署方式下应用程序的运行结果

Browse Directory

/user/dase-local/spark_outputcluster						
Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	dase-local	supergroup	0 B	3	128 MB	_SUCCESS
-rw-r--r--	dase-local	supergroup	132.85 MB	2	128 MB	part-00000
-rw-r--r--	dase-local	supergroup	132.87 MB	2	128 MB	part-00001
-rw-r--r--	dase-local	supergroup	132.53 MB	2	128 MB	part-00002
-rw-r--r--	dase-local	supergroup	132.67 MB	2	128 MB	part-00003
-rw-r--r--	dase-local	supergroup	132.89 MB	2	128 MB	part-00004
-rw-r--r--	dase-local	supergroup	133.54 MB	2	128 MB	part-00005
-rw-r--r--	dase-local	supergroup	132.23 MB	2	128 MB	part-00006
-rw-r--r--	dase-local	supergroup	132.73 MB	2	128 MB	part-00007
-rw-r--r--	dase-local	supergroup	132.77 MB	2	128 MB	part-00008
-rw-r--r--	dase-local	supergroup	132.97 MB	2	128 MB	part-00009
-rw-r--r--	dase-local	supergroup	132.51 MB	2	128 MB	part-00010
-rw-r--r--	dase-local	supergroup	132.92 MB	2	128 MB	part-00011
-rw-r--r--	dase-local	supergroup	132.86 MB	2	128 MB	part-00012
-rw-r--r--	dase-local	supergroup	132.8 MB	2	128 MB	part-00013
-rw-r--r--	dase-local	supergroup	132.91 MB	2	128 MB	part-00014
-rw-r--r--	dase-local	supergroup	132.6 MB	2	128 MB	part-00015
-rw-r--r--	dase-local	supergroup	132.03 MB	2	128 MB	part-00016
-rw-r--r--	dase-local	supergroup	132.95 MB	2	128 MB	part-00017

图 4.19 程序运行产生的文件

执行结果如图4.18所示，程序运行产生的文件如图4.19所示。

- 停止服务

```

1 ~/hadoop-2.10.1/sbin/stop-dfs.sh
2 ~/spark-2.4.7/sbin/stop-all.sh
3 ~/spark-2.4.7/sbin/stop-history-server.sh

```

4.5 思考题

- 1 既然采用 Cluster 提交方式在客户端看不到程序运行过程中的信息，那么这些运行过程中的信息如何查看？（提示：结合 HistoryServer 的 Web UI 进行阐述）
- 2 第4.4.5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。
- 3 对于第4.4.5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。