

## 实验二 Hadoop 2.x 部署与编程 (单机 + 伪分布式)

### 2.1 实验目的

- 学习 Hadoop 2.x 部署，理解单机集中式部署和单机伪分布式部署两种部署方式之间的区别
- 学会通过查找系统日志中的错误来解决系统部署中遇到的问题
- 学会基本的 HDFS Shell 操作命令
- 通过系统部署理解 Hadoop 2.x 的架构，以及 Hadoop 1.x 和 Hadoop 2.x 之间的差异
- 学会 IDEA 的安装与插件配置
- 掌握使用 IDEA 编写基于 Maven 项目的 Java/Scala 程序
- 学会使用 IDEA 将程序打成 jar 包
- 学习编写简单的基于 Java API 的 MapReduce 程序
- 学习在单机集中式和单机伪分布部署方式下运行 MapReduce 程序

### 2.2 实验任务

- 完成 Hadoop 2.x 的单机集中式部署与单机伪分布式部署
- 分别在两种部署方式下运行示例程序
- 完成 IDEA 的安装和激活，并配置 Maven 插件
- 完成 Java 环境的安装，并在 IDEA 中完成对 Java 的配置
- 完成 Scala 环境的安装，并在 IDEA 中完成对 Scala 的配置
- 完成将创建的 Maven 项目打成一个 jar 包
- 完成基于 Java API 编写的“WordCount”的 MapReduce 程序
- 在单机集中式和单机伪分布部署方式下调试、提交并运行该应用程序

### 2.3 实验环境

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

## 2.4 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）；使用 IntelliJ IDEA 进行编程的部分需要在本地机器上完成。

### 2.4.1 单机集中式部署

#### MapReduce

##### (1) 准备工作

- 登录用户 dase-local
- 在用户目录下，下载 Hadoop 2.10.1 并解压（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 cd ~  
2 wget https://archive.apache.org/dist/hadoop/common/hadoop-2.10.1/hadoop-2.10.1.  
tar.gz #下载安装包  
3 tar -xzf hadoop-2.10.1.tar.gz  
4 cd ~/hadoop-2.10.1  
5 ./bin/hadoop version #查看Hadoop版本信息
```

##### (2) 运行 MapReduce 应用程序

**注意：**在单机集中式部署 Hadoop 2.10.1 版本时，需要将其它版本 Hadoop 的相关进程都关闭，并且不需要修改配置，否则将会产生冲突。

- 提交 jar 包并查看运行结果

运行 Grep 示例：

```
1 cd ~/hadoop-2.10.1  
2 mkdir -p ~/input/grep #在用户目录下新建input文件夹  
3 cp ./etc/hadoop/*.*xml ~/input/grep #拷贝文件至输入文件夹中  
4 rm -rf ~/output/grep #删除输出路径  
5 ./bin/hadoop jar  
    ./share/hadoop/mapreduce/hadoop-mapreduce-examples-2.10.1.jar grep  
    ~/input/grep ~/output/grep 'dfs[a-z.]+' #运行 grep 示例  
6 cat ~/output/grep/* #查看输出结果
```

1           dfsadmin

图 2.1 jps 输出结果

- 在运行过程中查看进程
  - 运行 WordCount 示例，并且在执行该作业过程中查看系统启动的进程

```

1 cd ~/hadoop-2.10.1
2 rm -rf ~/output/wordcount #删除输出路径
3 ./bin/hadoop jar
   ./share/hadoop/mapreduce/hadoop-mapreduce-examples-2.10.1.jar
   wordcount ~/input/pd.train ~/output/wordcount

```

在作业执行时，另起一个终端，输入 jps 查看运行中的 Java 进程，详细信息如图2.2所示。

由图2.2(b)和图2.2(c)可知：在单机集中式部署方式下，系统只启动 RunJar 进程来运行整个 MapReduce 作业，此时也不会输出任何日志到本地，即生成 `~/hadoop-2.10.1/logs/` 目录和目录下日志文件。

## 2.4.2 单机伪分布式部署

### HDFS

#### (1) 准备工作

- 登录 dase-local 用户

#### (2) 修改 HDFS 配置

配置文件位于 `~/hadoop-2.10.1/etc/hadoop/` 目录下，修改其中的 `core-site.xml`、`hdfs-site.xml` 和 `hadoop-env.sh` 这 3 个文件。依次定位到这 3 个文件所在位置，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 `core-site.xml` 文件

```

1 <configuration>
2   <!-- 指定数据存放目录 -->
3   <property>
4     <name>hadoop.tmp.dir</name>
5     <value>/home/dase-local/hadoop-2.10.1/tmp</value>
6   </property>
7   <!-- 指定Hadoop所使用的文件系统schema (URI) , NameNode的地址 -->
8   <property>
9     <name>fs.defaultFS</name>
10    <value>hdfs://localhost:9000</value>
11  </property>
12</configuration>

```

- 修改 `hdfs-site.xml` 文件

```
File System Counters
  FILE: Number of bytes read=79944037369
  FILE: Number of bytes written=64178088942
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
Map-Reduce Framework
  Map input records=14892939
  Map output records=171038044
  Map output bytes=3974172730
  Map output materialized bytes=2629855579
  Input split bytes=6435
  Combine input records=171038044
  Combine output records=25326723
  Reduce input groups=23997207
  Reduce shuffle bytes=2629855579
  Reduce input records=25326723
  Reduce output records=23997207
  Spilled Records=50754861
  Shuffled Maps =65
  Failed Shuffles=0
  Merged Map outputs=65
  GC time elapsed (ms)=3025
  Total committed heap usage (bytes)=32548323328
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=2173827625
File Output Format Counters
  Bytes Written=2503965986
```

(a) 作业执行结束

```
17136 Jps
15739 RunJar
```

(b) 运行过程中执行 jps

```
bin  include  libexec      NOTICE.txt  sbin
etc  lib       LICENSE.txt  README.txt  share
```

(c) 执行完成后的目录内容

图 2.2 单机集中式部署作业运行情况

```
1 <configuration>
2   <!-- 指定HDFS上的文件副本数 -->
3   <property>
4     <name>dfs.replication</name>
5     <value>1</value>
6   </property>
7   <!-- 指定NameNode存储其元数据和编辑日志的目录的URI -->
8   <property>
9     <name>dfs.namenode.name.dir</name>
10    <value>file:/home/dase-local/hadoop-2.10.1/tmp/dfs/name</value>
11  </property>
12  <!-- 指定DataNode存储块所在目录的URI -->
13  <property>
14    <name>dfs.datanode.data.dir</name>
15    <value>file:/home/dase-local/hadoop-2.10.1/tmp/dfs/data</value>
16  </property>
17</configuration>
```

- 修改 hadoop-env.sh 文件

找到 `export JAVA_HOME=${JAVA_HOME}` 这行，将此行修改为：

`export JAVA_HOME=/usr/local/jdk1.8` 即可。

### (3) 启动 HDFS 服务

- 格式化 NameNode

**注意：**仅在第一次启动 HDFS 时才需要格式化 NameNode，如果是重启 HDFS，那么跳过“格式化 NameNode”这一步，直接执行下一步“启动 HDFS 服务”即可。此外，在进行 NameNode 格式化之前，如果 `~/hadoop-2.10.1/tmp/` 文件夹已存在，那么需要删除该文件夹后再执行以下格式化命令。

```
1 ~/hadoop-2.10.1/bin/hdfs namenode -format
```

- 启动 HDFS 服务

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

### (4) 查看 HDFS 服务信息

- 使用 `jps` 查看 HDFS 进程，验证 HDFS 是否成功启动

正常启动结果如图2.3所示。如启动失败，可参照下面的查看进程日志来排查原因。

- 查看 NameNode、DataNode、SecondaryNameNode 进程日志

```
13872 SecondaryNameNode
19366 Jps
13578 DataNode
13359 NameNode
```

图 2.3 单机伪分布式部署进程

本实验的进程日志记录在 `~/hadoop-2.10.1/logs/` 路径下，后缀为.log 的文件中。一般日志都是不断追加在日志文件末尾。因此，在文件末尾可以查看最近日志记录，通过查看记录的时间就能找到指定的日志信息。

- NameNode 进程日志  
默认位置：`~/hadoop-2.10.1/logs/hadoop-*-namenode-*.``log`
- DataNode 进程日志  
默认位置：`~/hadoop-2.10.1/logs/hadoop-*-datanode-*.``log`
- SecondaryNameNode 进程日志  
默认位置：`~/hadoop-2.10.1/logs/hadoop-dase-local-secondarynamenode-*.``log`
- 访问 HDFS Web 页面  
通过 NameNode 的 web 页面 `http://localhost:50070`，查看 HDFS 信息。可以看到目前集群中 Live Nodes 显示为 1（即有 1 个 DataNode 节点），如图2.4所示。

点击 Utilities-Browse the filesystem 可以查看文件目录信息。

#### (5) 常用的 HDFS Shell 命令操作

注意：第一次使用 HDFS 时，需要首先在 HDFS 中创建用户目录，如图2.5所示。

- 目录操作

HDFS 中的目录操作包括：新建目录、查看目录内容、删除目录等。示例如下：

```
1 cd ~/hadoop-2.10.1
2 ./bin/hdfs dfs -mkdir -p /user/dase-local #在 HDFS 中为当前 dase-local
   用户创建一个用户根目录 hdfs:///user/dase-local
3 ./bin/hdfs dfs -ls . #显示 hdfs:///user/dase-local 下的文件
4 ./bin/hdfs dfs -ls /user/dase-local #显示 hdfs://user/dase-local 下的文件
5 ./bin/hdfs dfs -mkdir input #新建 hdfs:///user/dase-local/input 目录
6 ./bin/hdfs dfs -rm -r input #删除 hdfs:///user/dase-local/input 目录
7 ./bin/hdfs dfs -mkdir /input #新建 hdfs:///input 目录
8 ./bin/hdfs dfs -rm -r /input #删除 hdfs:///input 目录
```

- 文件操作



图 2.4 HDFS 信息



图 2.5 新建目录后的 HDFS 信息

HDFS 中的文件操作包括：上传文件、下载文件、拷贝文件等。示例如下：

- 从本地文件系统向 HDFS 中上传文件

```

1 cd ~/hadoop-2.10.1
2 ./bin/hdfs dfs -mkdir input #新建 hdfs://user/dase-local/input 文件夹
3 ./bin/hdfs dfs -put README.txt input/ #把本地的 README.txt 上传到
   hdfs://user/dase-local/input
4 ./bin/hdfs dfs -cat input/README.txt #查看上传的 README.txt

```

- 将之前下载保存至 `~/input/` 的文件 `pd.train` 上传至 HDFS

```

1 ./bin/hdfs dfs -put ~/input/pd.train input/
#将本地文件系统的“~/input/pd.train”上传至

```

```
hdfs:///user/dase-local/input/目录下
```

此时另起一个终端，在上传过程中运行 jps，查看出现的进程名称，如图2.6所示。

```
6178 Jps
15622 NameNode
16134 SecondaryNameNode
6121 FsShell
15855 DataNode
```

图 2.6 文件上传过程中的进程

- 把 HDFS 中的文件下载到本地文件系统中

```
1 ~/hadoop-2.10.1/bin/hdfs dfs -get input/README.txt ~/Downloads #把
hdfs:///user/dase-local/input/README.txt 下载到本地 ~/Downloads
2 cat ~/Downloads/README.txt #查看下载的 README.txt
```

- 把文件从 HDFS 中的一个目录拷贝至 HDFS 中的另外一个目录

```
1 ~/hadoop-2.10.1/bin/hdfs dfs -cp input/README.txt . #把
hdfs:///user/dase-local/input/README.txt 拷贝至
hdfs:///user/dase-local 目录下
2 ~/hadoop-2.10.1/bin/hdfs dfs -cat README.txt #查看拷贝的 README.txt文件
```

## (6) 停止命令

- 关闭 HDFS

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

- 查看进程，验证是否成功停止服务

使用 jps 查看进程，不再出现 NameNode、DataNode、SecondaryNameNode 进程则表示服务已停止。

## MapReduce

### (1) 修改配置

以下修改的配置文件均在路径 `~/hadoop-2.10.1/etc/hadoop/` 下。

- 新建 mapred-site.xml 文件

```
1 cp ~/hadoop-2.10.1/etc/hadoop/mapred-site.xml.template
~/hadoop-2.10.1/etc/hadoop/mapred-site.xml #文件重命名
```

- 修改 mapred-site.xml

```
1 <configuration>
2   <!-- 执行框架设置为Hadoop YARN -->
3   <property>
4     <name>mapreduce.framework.name</name>
5     <value>yarn</value>
6   </property>
7   <!-- JobHistory服务器Web UI的端口-->
8   <property>
9     <name>mapreduce.jobhistory.webapp.address</name>
10    <value>localhost:19888</value>
11  </property>
12 </configuration>
```

- 修改 yarn-site.xml 文件：

```
1 <configuration>
2   <!-- 指定 Map Reduce 应用程序的 Shuffle Service 类型 -->
3   <property>
4     <name>yarn.nodemanager.aux-services</name>
5     <value>mapreduce_shuffle</value>
6   </property>
7   <!-- Task 的虚拟内存使用量超过物理内存的最大比值 -->
8   <property>
9     <name>yarn.nodemanager.vmem-pmem-ratio</name>
10    <value>2.1</value>
11  </property>
12 <!-- 是否进行虚拟内存限制比较 -->
13 <property>
14   <name>yarn.nodemanager.vmem-check-enabled</name>
15   <value>false</value>
16 </property>
17 <!-- 启用日志聚合，将程序运行日志信息上传到HDFS系统上，方便Yarn的UI查看日志 -->
18 <property>
19   <name>yarn.log-aggregation-enable</name>
20   <value>true</value>
21 </property>
22 <!-- 日志聚合服务器的 URL -->
23 <property>
24   <name>yarn.log.server.url</name>
25   <value>http://localhost:19888/jobhistory/logs</value>
26 </property>
```

27 | </configuration>

(2) 启动 Yarn 服务

- 启动 Yarn 服务

```
1 ~/hadoop-2.10.1/sbin/start-yarn.sh #启动Yarn  
2 ~/hadoop-2.10.1/sbin/mr-jobhistory-daemon.sh start historyserver  
#开启历史服务器，才能在 web 中查看作业日志
```

- 启动 HDFS 服务

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

(3) 查看 Yarn 服务信息

- 使用 jps 查看进程，验证是否成功启动服务

正常启动结果如图 2.7 所示。如果启动后缺少某些进程，说明启动过程中出现错误。参考下一步查看进程日志来排查原因，根据日志中的出错信息，在网上搜索查找相关解决方法。

```
29440 NodeManager  
30961 JobHistoryServer  
29266 ResourceManager  
31061 Jps  
28762 DataNode  
29050 SecondaryNameNode  
28527 NameNode
```

图 2.7 成功启动后的进程

- 查看 Yarn 进程日志

服务日志信息记录在 `~/hadoop-2.10.1/logs/` 路径下，后缀为 `.log` 的相关文件中。

- ResourceManager 进程日志

默认位置：`~/hadoop-2.10.1/logs/yarn-*-resourcemanager-* .log`

- NodeManager 进程日志

默认位置：`~/hadoop-2.10.1/logs/yarn-*-nodemanager-* .log`

- 访问 Yarn 的 Web 界面

通过 `http://localhost:8088`，查看 Yarn 的 Cluster 信息，如图 2.8 所示。

(4) 运行 MapReduce 程序

- 提交 jar 包并查看运行结果



图 2.8 Cluster 界面

```
6      dfs.audit.logger
4      dfs.class
3      dfs.logger
3      dfs.server.namenode.
2      dfs.audit.log.maxbackupindex
2      dfs.period
2      dfs.audit.log.maxfilesize
1      dfs.replication
1      dfs.log
1      dfs.file
1      dfs.datanode.data.dir
1      dfs.servers
1      dfsadmin
1      dfsmetrics.log
1      dfs.namenode.name.dir
```

图 2.9 输出文件内容

运行 Grep 示例：

```
1 cd ~/hadoop-2.10.1
2 ./bin/hdfs dfs -mkdir -p input/grep #新建输入文件夹
3 ./bin/hdfs dfs -put etc/hadoop/* input/grep #上传输入文件
4 ./bin/hdfs dfs -rm -r output/grep #删除输出路径
5 ./bin/yarn jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.10.1.jar
   grep input/grep output/grep 'dfs[a-z].+'
6 ./bin/hdfs dfs -cat output/grep/p* #查看输出文件
```

运行成功后，结果如图2.9所示。

- 查看运行过程中的进程
  - 运行 WordCount 示例，并且查看系统执行该作业过程中启动的进程

```
1 cd ~/hadoop-2.10.1
2 ./bin/hdfs dfs -rm -r output/wordcount #删除输出路径
3 ./bin/yarn jar
   share/hadoop/mapreduce/hadoop-mapreduce-examples-2.10.1.jar
```

wordcount input/pd.train output/wordcount

```
16642 NodeManager  
15622 NameNode  
16134 SecondaryNameNode  
9385 YarnChild  
9417 YarnChild  
9354 YarnChild  
9418 YarnChild  
17162 JobHistoryServer  
9358 YarnChild  
15855 DataNode  
9137 MRAppMaster  
10261 Jps  
16472 ResourceManager  
8988 RunJar  
9407 YarnChild
```

图 2.10 作业执行过程中的进程

- 在作业执行过程中，会新出现三种进程 RunJar, MRAppMaster, YarnChild，可通过 jps 查看，如图2.10所示。
    - \* RunJar: 唯一, 出现在提交作业的节点上
    - \* MRAppMaster: 唯一, 代表该作业的主进程
    - \* YarnChild: 多个, 代表执行作业的进程
- 可见伪分布式部署方式下, 系统确实进行了多进程并行计算。

#### (5) 查看 MapReduce 程序运行信息

- 访问 Hadoop Map/Reduce Administration 界面

通过 <http://localhost:8088/>，可以看到本次启动 Yarn 后提交的所有应用程序的相关信息。如图2.11所示。

ID	Name	Version	Submitter	AppMaster	Start time	Last modified	State	Progress	Running	Allocated	Allocated MB	Allocated Vcores	Allocated VMs	Received	Received MB	Received Vcores	Received VMs	Lost	Lost %	Lost Duration	Spilling	Spilled MB	Spilled Vcores	Spilled VMs	
mapred-site_161216072807_2012	word count	MAPREDUCE_DEFAULT_0	Map Reduce	Map Red	Mon Feb 1 15:54:37 2012	Mon Feb 1 15:54:38 2012	FINISHED	SUCCESS	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0%	00:00:00	0	0.0	0.0	0.0
mapred-site_161216072807_2012	word count	MAPREDUCE_DEFAULT_0	Map Reduce	Map Red	Tue Feb 2 13:39:54 2012	Tue Feb 2 13:40:53 2012	FINISHED	SUCCESS	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0%	00:00:59	0	0.0	0.0	0.0
mapred-site_161216072807_2012	word count	MAPREDUCE_DEFAULT_0	Map Reduce	Map Red	Tue Feb 2 13:39:54 2012	Tue Feb 2 13:40:53 2012	FINISHED	SUCCESS	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0%	00:00:59	0	0.0	0.0	0.0
mapred-site_161216072807_2012	word count	MAPREDUCE_DEFAULT_0	Map Reduce	Map Red	Tue Feb 2 13:39:54 2012	Tue Feb 2 13:40:53 2012	FINISHED	SUCCESS	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0%	00:00:59	0	0.0	0.0	0.0

图 2.11 All Applications

- 查看 MapReduce 应用程序日志
  - Task 日志

默认位置: `~/hadoop-2.10.1/logs/userlogs/<jobid>/<attempt-id>`

- 查看 MapReduce 应用程序历史记录

通过 `http://localhost:19888`, 即可看到程序的历史运行记录, 如图2.12所示。

Retired Jobs										
Show 20   Entries										
Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total
2021-02-08 15:49:57	2021-02-01 12:50:00	2021-02-01 13:34:17	job_16121264018697_0003	word count	dase-dis	default	SUCCEEDED	17	17	1
CST	CST	CST	2021-02-08 2021-02-01 2021-02-01	prep-search	dase-dis	default	SUCCEEDED	1	1	1
2021-02-11 15:34:22	15:35:32	15:35:32	job_16121264018697_0002	sort	dase-dis	default	SUCCEEDED	1	1	1
CST	CST	CST	2021-02-01 2021-02-01 2021-02-01	prep-search	dase-dis	default	SUCCEEDED	29	29	1
2021-02-01 15:34:23	15:35:07	15:35:07	job_16121264018697_0001	search	dase-dis	default	SUCCEEDED	1	1	1

图 2.12 JobHistory

#### (6) 停止 Yarn 服务

- 停止命令

```
1 ~/hadoop-2.10.1/sbin/stop-yarn.sh
2 ~/hadoop-2.10.1/sbin/mr-jobhistory-daemon.sh stop historyserver
3 ~/hadoop-2.10.1/sbin/stop-dfs.sh
```

- 查看进程, 验证是否成功停止服务

正常停止后, 执行 `jps` 命令, ResourceManger、NodeManager、JobHistoryServer、NameNode、SecondaryNameNode、DataNode 进程不会出现。

### 2.4.3 IDEA 的安装与激活

(1) 从官网 (<https://www.jetbrains.com/zh-cn/idea/>) 下载 IntelliJ IDEA 的 Ultimate 版本

(2) 具体安装与激活过程

- 运行安装程序
- 勾选 “I confirm that I have read and accept the terms of this User Agreement”, 并点击 “Continue”
- 选择是否 “DATA SHAREING”, 点击 “Don’t Send”
- 在 IDEA 官网 (<https://account.jetbrains.com/login>) 利用学校邮箱注册一个账户, 并登录学校邮箱操作, 获得 License ID
- 利用注册账号的用户名或者邮箱激活 IDEA, 如图2.13所示, 激活后的界面如图2.14 所示

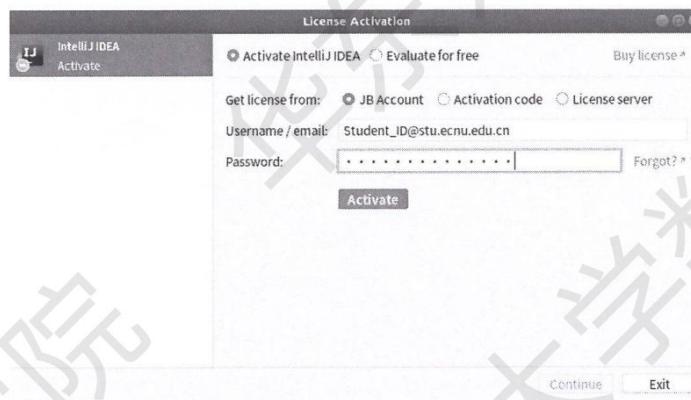


图 2.13 IDEA 激活界面



图 2.14 IDEA 开始界面

#### 2.4.4 IDEA 中 Java 环境的安装与配置

- (1) 打开 IDEA，如若进入工程界面，依次选择 File->Close Project，返回 IDEA 主界面
- (2) 在 IDEA 主界面中选择 “Configure” -> “Structure for New Projects”
- (3) 在弹出页面中选中左侧 “Platform settings” 中的 “SDKs”，此时点击中栏中的“+”号，选择 “Download JDK...”，如图2.15所示
- (4) 在弹出页面中选中如图2.16所示的 JDK 来源与版本，点击 Download
- (5) 下载完成后，在如图2.17所示的界面中点击 OK

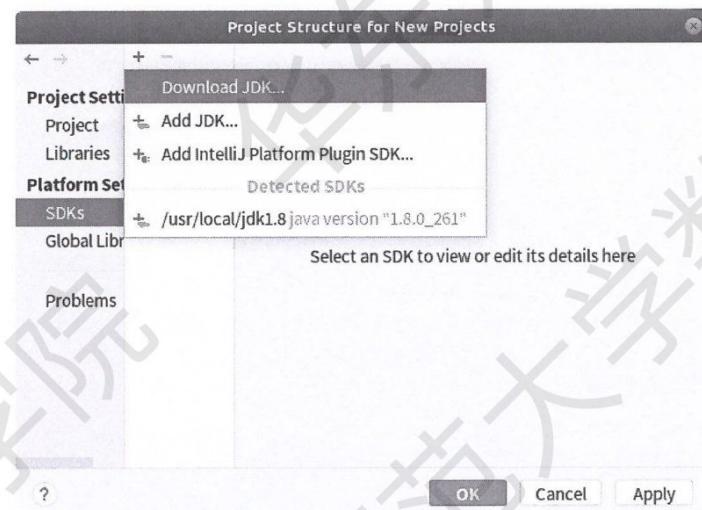


图 2.15 选择下载 JDK



图 2.16 选择 JDK 来源与版本

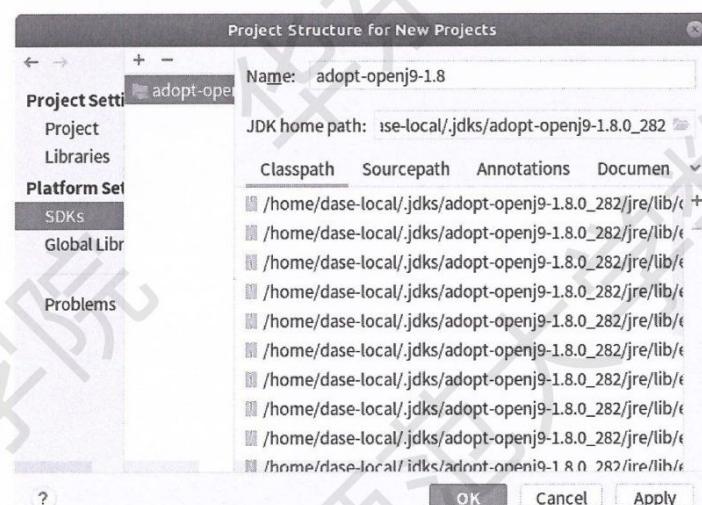


图 2.17 使用下载完成的 JDK

### 2.4.5 基于 Maven 项目编写 Java 程序

具体创建过程如下：

#### (1) 更新 maven 插件

IDEA 中集成了 maven 插件。在 IDEA 开始界面点击“Configure”->“Setting”进入配置界面，如图2.18 所示。

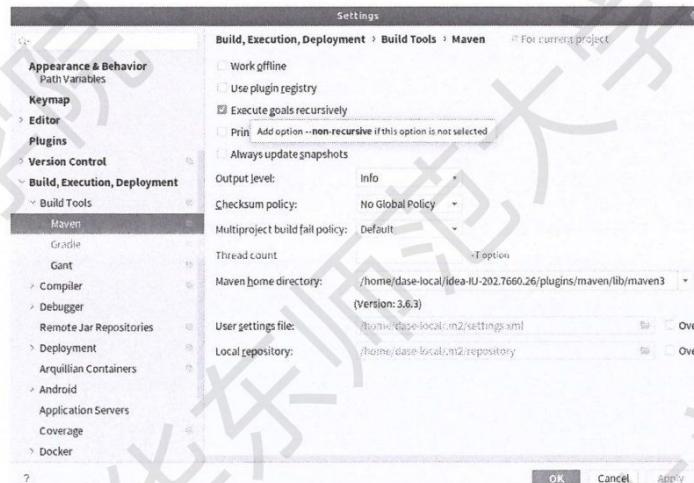


图 2.18 Maven 插件配置界面

#### (2) 新建一个 Maven 项目并调试运行

- 在 IDEA 开始界面点击“+ NEW Project”
- 选择相应配置

在弹出来的界面中，选择“Maven”、已安装的 JDK，并点击“next”，如图2.19 所示。

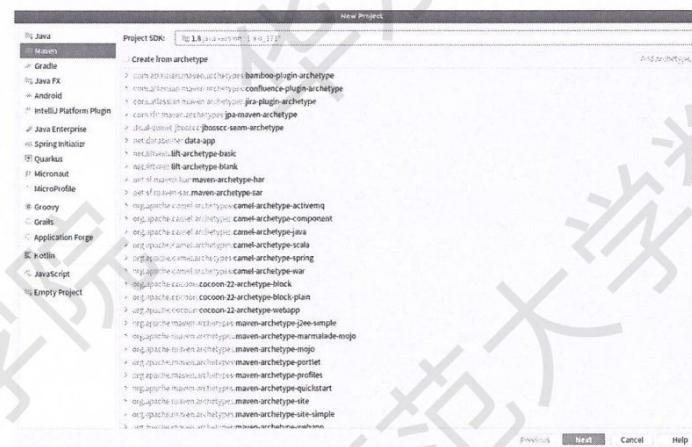


图 2.19 新建 Maven 项目

- 输入项目信息

在弹出来的界面中，填写“Name”（项目名）、“Location”（项目存储位置）、GroupId（项目组织唯一的标识符，实际对应Java的包的结构，是main目录里java的目录结构名称）、ArtifactId（项目的唯一标识符，实际对应项目的名称）和Version（项目的当前版本），点击“Finish”。如图2.20所示，新建了一个名为“MavenDemo”的项目。



图 2.20 Maven 项目信息

- 新建 Java 类

在项目的src/main/java路径下，新建一个名为“HelloWorld”的类。

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         System.out.println(args[0]);  
4     }  
5 }
```

- 配置运行环境

- 在菜单界面选择“Run”->“Edit Configurations”
- 在弹出的窗口中左侧点击“+”，选择“Application”
- 在新建的页面中，填写“Name”（应用名称）、“Mainclass”（主类的名称）、“Program arguments”（运行参数，即args[x]的值，多个值之间用空格隔开），如图2.21所示
- 点击“Apply”后点击“OK”

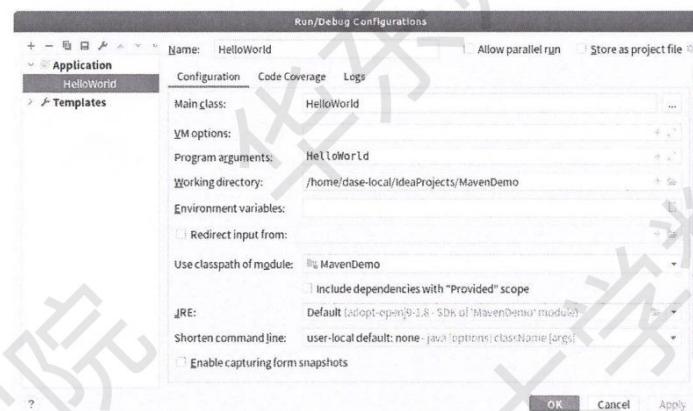


图 2.21 运行参数配置

- 在 IDEA 中直接运行

在菜单界面选择 “Run” -> “Run ‘HelloWorld’”，控制台输出如图 2.22 所示。

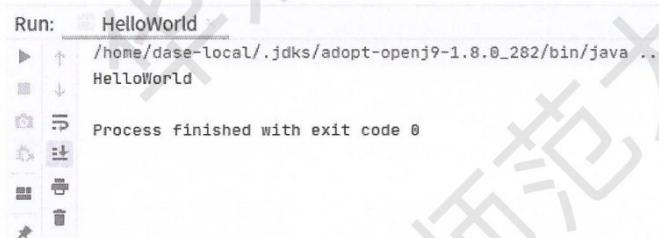


图 2.22 IDEA 控制台输出结果

#### 2.4.6 IDEA 中 Scala 环境的安装与配置（选做）

具体安装与配置过程如下：

##### (1) 为 IDEA 安装 Scala 插件

在菜单中选择 “File” -> “Settings”，在弹出窗口的左侧边栏选择 “Plugins”，在搜索栏输入 “Scala”，如图2.23所示。选择 “Install” 进行安装，随后选择 “RestartIDE” 重启 IDEA。



图 2.23 搜索 Scala 插件

## (2) 安装 Scala SDK

- 返回 IDEA 主界面，依次选择“Configure”->“Structure for New Projects”，点击中栏的“+”号，选择“Scala SDK”选项，如图2.24所示

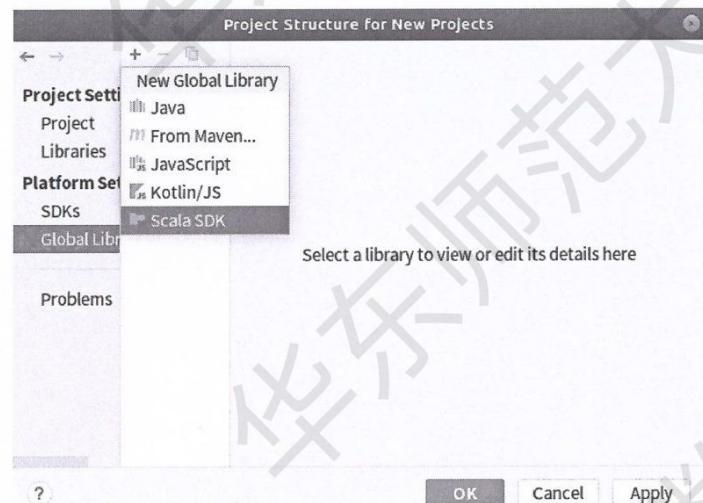


图 2.24 选择为新建项目配置 Scala SDK

- 点击 Scala SDK 选项后，在弹出的界面中选择 Download，进入如图2.25所示界面，选择 OK

## 2.4.7 基于 Maven 项目编写 Scala 程序（选做）

### (1) 新建 Maven 项目

按照2.4.5节内容，新建一个 Maven 项目。

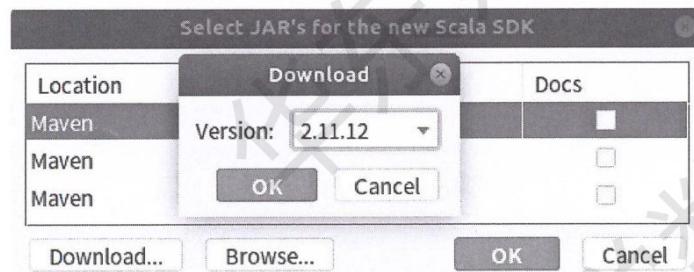


图 2.25 选择 Scala SDK 对应版本

#### (2) 创建 Scala 代码路径

展开项目目录，右键点击 `src/main/` 目录，依次选择 “New” -> “Directory”，输入文件夹名称 “scala”。接着右键点击 `scala` 目录，依次选择 `Mark Directory as -> Sources Root`。`scala` 文件夹颜色会变为蓝色。

#### (3) 导入 Scala 相关配置

依次选择 “File” -> “Project Structure...”，在弹出的页面中选择 “Platform Settings” 下的 “Global Libraries”。右键选中中栏中的 `scala-sdk-2.11.12`，选择 `Add to Modules... ,` 接着选择 “OK”。

#### (4) 新建 object “Hello World”

右键单击 `src/main/scala` 目录，依次选择 “New->Scala Class”，输入文件名称 “HelloWorld”，选择 “Object”。输入如下代码：

```
1 object HelloWorld {  
2     def main(array:Array[String]): Unit ={  
3         println("HelloWorld");  
4     }  
5 }
```

#### (5) 配置运行环境

- 在菜单界面选择 “Run” -> “Edit Configurations”
- 在弹出的窗口中左侧点击 “+”，选择 “Application”
- 在新建的页面中，填写“Name”(应用名称)、“Main class”(主类的名称)、“Program arguments”(运行参数，即 `args[x]` 的值，多个值之间用空格隔开)，如图2.26 所示
- 点击 “Apply” 后点击 “OK”



图 2.26 运行参数配置

#### (6) 在 IDEA 中直接运行

在菜单界面选择“Run”->“Run ‘HelloWorld’”，控制台输出如图 2.27 所示。

图 2.27 IDEA 控制台输出结果

### 2.4.8 使用 IDEA 将程序打包为 jar 包

以上述基于 Maven 项目编写的 Java 程序为例，具体打包过程如下：

- (1) 选中 Java 项目工程名称，在菜单中选择“File”->“Project Structure”
- (2) 在弹出的窗口中左侧选中“Artifacts”，点击“+”按钮，选择“jar”->“Empty”
- (3) 配置 jar 包参数

在打开的界面中配置相关参数，如图 2.28 所示，配置了名为“HelloWorld”的 jar 包，存放在“/home/dase-local/IdeaProjects/HelloWorld/out/artifacts/HelloWorld”目录下。配置完成后，点击“Apply”后点击“OK”。

- 填写“Name”（生成 jar 包的名称）
- 点击“Output Layout”下的“+”按钮，选择“Module Output”，然后选择需要打包的“Module”，即将编译生成的 class 文件打包到 jar 包中
- 选中 jar 包，点击“Create Manifest File”，选择项目根目录
- 选中 jar 包，填写“Main Class”（主类）

- 勾选“Include in project build”，即 IDEA 每次运行项目时，都会自动执行打包流程，更新 jar 包
- 点击“Apply”后点击“OK”

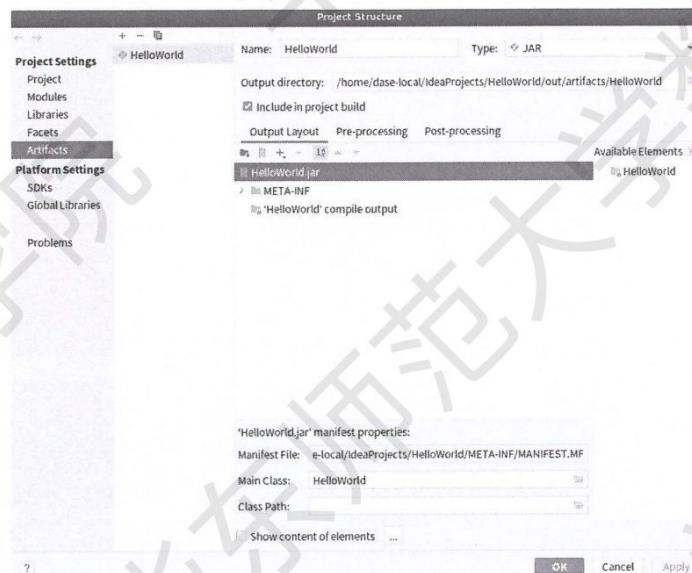


图 2.28 jar 包配置界面

#### (4) 在主界面构建项目以生成 jar 包

- 方法一：重新运行 Java 程序，项目自动打包生成 jar 包
- 方法二：在菜单中选择“Build”->“Build Artifacts”，然后选择“Build”或者“Rebuild”，生成 jar 包

### 2.4.9 编写 MapReduce 应用程序

#### (1) 新建 Maven 项目并添加依赖

- 新建名为“MapReduceDemo”的 Maven 项目（可参考第2.4.5节）
- 编辑项目根目录下的 pom.xml 文件，在<dependencies>标签中添加 Hadoop 相关的依赖包 hadoop-common, hadoop-client, hadoop-hdfs

```

1 <dependencies>
2   <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common
3       -->
4   <dependency>
5     <groupId>org.apache.hadoop</groupId>
6     <artifactId>hadoop-common</artifactId>
7     <version>2.10.1</version>
8   </dependency>

```

```

8   <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-client
9   -->
10  <dependency>
11    <groupId>org.apache.hadoop</groupId>
12    <artifactId>hadoop-client</artifactId>
13    <version>2.10.1</version>
14  </dependency>
15  <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs -->
16  <dependency>
17    <groupId>org.apache.hadoop</groupId>
18    <artifactId>hadoop-hdfs</artifactId>
19    <version>2.10.1</version>
20  </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 Java 程序

编写一个 Java 应用程序，实现对给定文本中的单词进行计数。

- 在项目的 *src/main/java* 目录下，选择 New->Package，输入名称 *cn.edu.ecnu.mapreduce.example.java.wordcount*
- 右键单击建好的包，选择 New->Java Class，输入名称 *WordCountMapper*

```

1 package cn.edu.ecnu.mapreduce.example.java.wordcount;
2
3 import org.apache.hadoop.io.IntWritable;
4 import org.apache.hadoop.io.LongWritable;
5 import org.apache.hadoop.io.Text;
6 import org.apache.hadoop.mapreduce.Mapper;
7
8 import java.io.IOException;
9
10 /* 步骤1：确定输入键值对[K1,V1]的数据类型为[LongWritable,Text]，输出键值对
11    [K2,V2]的数据类型为[Text,IntWritable] */
12 public class WordCountMapper extends Mapper<LongWritable, Text, Text,
13                                     IntWritable> {
14
15     @Override
16     protected void map(LongWritable key, Text value, Context context)
17         throws IOException, InterruptedException {
18         /* 步骤2：编写处理逻辑将[K1,V1]转换为[K2,V2]并输出 */
19     }
20
21 }

```

```
17 // 以空格作为分隔符拆分成单词  
18 String[] datas = value.toString().split(" ");  
19 for (String data : datas) {  
20     // 输出分词结果  
21     context.write(new Text(data), new IntWritable(1));  
22 }  
23 }  
24 }
```

- 右键单击建好的包，选择 New->Java Class，输入名称 *WordCountReducer*

```
1 package cn.edu.ecnu.mapreduce.example.java.wordcount;  
2  
3 import org.apache.hadoop.io.IntWritable;  
4 import org.apache.hadoop.io.Text;  
5 import org.apache.hadoop.mapreduce.Reducer;  
6  
7 import java.io.IOException;  
8  
9 /* 步骤1：确定输入键值对 [K2,List(V2)] 的数据类型为 [Text,  
10    IntWritable]，输出键值对 [K3,V3] 的数据类型为 [Text,IntWritable] */  
11 public class WordCountReducer extends Reducer<Text, IntWritable, Text,  
12    IntWritable> {  
13     @Override  
14     protected void reduce(Text key, Iterable<IntWritable> values, Context  
15                           context)  
16         throws IOException, InterruptedException {  
17         /* 步骤2：编写处理逻辑将 [K2,List(V2)] 转换为 [K3,V3] 并输出 */  
18         int sum = 0;  
19         // 遍历累加求和  
20         for (IntWritable value : values) {  
21             sum += value.get();  
22         }  
23         // 输出计数结果  
24         context.write(key, new IntWritable(sum));  
25     }  
26 }
```

- 右键单击建好的包，选择 New->Java Class，输入名称 *WordCount*

```
1 package cn.edu.ecnu.mapreduce.example.java.wordcount;  
2  
3 import org.apache.hadoop.conf.Configured;
```

```
4 import org.apache.hadoop.fs.Path;
5 import org.apache.hadoop.io.IntWritable;
6 import org.apache.hadoop.io.Text;
7 import org.apache.hadoop.mapreduce.Job;
8 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
10 import org.apache.hadoop.util.Tool;
11 import org.apache.hadoop.util.ToolRunner;
12
13 public class WordCount extends Configured implements Tool {
14
15     @Override
16     public int run(String[] args) throws Exception {
17         /* 步骤1：设置作业的信息 */
18         Job job = Job.getInstance(getConf(), getClass().getSimpleName());
19         // 设置程序的类名
20         job.setJarByClass(getClass());
21
22         // 设置数据的输入输出路径
23         FileInputFormat.addInputPath(job, new Path(args[0]));
24         FileOutputFormat.setOutputPath(job, new Path(args[1]));
25
26         // 设置map和reduce方法
27         job.setMapperClass(WordCountMapper.class);
28         job.setReducerClass(WordCountReducer.class);
29
30         // 设置map方法的输出键值对数据类型
31         job.setMapOutputKeyClass(Text.class);
32         job.setMapOutputValueClass(IntWritable.class);
33         // 设置reduce方法的输出键值对数据类型
34         job.setOutputKeyClass(Text.class);
35         job.setOutputValueClass(IntWritable.class);
36
37         return job.waitForCompletion(true) ? 0 : 1;
38     }
39
40     public static void main(String[] args) throws Exception {
41         /* 步骤2：运行作业 */
42         int exitCode = ToolRunner.run(new WordCount(), args);
43         System.exit(exitCode);
44     }
45 }
```

### 2.4.10 调试 MapReduce 应用程序

使用 IDEA 调试 WordCount 应用程序，若在 Windows 下调试 MapReduce 应用程序，则需按照附录 A 先进行相关配置。

#### (1) 准备数据

使用之前准备的数据集 pd.train，数据集的位置为 `/home/dase-local/input/`。

#### (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 WordCount，配置界面如图 2.29 所示。

a) 配置 Main Class 为 `cn.edu.ecnu.mapreduce.example.java.wordcount.WordCount`

b) 配置 Program arguments 为 `/home/dase-local/input/ /home/dase-local/IdeaProjects/MapReduceDemo/output`

该程序需要传入两个参数，第一个参数为输入文件路径，即要统计单词数的文本文件或文件夹的路径，若输入的是文件夹的路径，输出的是文件夹中所有文本文件的单词数统计结果；第二个参数为输出文件路径，即存放统计结果的文件路径。输出文件路径无需在程序运行前创建好，程序运行过程中会生成该输出文件路径。

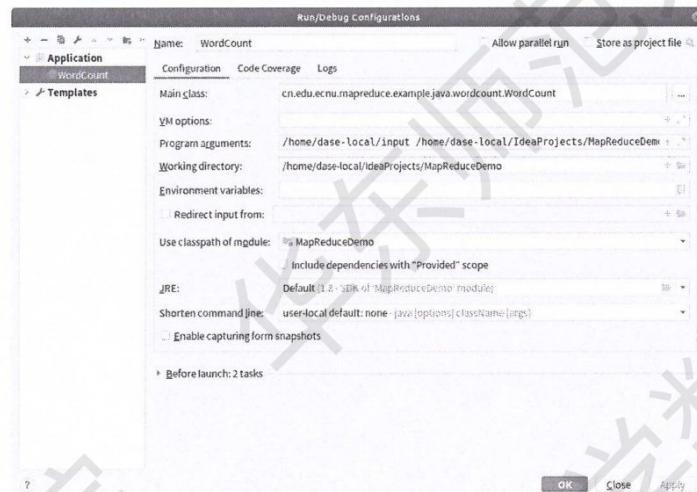


图 2.29 运行参数配置界面

#### (3) 运行应用程序

在菜单栏点击 Run->Run ‘WordCount’，在 IDEA 中直接运行程序，待运行完毕后，项目根目录下会出现“output”文件夹，其中“part-r-00000”文件就是运行结果，如图 2.30 所示。

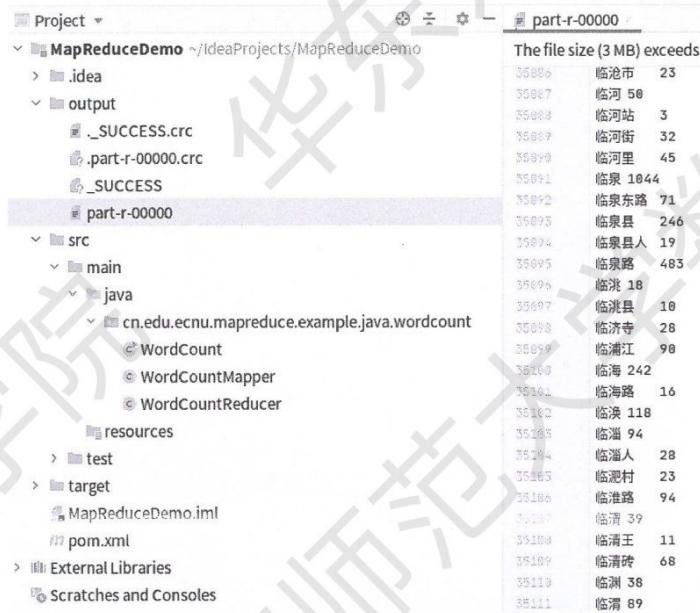


图 2.30 IDEA 中应用程序调试结果

### 2.4.11 运行 MapReduce 应用程序

在部署的 Hadoop 上运行 WordCount 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 将项目打包成可执行 jar 包

jar 包名称为 WordCount.jar, 打包路径为 /home/dase-local/IdeaProjects/MapReduceDemo/out/artifacts/WordCount/, 配置界面如图2.31所示。

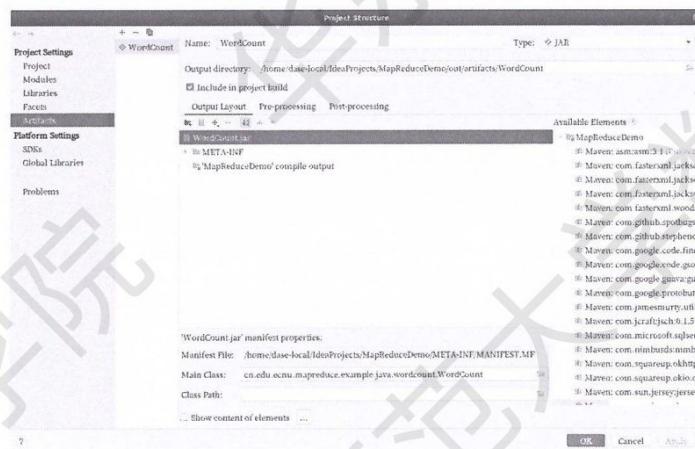


图 2.31 将项目打包为可执行 jar 包

- 将之前打好的包拷贝至`/home/dase-local/hadoop-2.10.1/myApp`路径下。

## (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 Yarn 和 HDFS 服务

```

1 su dase-local
2 ~/hadoop-2.10.1/sbin/start-yarn.sh
3 ~/hadoop-2.10.1/sbin/mr-jobhistory-daemon.sh start historyserver
4 ~/hadoop-2.10.1/sbin/start-dfs.sh

```

- 上传输入文件

输入文件 `pd.train` 在上一章中已上传至 `hdfs://localhost:9000/user/dase-local/input/`。

- 通过提交 jar 包运行应用程序

```

1 cd hadoop-2.10.1/
2 ./bin/hadoop jar ./myApp/WordCount.jar input/pd.train output
    #使用hadoop命令在Yarn模式下运行jar包，并在运行时输入参数

```

该程序实现了统计 HDFS “`/user/dase-local/input/`” 目录下的 “`pd.train`” 文件单词数量的操作。运行结果存储在 HDFS 的 “`/user/dase-local/output/`” 目录下的 “`part-r-00000`” 文件中，如图2.32所示。

- 查看运行结果

```

1 cd hadoop-2.10.1/
2 ./bin/hdfs dfs -tail output/part-r-00000 #查看输出结果

```

通过命令行查看的输出结果如图 2.33 所示。

图 2.32 单机伪分布式部署下应用程序运行结果存储位置

File	Size	Last Modified
SUCCESS	0B	Apr 26 14:38
part-r-00000	2.85 MB	Apr 26 14:38

- 停止相关服务

```

1 ~/hadoop-2.10.1/sbin/stop-yarn.sh
2 ~/hadoop-2.10.1/sbin/mr-jobhistory-daemon.sh stop historyserver
3 ~/hadoop-2.10.1/sbin/stop-dfs.sh

```

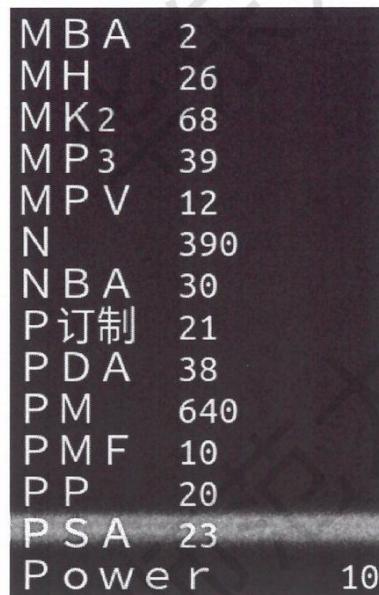


图 2.33 单机伪分布式部署下应用程序运行结果

## 2.5 思考题

- 1 图2.21中的 VM option 是什么？
- 2 JVM 是什么？它是操作系统中的进程吗？
- 3 同一个工程中的代码是否可以同时使用 Java 和 Scala 两种语言编写？
- 4 如何从 Hadoop JobHistory 的 Web UI 中查看某个 MapReduce 应用程序分别启动了多少个 Map 任务和 Reduce 任务？
- 5 如何从 Hadoop JobHistory 的 Web UI 中查看某个 MapReduce 应用程序启动的 Map 任务和 Reduce 任务分别是在哪些 NodeManager 上执行的？
- 6 对于一个 MapReduce 应用程序，是否存在某一时刻 Map 任务和 Reduce 任务同时运行？请结合 Hadoop JobHistory 的 Web UI 给出例证。

## 附录

### A Windows 下调试 MapReduce 应用程序

#### (1) 下载 winutils

在 Windows 下, Hadoop 需要一些 native 库来访问本地文件系统, 否则无法正确运行 MapReduce 程序。因此, 需要下载所需的 native 库, 即 winutils<sup>1</sup>。将文件夹下的 hadoop.dll、winutils.exe 等文件放置在 C:\Hadoop\bin 目录下。

#### (2) 配置环境变量

- 在“此电脑”图标上右键选择属性, 然后选择“高级系统设置”, 如图 2.34 所示。

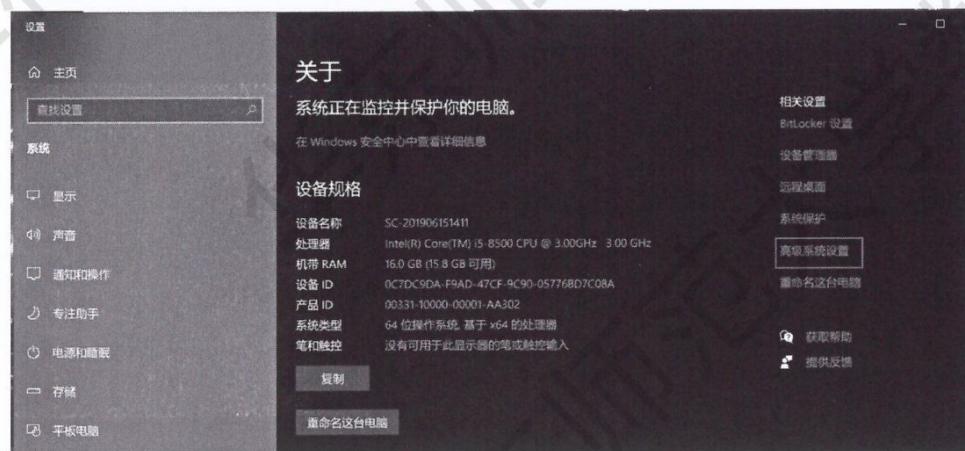


图 2.34 选择高级系统设置

- 在弹出的界面中选择“高级”并点击该界面的“环境变量”，如图 2.35 所示。
- 点击系统变量下的“新建”按钮，并添加名为“HADOOP\_HOME”的环境变量，如图 2.36 所示。
- 点击系统变量中名为“PATH”的环境变量，并在其中添加 winutils 相关的路径，如图 2.37 所示。

#### (3) 编程注意事项

- 环境变量配置完成之后需要重启 IDE。
- 请确保 winutils 版本大于或等于程序引入的 hadoop 依赖版本, 推荐下载最新的 3.3.6 版本的 winutils。

<sup>1</sup><https://github.com/cdarlnt/winutils/tree/master/hadoop-3.3.6/bin>

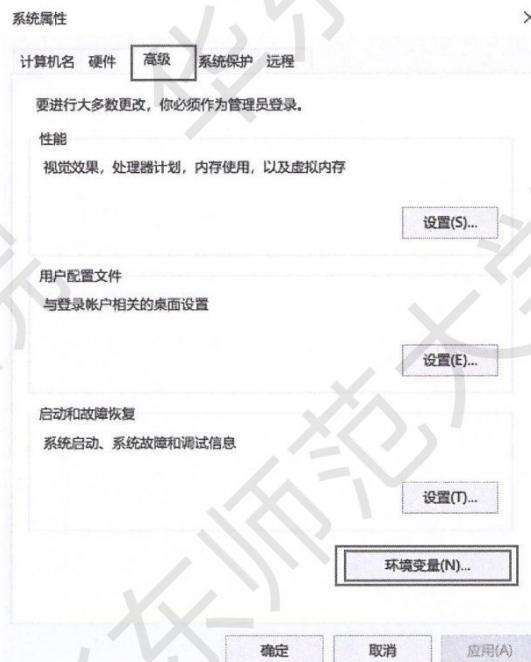


图 2.35 更改环境变量配置



图 2.36 添加名为 HADOOP\_HOME 的环境变量



图 2.37 在 PATH 中添加 winutils 相关路径