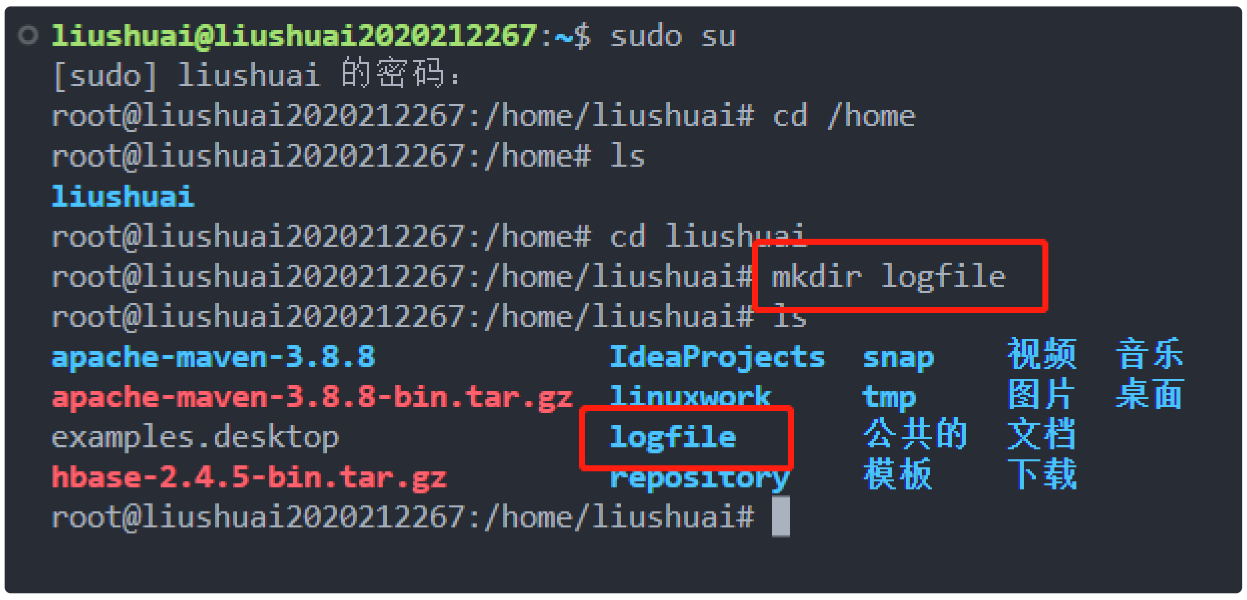
实验四 Spark Streaming

刘帅 2020212267

一. Spark Streaming 伪分布实现

1.1 利用 Spark Streaming 对文件流进行处理

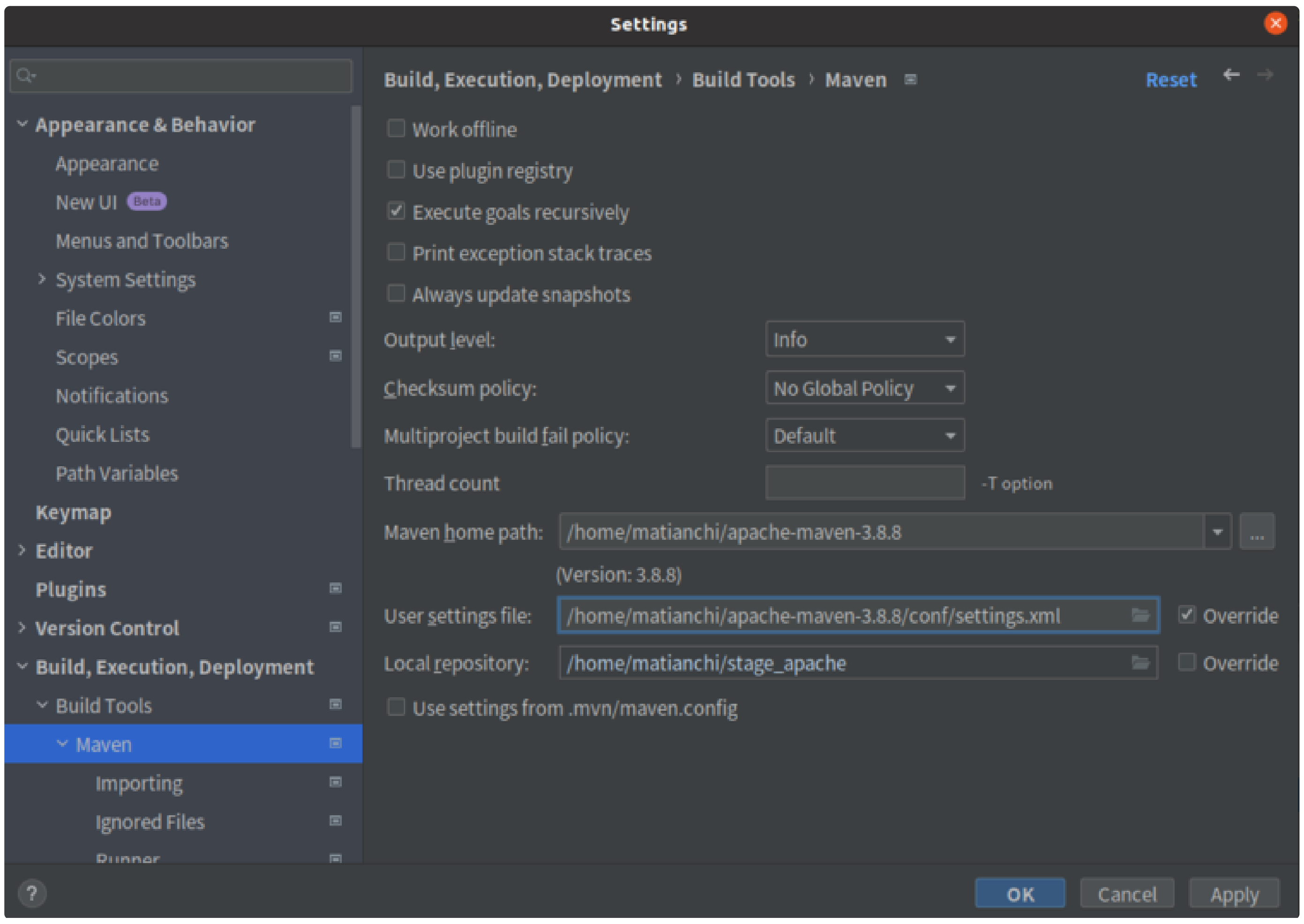
1.1.1 在 Linux 系统中创建一个 logfile 目录



```
liushuai@liushuai2020212267:~$ sudo su
[sudo] liushuai 的密码：
root@liushuai2020212267:/home/liushuai# cd /home
root@liushuai2020212267:/home# ls
liushuai
root@liushuai2020212267:/home# cd liushuai
root@liushuai2020212267:/home/liushuai# mkdir logfile
root@liushuai2020212267:/home/liushuai# ls
apache-maven-3.8.8              IdeaProjects  snap    视频  音乐
apache-maven-3.8.8-bin.tar.gz   linuxwork     tmp     图片  桌面
examples.desktop                logfile       公共的  文档
hbase-2.4.5-bin.tar.gz          repository    模板    下载
root@liushuai2020212267:/home/liushuai# 
```

1.1.2 新建 SparkStream 项目

1.1.2.1 配置 setting，指定 `Maven` 的仓库目录



Build, Execution, Deployment › Build Tools › Maven

□ Work offline

□ Use plugin registry

☑ Execute goals recursively

□ Print exception stack traces

□ Always update snapshots

Output level: Info

Checksum policy: No Global Policy

Multiproject build fail policy: Default

Thread count: -T option

Maven home path: /home/matianchi/apache-maven-3.8.8

(Version: 3.8.8)

User settings file: /home/matianchi/apache-maven-3.8.8/conf/settings.xml ☑ Override

Local repository: /home/matianchi/stage_apache □ Override

□ Use settings from .mvn/maven.config

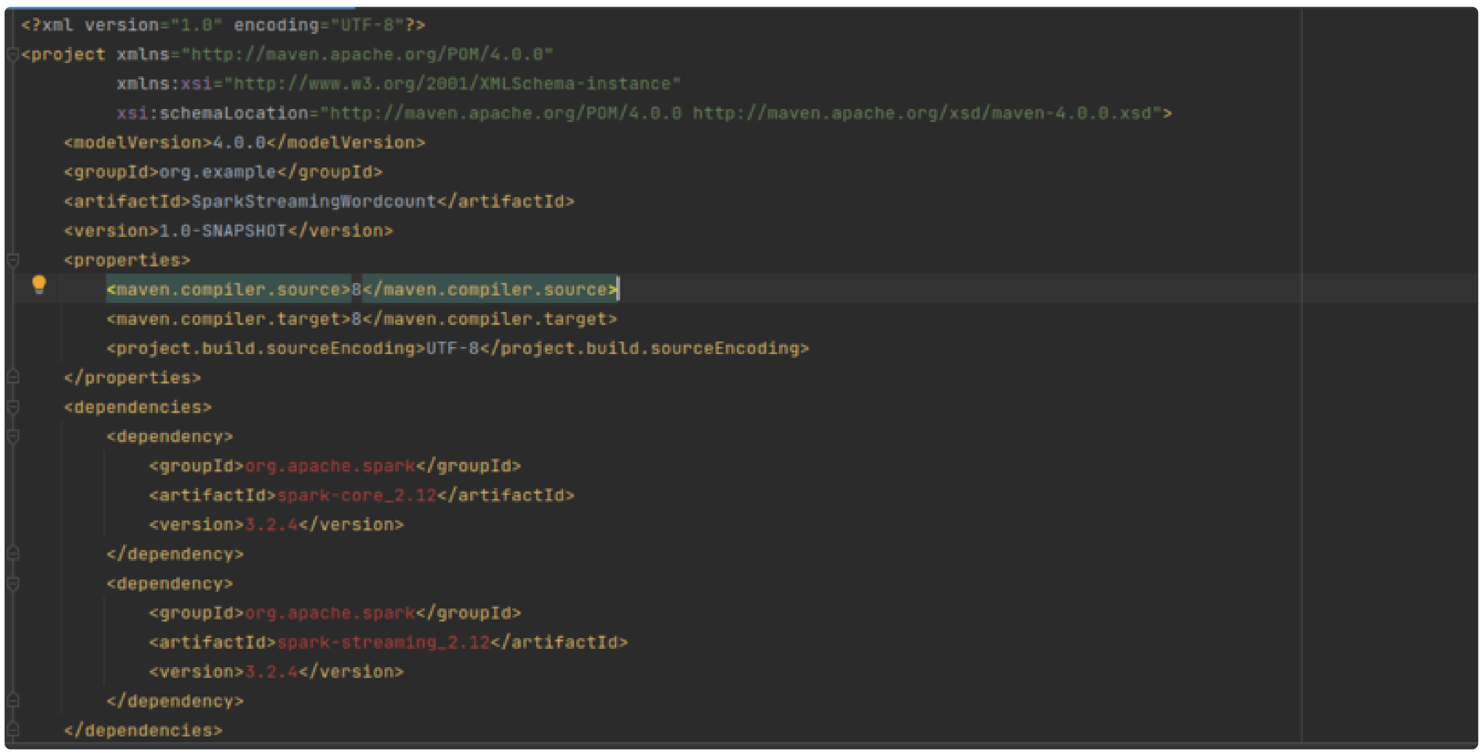
1.1.2.2 更改 `Importing` 和 `Runner` 设置，在 IDEA 中设置 `maven` 编译时忽略 HTTPS 的 SSL 证书验证.

在 `Importing` 中添加

在 `Runner` 中添加

1.1.2.3 pom.xml 文件如下

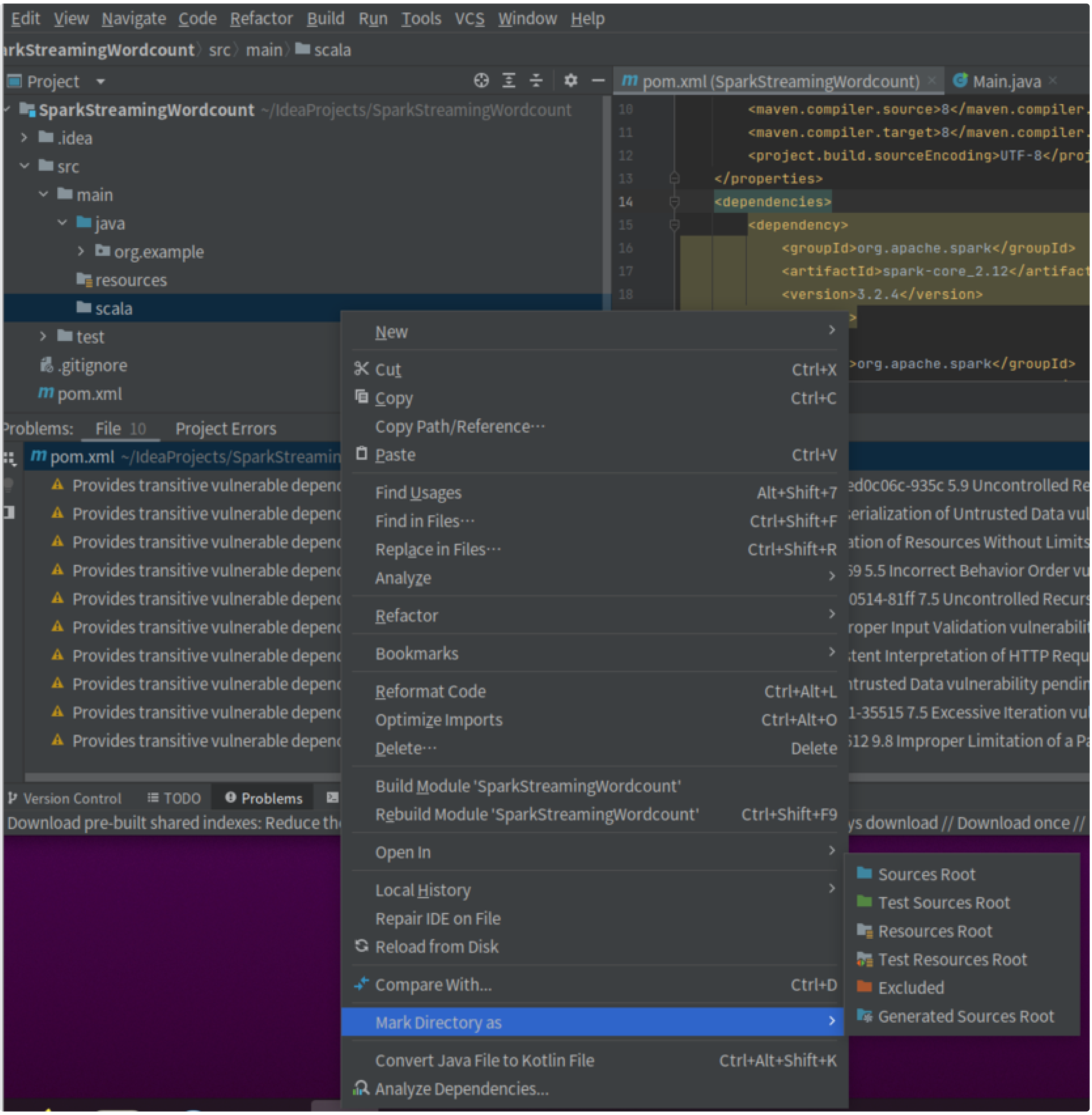
主要是添加 `spark-core_2.12` 依赖和 `spark-streaming_2.12` 依赖



```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.example</groupId>
    <artifactId>SparkStreamingWordcount</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_2.12</artifactId>
            <version>3.2.4</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming_2.12</artifactId>
            <version>3.2.4</version>
        </dependency>
    </dependencies>
```

创建一个 SparkConf 对象，设置应用程序的名称为"FileStreamWordCount"，并将 Spark 的运行模式设置为本地模式，使用所有可用的 CPU 核心。

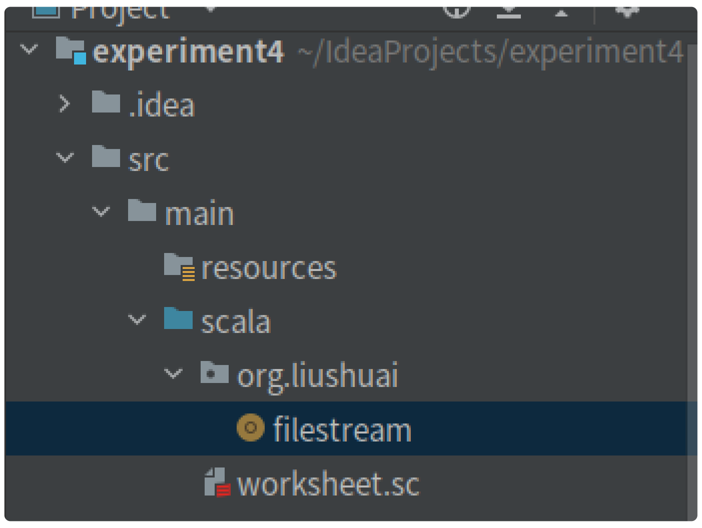
1.1.2.4 新建 `scala` 目录，并将其设置为主目录


Edit View Navigate Code Refactor Build Run Tools VCS Window Help
rkStreamingWordcount 〉 src 〉 main 〉 scala
Project
pom.xml (SparkStreamingWordcount)
Main.java
SparkStreamingWordcount ~/IdeaProjects/SparkStreamingWordcount
.idea
src
main
java
org.example
resources
scala
test
.gitignore
pom.xml
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path/Reference…
Paste Ctrl+V
Find Usages Alt+Shift+7
Find in Files… Ctrl+Shift+F
Replace in Files… Ctrl+Shift+R
Analyze
Refactor
Bookmarks
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Delete… Delete
Build Module 'SparkStreamingWordcount'
Rebuild Module 'SparkStreamingWordcount' Ctrl+Shift+F9
Open In
Local History
Repair IDE on File
Reload from Disk
Compare With... Ctrl+D
Mark Directory as
Convert Java File to Kotlin File Ctrl+Alt+Shift+K
Analyze Dependencies...
Sources Root
Test Sources Root
Resources Root
Test Resources Root
Excluded
Generated Sources Root
Problems: File 10 Project Errors
pom.xml ~/IdeaProjects/SparkStreamin
Version Control
TODO
Problems

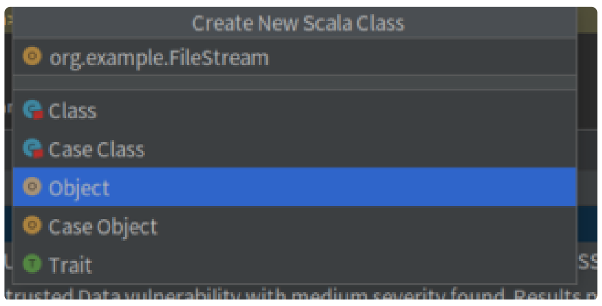
```
        <maven.compiler.source>8</maven.compiler.
        <maven.compiler.target>8</maven.compiler.
        <project.build.sourceEncoding>UTF-8</proj
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_2.12</artifact
            <version>3.2.4</version>
```

```
Provides transitive vulnerable depenc  ed0c06c-935c 5.9 Uncontrolled Re
Provides transitive vulnerable depenc  erialization of Untrusted Data vul
Provides transitive vulnerable depenc  ation of Resources Without Limits
Provides transitive vulnerable depenc  59 5.5 Incorrect Behavior Order vu
Provides transitive vulnerable depenc  0514-81ff 7.5 Uncontrolled Recurs
Provides transitive vulnerable depenc  roper Input Validation vulnerabilit
Provides transitive vulnerable depenc  tent Interpretation of HTTP Requ
Provides transitive vulnerable depenc  ntrusted Data vulnerability pendin
Provides transitive vulnerable depenc  1-35515 7.5 Excessive Iteration vul
Provides transitive vulnerable depenc  512 9.8 Improper Limitation of a Pa
Download pre-built shared indexes: Reduce th  ys download // Download once //
```

文件组织形式



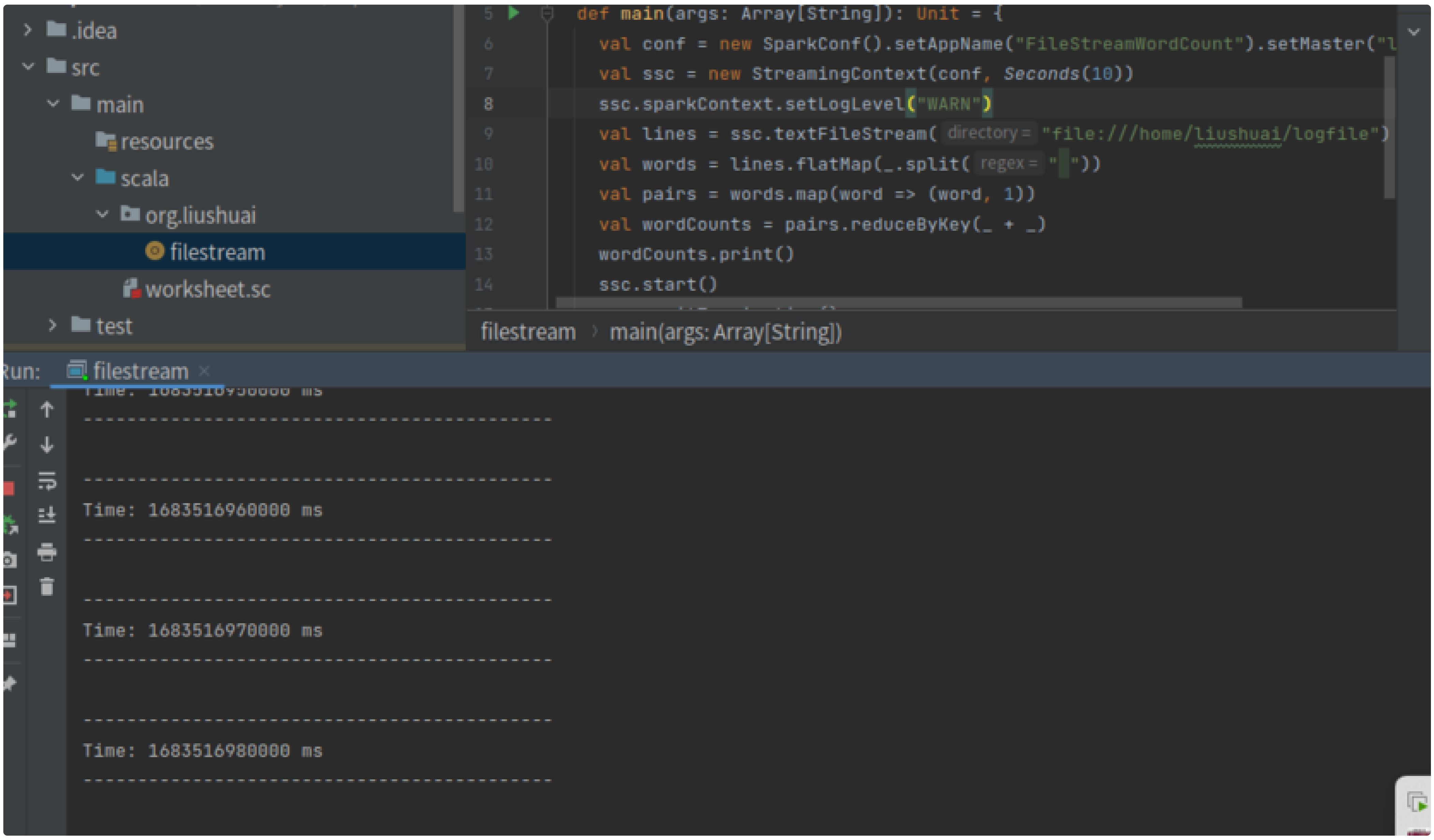
1.1.2.5 添加 `scala library` 库，打开 `project setting/library` 添加 `scala library`



1.1.2.6 编写 `FileStream.scala` 文件并且 compile 整个项目，安装项目的环境依赖。

代码内容为：新建一个 SparkConf 对象，命名为 FileStreamWordCount，并且使用本地 localhost 打开（伪分布，然后新建实时数据流 ssc，并且设置检查时间为 10s。设置应用程序的日志级别为 `WARN` 。定义 line 文件流，检测这个文件夹中的输入的文件，对输入的文件中内容进行词频统计，然后打印单词：频数键值对。

```Kotlin
package org.liushuai
import org.apache.spark.streaming._
import org.apache.spark.SparkConf
object filestream {
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setAppName("FileStreamWordCount").setMaster("local[*]")
    val ssc = new StreamingContext(conf, Seconds(10))
    ssc.sparkContext.setLogLevel("WARN")
    val lines = ssc.textFileStream("file:///home/liushuai/logfile")
    val words = lines.flatMap(_.split(" "))
    val pairs = words.map(word => (word, 1))
    val wordCounts = pairs.reduceByKey(_ + _)
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```

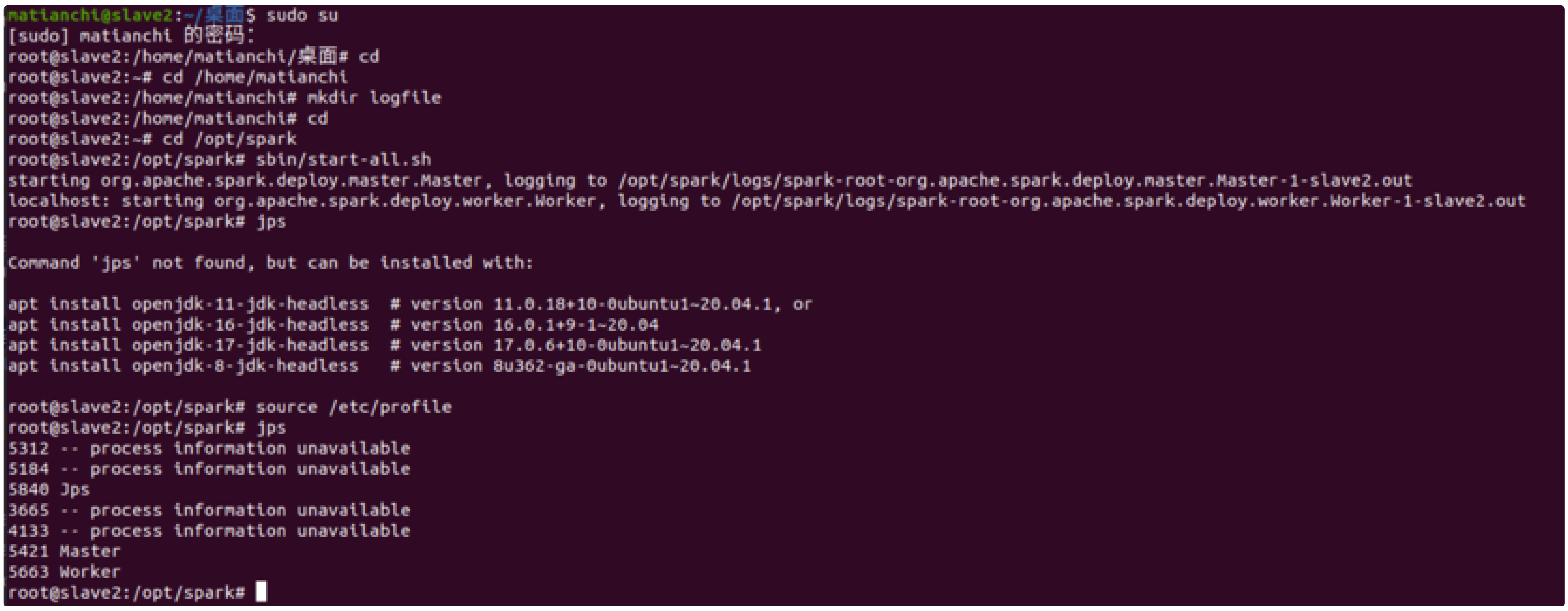


```
def main(args: Array[String]): Unit = {
  val conf = new SparkConf().setAppName("FileStreamWordCount").setMaster("l
  val ssc = new StreamingContext(conf, Seconds(10))
  ssc.sparkContext.setLogLevel("WARN")
  val lines = ssc.textFileStream( directory = "file:///home/liushuai/logfile")
  val words = lines.flatMap(_.split( regex = " "))
  val pairs = words.map(word => (word, 1))
  val wordCounts = pairs.reduceByKey(_ + _)
  wordCounts.print()
  ssc.start()
```

```
Time: 1683516950000 ms
-------------------------------------------


-------------------------------------------
Time: 1683516960000 ms
-------------------------------------------


-------------------------------------------
Time: 1683516970000 ms
-------------------------------------------


-------------------------------------------
Time: 1683516980000 ms
-------------------------------------------
```

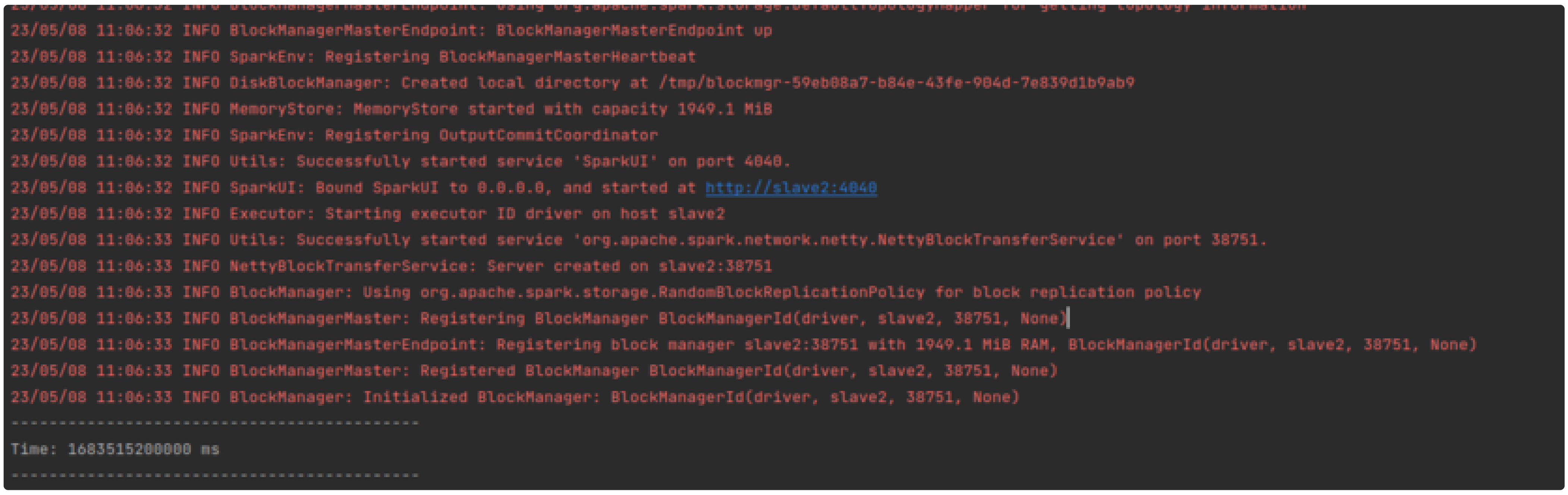
1.1.3 运行测试

1.1.3.1 启动 spark



```
matianchi@slave2:~/桌面$ sudo su
[sudo] matianchi 的密码:
root@slave2:/home/matianchi/桌面# cd
root@slave2:~# cd /home/matianchi
root@slave2:/home/matianchi# mkdir logfile
root@slave2:/home/matianchi# cd
root@slave2:~# cd /opt/spark
root@slave2:/opt/spark# sbin/start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark-root-org.apache.spark.deploy.master.Master-1-slave2.out
localhost: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-slave2.out
root@slave2:/opt/spark# jps

Command 'jps' not found, but can be installed with:

apt install openjdk-11-jdk-headless  # version 11.0.18+10-0ubuntu1~20.04.1, or
apt install openjdk-16-jdk-headless  # version 16.0.1+9-1~20.04
apt install openjdk-17-jdk-headless  # version 17.0.6+10-0ubuntu1~20.04.1
apt install openjdk-8-jdk-headless   # version 8u362-ga-0ubuntu1~20.04.1

root@slave2:/opt/spark# source /etc/profile
root@slave2:/opt/spark# jps
5312 -- process information unavailable
5184 -- process information unavailable
5840 Jps
3665 -- process information unavailable
4133 -- process information unavailable
5421 Master
5663 Worker
root@slave2:/opt/spark# 
```

1.1.3.2 启动程序，可以在 idea 的控制台中看到程序的输出。由于目前文件夹中没有文件，所以没有 wordcount 输出。



```
23/05/08 11:06:32 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
23/05/08 11:06:32 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
23/05/08 11:06:32 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
23/05/08 11:06:32 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-59eb08a7-b84e-43fe-904d-7e839d1b9ab9
23/05/08 11:06:32 INFO MemoryStore: MemoryStore started with capacity 1949.1 MiB
23/05/08 11:06:32 INFO SparkEnv: Registering OutputCommitCoordinator
23/05/08 11:06:32 INFO Utils: Successfully started service 'SparkUI' on port 4040.
23/05/08 11:06:32 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://slave2:4040
23/05/08 11:06:32 INFO Executor: Starting executor ID driver on host slave2
23/05/08 11:06:33 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 38751.
23/05/08 11:06:33 INFO NettyBlockTransferService: Server created on slave2:38751
23/05/08 11:06:33 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
23/05/08 11:06:33 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, slave2, 38751, None)
23/05/08 11:06:33 INFO BlockManagerMasterEndpoint: Registering block manager slave2:38751 with 1949.1 MiB RAM, BlockManagerId(driver, slave2, 38751, None)
23/05/08 11:06:33 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, slave2, 38751, None)
23/05/08 11:06:33 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, slave2, 38751, None)
-------------------------------------------
Time: 1683515200000 ms
-------------------------------------------
```

创建一个输入 DStream，使用 textFileStream()方法从指定的文件路径读取文本文件。textFileStream()方法会自动监视指定路径下的新文件，并将它们转换为 DStream。

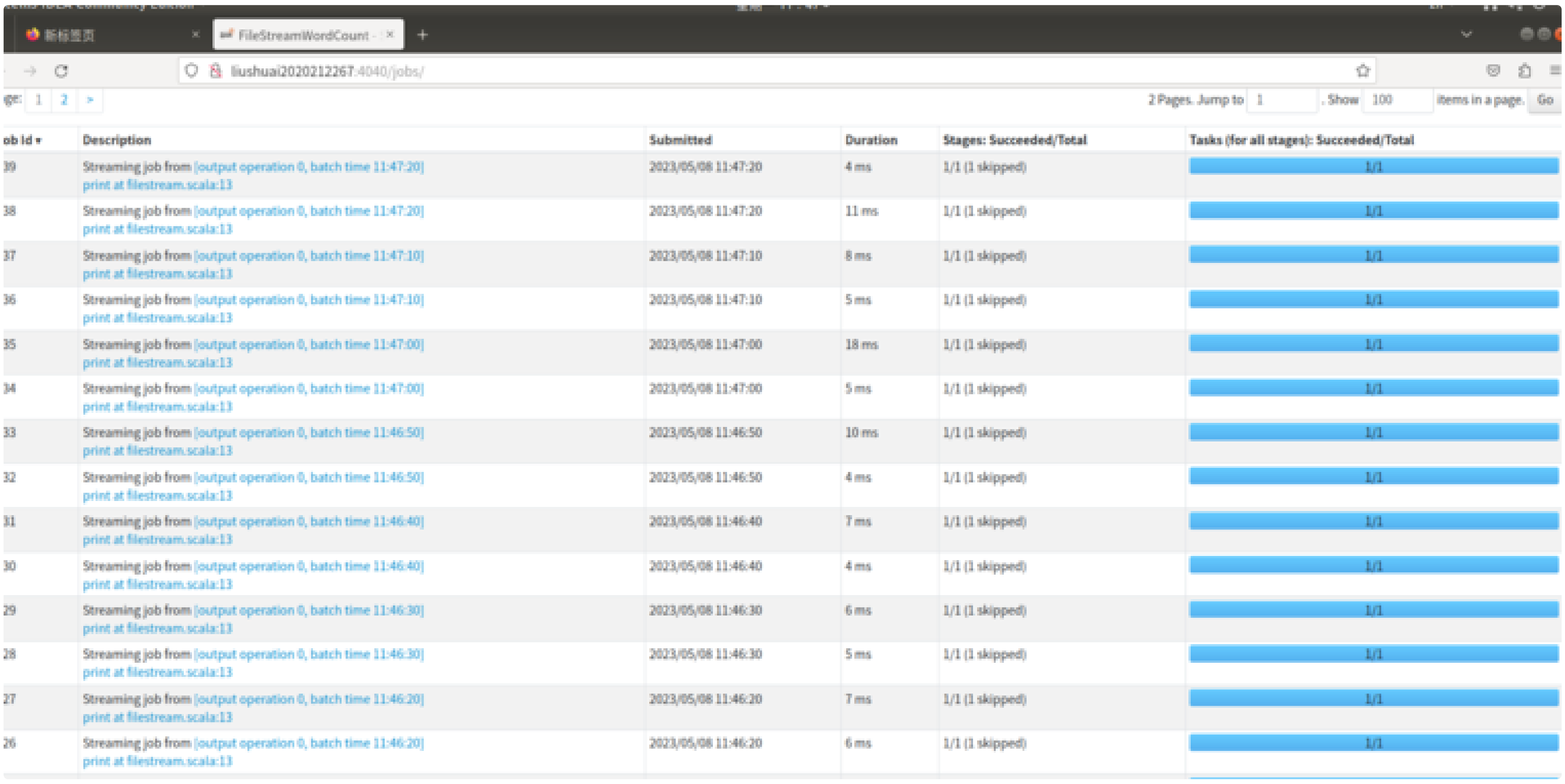
对输入 DStream 进行转换，使用 flatMap()方法将每一行拆分成单词，使用 map()方法将每个单词映射为一个键值对（单词，1）。

对键值对 DStream 应用 reduceByKey()方法进行聚合操作，以计算每个单词出现的次数

对最终的 DStream 应用 print()方法，将每个批处理间隔计算的结果输出到控制台。

启动 StreamingContext 并等待作业完成。

1.1.3.3 `http://localhost:4040/job` 网站截图



ge: 1 2 >   2 Pages. Jump to 1 . Show 100 items in a page. Go

ob Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
39	Streaming job from [output operation 0, batch time 11:47:20] print at filestream.scala:13	2023/05/08 11:47:20	4 ms	1/1 (1 skipped)	1/1
38	Streaming job from [output operation 0, batch time 11:47:20] print at filestream.scala:13	2023/05/08 11:47:20	11 ms	1/1 (1 skipped)	1/1
37	Streaming job from [output operation 0, batch time 11:47:10] print at filestream.scala:13	2023/05/08 11:47:10	8 ms	1/1 (1 skipped)	1/1
36	Streaming job from [output operation 0, batch time 11:47:10] print at filestream.scala:13	2023/05/08 11:47:10	5 ms	1/1 (1 skipped)	1/1
35	Streaming job from [output operation 0, batch time 11:47:00] print at filestream.scala:13	2023/05/08 11:47:00	18 ms	1/1 (1 skipped)	1/1
34	Streaming job from [output operation 0, batch time 11:47:00] print at filestream.scala:13	2023/05/08 11:47:00	5 ms	1/1 (1 skipped)	1/1
33	Streaming job from [output operation 0, batch time 11:46:50] print at filestream.scala:13	2023/05/08 11:46:50	10 ms	1/1 (1 skipped)	1/1
32	Streaming job from [output operation 0, batch time 11:46:50] print at filestream.scala:13	2023/05/08 11:46:50	4 ms	1/1 (1 skipped)	1/1
31	Streaming job from [output operation 0, batch time 11:46:40] print at filestream.scala:13	2023/05/08 11:46:40	7 ms	1/1 (1 skipped)	1/1
30	Streaming job from [output operation 0, batch time 11:46:40] print at filestream.scala:13	2023/05/08 11:46:40	4 ms	1/1 (1 skipped)	1/1
29	Streaming job from [output operation 0, batch time 11:46:30] print at filestream.scala:13	2023/05/08 11:46:30	6 ms	1/1 (1 skipped)	1/1
28	Streaming job from [output operation 0, batch time 11:46:30] print at filestream.scala:13	2023/05/08 11:46:30	5 ms	1/1 (1 skipped)	1/1
27	Streaming job from [output operation 0, batch time 11:46:20] print at filestream.scala:13	2023/05/08 11:46:20	7 ms	1/1 (1 skipped)	1/1
26	Streaming job from [output operation 0, batch time 11:46:20] print at filestream.scala:13	2023/05/08 11:46:20	6 ms	1/1 (1 skipped)	1/1

写入 socketstream 类，绑定 9999 端口，用于监听 netcat 的输入结果

```Kotlin
package org.liushuai
import org.apache.spark._
import org.apache.spark.streaming._
object socketstream {
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setMaster("local[2]").setAppName("NetworkWordCount")
    val ssc = new StreamingContext(conf, Seconds(5))
    val lines = ssc.socketTextStream("localhost", 9999)
    val words = lines.flatMap(_.split(" "))
    val pairs = words.map(word => (word, 1))
    val wordCounts = pairs.reduceByKey(_ + _)
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```

执行 socektstream 类

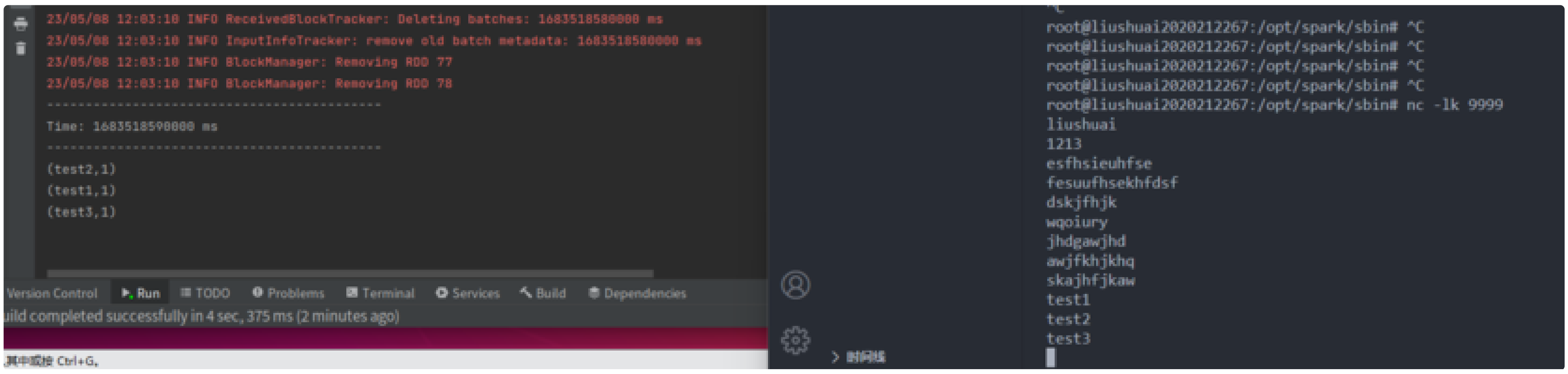
输入脚本

```Kotlin
nc -lk 9999
```

启动 netcat 在 TCP/UDP 协议上进行数据传输。

- `-l`：表示 nc 命令启动一个监听模式的服务器，等待连接请求。
- `-k`：表示在一个客户端断开连接后不要退出 nc 命令，而是继续监听等待新的连接请求。

在下方输入文本，并在 idea 中进行监听，



```
23/05/08 12:03:10 INFO ReceivedBlockTracker: Deleting batches: 1683518580000 ms
23/05/08 12:03:10 INFO InputInfoTracker: remove old batch metadata: 1683518580000 ms
23/05/08 12:03:10 INFO BlockManager: Removing RDD 77
23/05/08 12:03:10 INFO BlockManager: Removing RDD 78
-------------------------------------------
Time: 1683518590000 ms
-------------------------------------------
(test2,1)
(test1,1)
(test3,1)
```

uild completed successfully in 4 sec, 375 ms (2 minutes ago)

```
root@liushuai2020212267:/opt/spark/sbin# ^C
root@liushuai2020212267:/opt/spark/sbin# ^C
root@liushuai2020212267:/opt/spark/sbin# ^C
root@liushuai2020212267:/opt/spark/sbin# ^C
root@liushuai2020212267:/opt/spark/sbin# nc -lk 9999
liushuai
1213
esfhsieuhfse
fesuufhsekhfdsf
dskjfhjk
wqoiury
jhdgawjhd
awjfkhjkhq
skajhfjkaw
test1
test2
test3
```

该程序创建 treamingContext 对象，设置批处理时间间隔为 5 秒，而后使用 socketTextStream()方法从指定的主机和 9999 端口读取输出流，socketTextStream()方法会自动监视该主机和端口的输入流，并将它们转换为 DStream。而后对输入的 DStream 进行转换，使用 flatMap()方法将每一行拆分成单词，使用 map()方法将每个单词映射为一个键值对（单词，1）。随后进行 reduceByKey 进行聚合操作，从而计算出单词出现的次数

1.2.3 启动 NetCat 并启动 SocketStream.scala 程序

编辑 socketstreamtofile 文件并执行，同时执行脚本

Kotlin

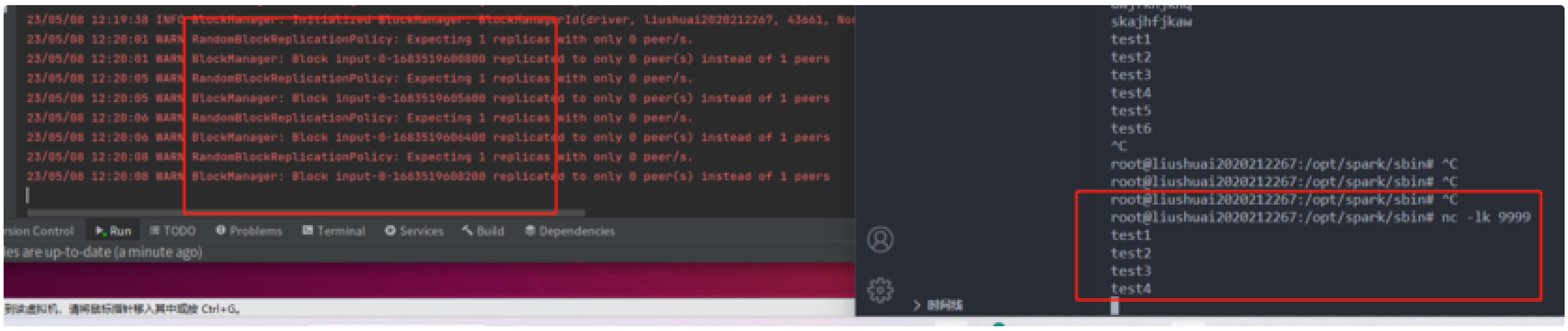
```
nc -lk 9999
```

向 9999 端口输入字节并进行监听。

Kotlin

```
package org.liushuai
import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StreamingContext}
object socketstreamtofile {
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setMaster("local[2]").setAppName("NetworkWordCount")
    val ssc = new StreamingContext(conf, Seconds(5))
    ssc.sparkContext.setLogLevel("WARN")
    val lines = ssc.socketTextStream("localhost", 9999)
    val words = lines.flatMap(_.split(" "))
    val pairs = words.map(word => (word, 1))
    val wordCounts = pairs.reduceByKey(_ + _)
    wordCounts.foreachRDD { rdd =>
      if (!rdd.isEmpty()) {
        val current = java.time.LocalDateTime.now.toString.replace(":", "-")
        rdd.sortByKey(ascending = true).map(x => x._1 + " " + x._2.toString)
          .saveAsTextFile(s"wordcount-$current")
      }
    }
    ssc.start()
    ssc.awaitTermination()
  }
}
```

在 netcat 的控制台中输入任意文字，工程目录下会新增对输入文本的 word count 结果

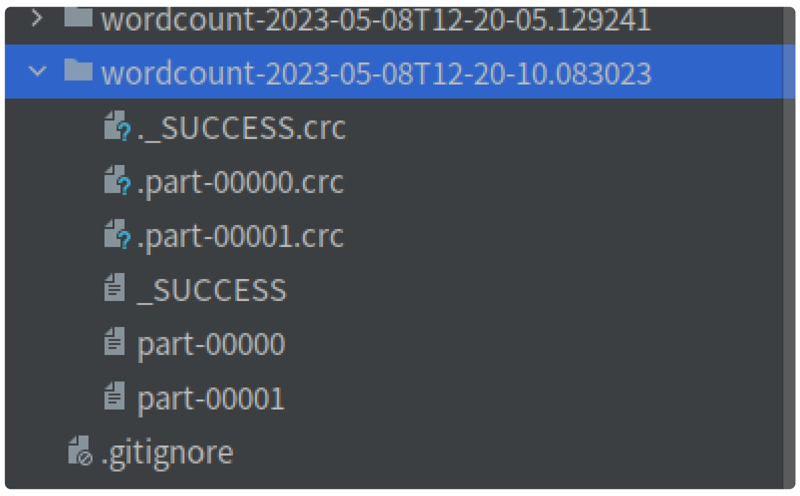


```
23/05/08 12:19:38 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, liushuai2020212267, 43661, No
23/05/08 12:20:01 WARN RandomBlockReplicationPolicy: Expecting 1 replicas with only 0 peer/s.
23/05/08 12:20:01 WARN BlockManager: Block input-0-1683519600800 replicated to only 0 peer(s) instead of 1 peers
23/05/08 12:20:05 WARN RandomBlockReplicationPolicy: Expecting 1 replicas with only 0 peer/s.
23/05/08 12:20:05 WARN BlockManager: Block input-0-1683519605600 replicated to only 0 peer(s) instead of 1 peers
23/05/08 12:20:06 WARN RandomBlockReplicationPolicy: Expecting 1 replicas with only 0 peer/s.
23/05/08 12:20:06 WARN BlockManager: Block input-0-1683519606400 replicated to only 0 peer(s) instead of 1 peers
23/05/08 12:20:08 WARN RandomBlockReplicationPolicy: Expecting 1 replicas with only 0 peer/s.
23/05/08 12:20:08 WARN BlockManager: Block input-0-1683519608200 replicated to only 0 peer(s) instead of 1 peers
```

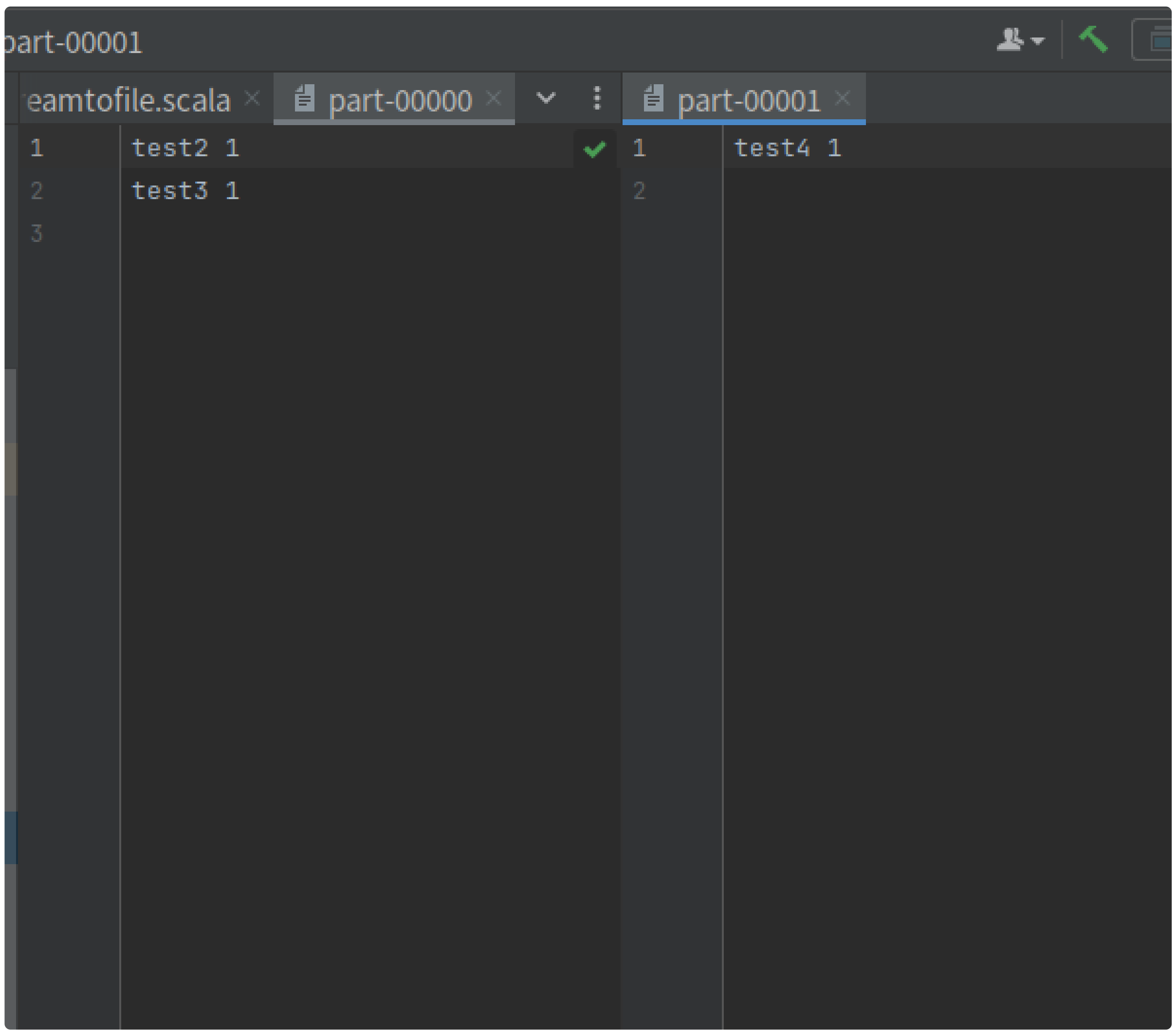
les are up-to-date (a minute ago)

```
skajhfjkaw
test1
test2
test3
test4
test5
test6
^C
root@liushuai2020212267:/opt/spark/sbin# ^C
root@liushuai2020212267:/opt/spark/sbin# ^C
root@liushuai2020212267:/opt/spark/sbin# ^C
root@liushuai2020212267:/opt/spark/sbin# nc -lk 9999
test1
test2
test3
test4
```

工程目录下会新增对输入文本的 word count 结果


```
wordcount-2023-05-08T12-20-05.129241
wordcount-2023-05-08T12-20-10.083023
    ._SUCCESS.crc
    .part-00000.crc
    .part-00001.crc
    _SUCCESS
    part-00000
    part-00001
  .gitignore
```

可见 word count 结果被保存到本地文件中了



```
test2 1
test3 1
```

```
test4 1
```

1.2 把处理结果保存到 MySQL 数据库中

1.2.1 安装并配置 MySQL

安装 mySQL

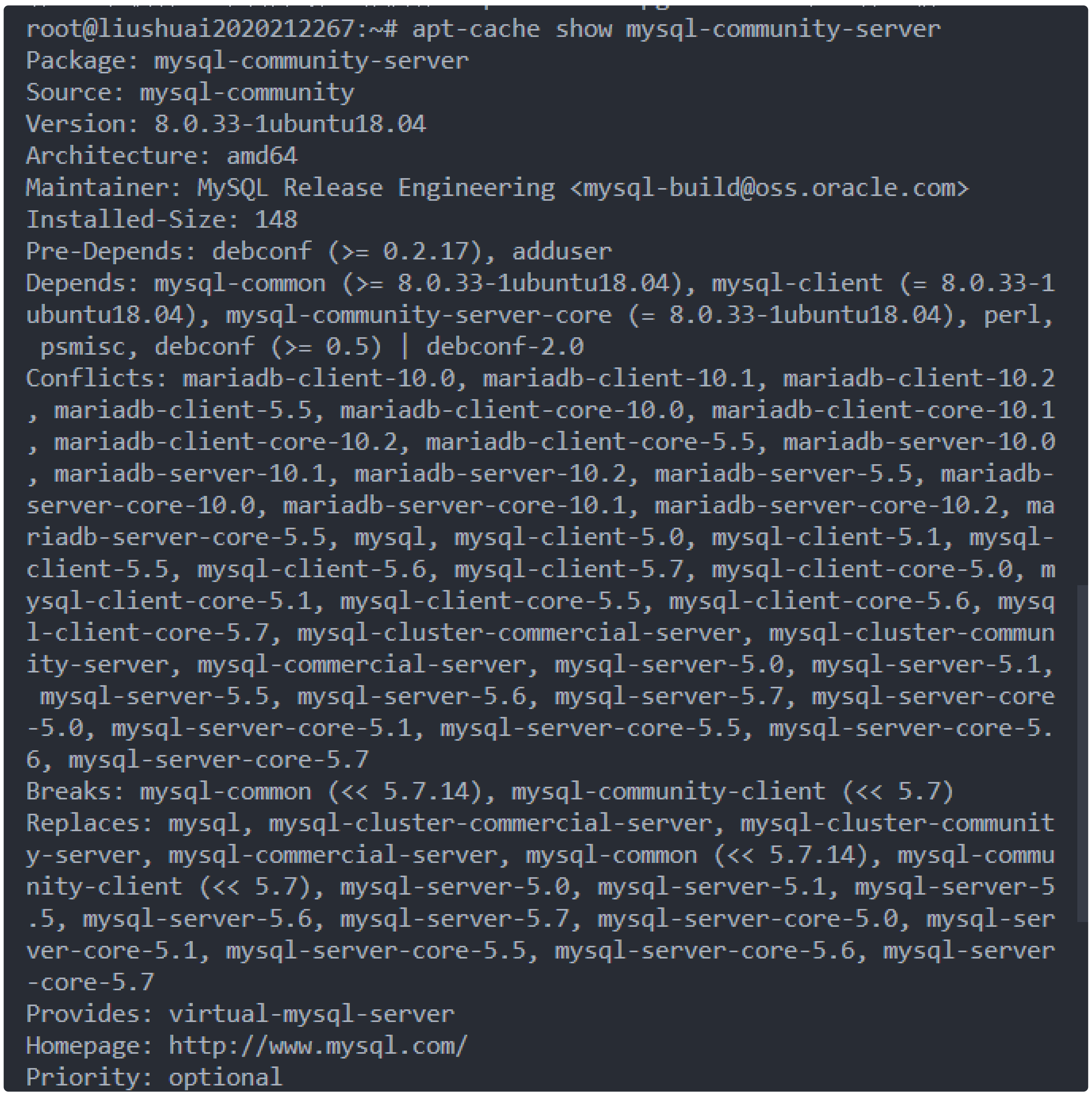
```Kotlin
wget https://dev.mysql.com/get/mysql-apt-config_0.8.25-1_all.deb
```

执行下列命令，添加 mysql 的原仓库并选择版本进行安装

```Kotlin
sudo dpkg -i mysql-apt-config_0.8.25-1_all.deb
```

查看 mysql-community-server 版本

```Kotlin
sudo apt-cache show mysql-community-server
```

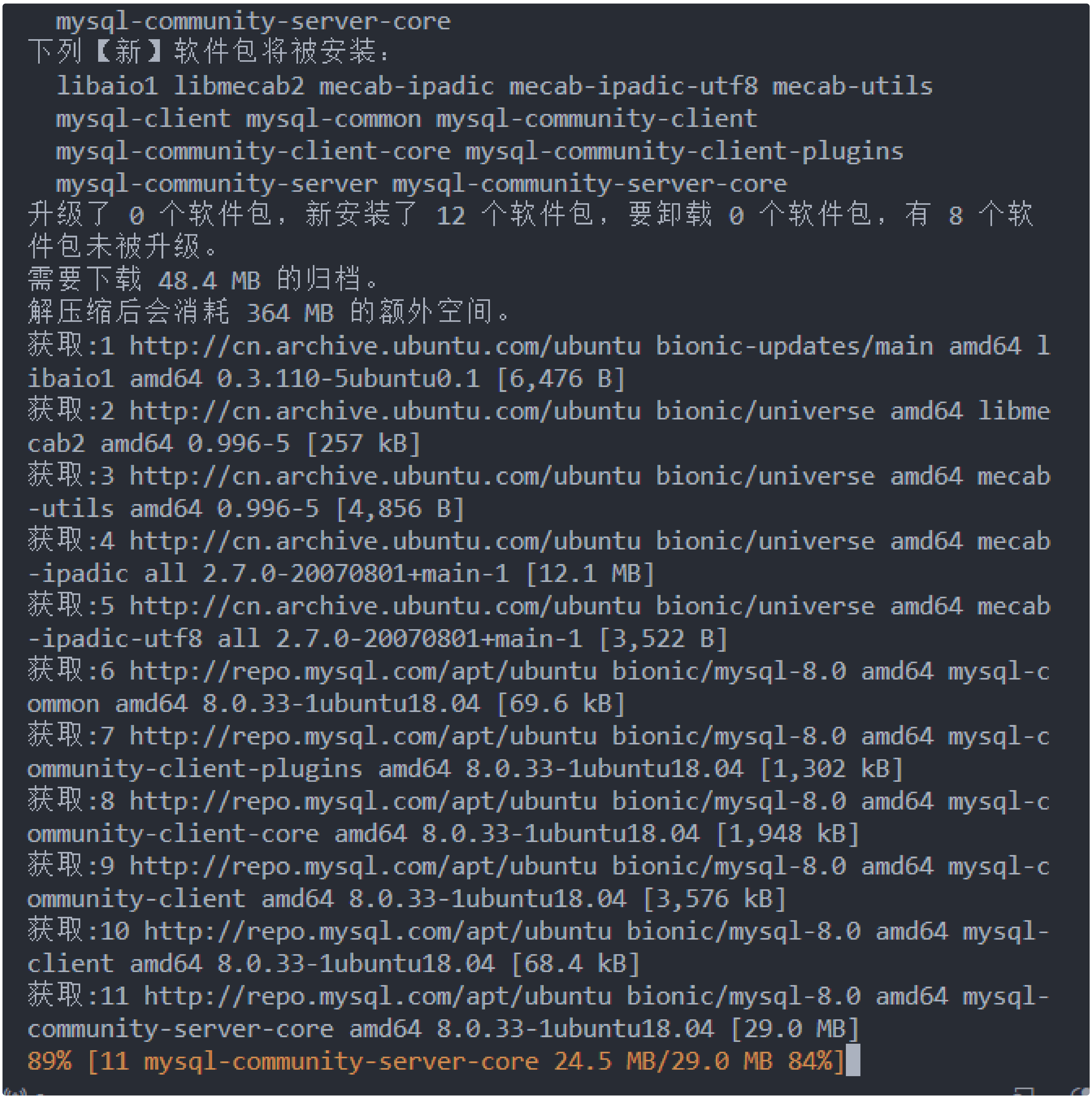


```
root@liushuai2020212267:~# apt-cache show mysql-community-server
Package: mysql-community-server
Source: mysql-community
Version: 8.0.33-1ubuntu18.04
Architecture: amd64
Maintainer: MySQL Release Engineering <mysql-build@oss.oracle.com>
Installed-Size: 148
Pre-Depends: debconf (>= 0.2.17), adduser
Depends: mysql-common (>= 8.0.33-1ubuntu18.04), mysql-client (= 8.0.33-1
ubuntu18.04), mysql-community-server-core (= 8.0.33-1ubuntu18.04), perl,
 psmisc, debconf (>= 0.5) | debconf-2.0
Conflicts: mariadb-client-10.0, mariadb-client-10.1, mariadb-client-10.2
, mariadb-client-5.5, mariadb-client-core-10.0, mariadb-client-core-10.1
, mariadb-client-core-10.2, mariadb-client-core-5.5, mariadb-server-10.0
, mariadb-server-10.1, mariadb-server-10.2, mariadb-server-5.5, mariadb-
server-core-10.0, mariadb-server-core-10.1, mariadb-server-core-10.2, ma
riadb-server-core-5.5, mysql, mysql-client-5.0, mysql-client-5.1, mysql-
client-5.5, mysql-client-5.6, mysql-client-5.7, mysql-client-core-5.0, m
ysql-client-core-5.1, mysql-client-core-5.5, mysql-client-core-5.6, mysq
l-client-core-5.7, mysql-cluster-commercial-server, mysql-cluster-commun
ity-server, mysql-commercial-server, mysql-server-5.0, mysql-server-5.1,
 mysql-server-5.5, mysql-server-5.6, mysql-server-5.7, mysql-server-core
-5.0, mysql-server-core-5.1, mysql-server-core-5.5, mysql-server-core-5.
6, mysql-server-core-5.7
Breaks: mysql-common (<< 5.7.14), mysql-community-client (<< 5.7)
Replaces: mysql, mysql-cluster-commercial-server, mysql-cluster-communit
y-server, mysql-commercial-server, mysql-common (<< 5.7.14), mysql-commu
nity-client (<< 5.7), mysql-server-5.0, mysql-server-5.1, mysql-server-5
.5, mysql-server-5.6, mysql-server-5.7, mysql-server-core-5.0, mysql-ser
ver-core-5.1, mysql-server-core-5.5, mysql-server-core-5.6, mysql-server
-core-5.7
Provides: virtual-mysql-server
Homepage: http://www.mysql.com/
Priority: optional
```

1.2.2 安装 mysql-community-server

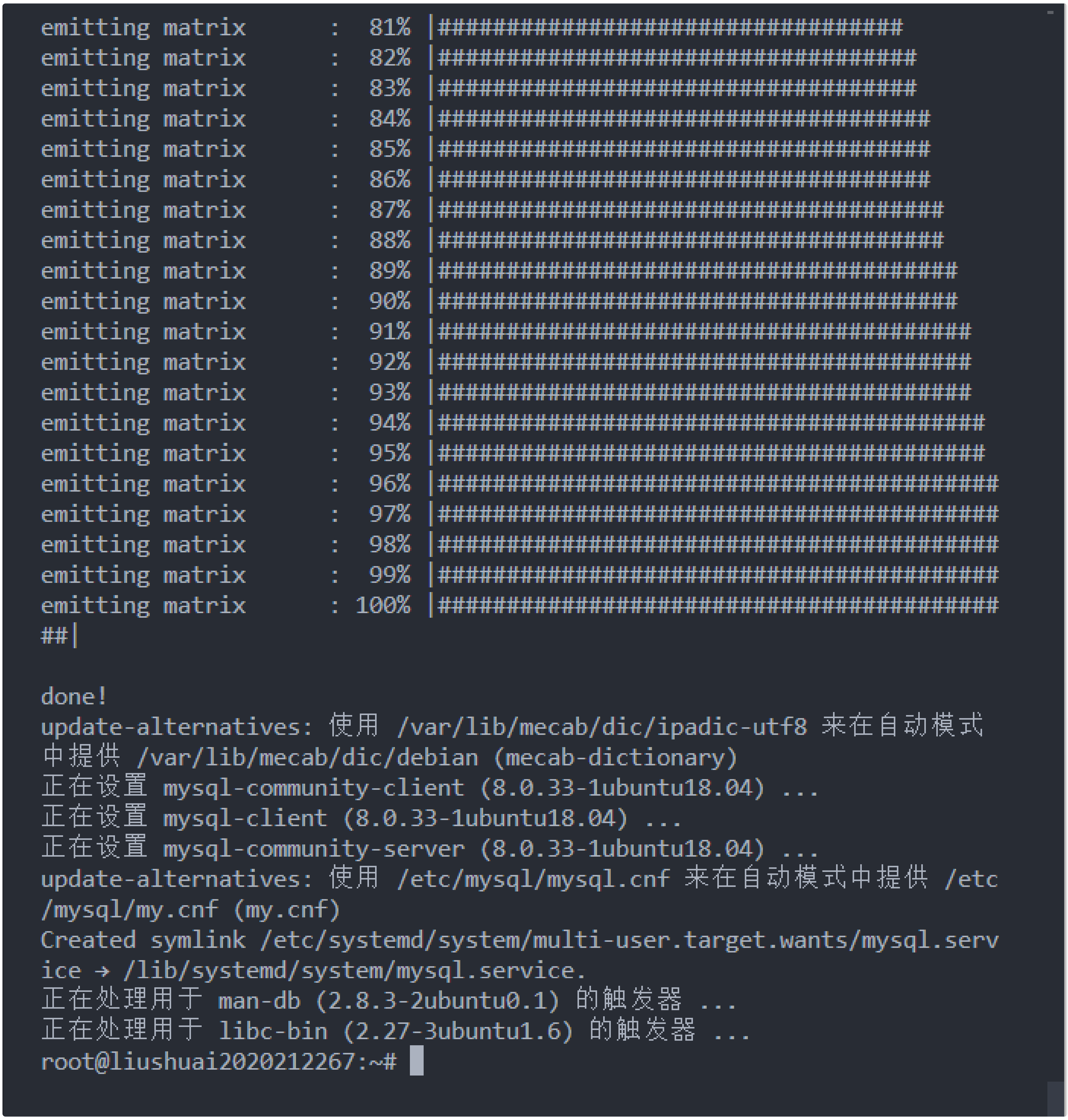
Kotlin

```
sudo apt install -y mysql-community-server
```



```
  mysql-community-server-core
下列【新】软件包将被安装：
  libaio1 libmecab2 mecab-ipadic mecab-ipadic-utf8 mecab-utils
  mysql-client mysql-common mysql-community-client
  mysql-community-client-core mysql-community-client-plugins
  mysql-community-server mysql-community-server-core
升级了 0 个软件包，新安装了 12 个软件包，要卸载 0 个软件包，有 8 个软
件包未被升级。
需要下载 48.4 MB 的归档。
解压缩后会消耗 364 MB 的额外空间。
获取:1 http://cn.archive.ubuntu.com/ubuntu bionic-updates/main amd64 l
ibaio1 amd64 0.3.110-5ubuntu0.1 [6,476 B]
获取:2 http://cn.archive.ubuntu.com/ubuntu bionic/universe amd64 libme
cab2 amd64 0.996-5 [257 kB]
获取:3 http://cn.archive.ubuntu.com/ubuntu bionic/universe amd64 mecab
-utils amd64 0.996-5 [4,856 B]
获取:4 http://cn.archive.ubuntu.com/ubuntu bionic/universe amd64 mecab
-ipadic all 2.7.0-20070801+main-1 [12.1 MB]
获取:5 http://cn.archive.ubuntu.com/ubuntu bionic/universe amd64 mecab
-ipadic-utf8 all 2.7.0-20070801+main-1 [3,522 B]
获取:6 http://repo.mysql.com/apt/ubuntu bionic/mysql-8.0 amd64 mysql-c
ommon amd64 8.0.33-1ubuntu18.04 [69.6 kB]
获取:7 http://repo.mysql.com/apt/ubuntu bionic/mysql-8.0 amd64 mysql-c
ommunity-client-plugins amd64 8.0.33-1ubuntu18.04 [1,302 kB]
获取:8 http://repo.mysql.com/apt/ubuntu bionic/mysql-8.0 amd64 mysql-c
ommunity-client-core amd64 8.0.33-1ubuntu18.04 [1,948 kB]
获取:9 http://repo.mysql.com/apt/ubuntu bionic/mysql-8.0 amd64 mysql-c
ommunity-client amd64 8.0.33-1ubuntu18.04 [3,576 kB]
获取:10 http://repo.mysql.com/apt/ubuntu bionic/mysql-8.0 amd64 mysql-
client amd64 8.0.33-1ubuntu18.04 [68.4 kB]
获取:11 http://repo.mysql.com/apt/ubuntu bionic/mysql-8.0 amd64 mysql-
community-server-core amd64 8.0.33-1ubuntu18.04 [29.0 MB]
89% [11 mysql-community-server-core 24.5 MB/29.0 MB 84%]
```

安装完成

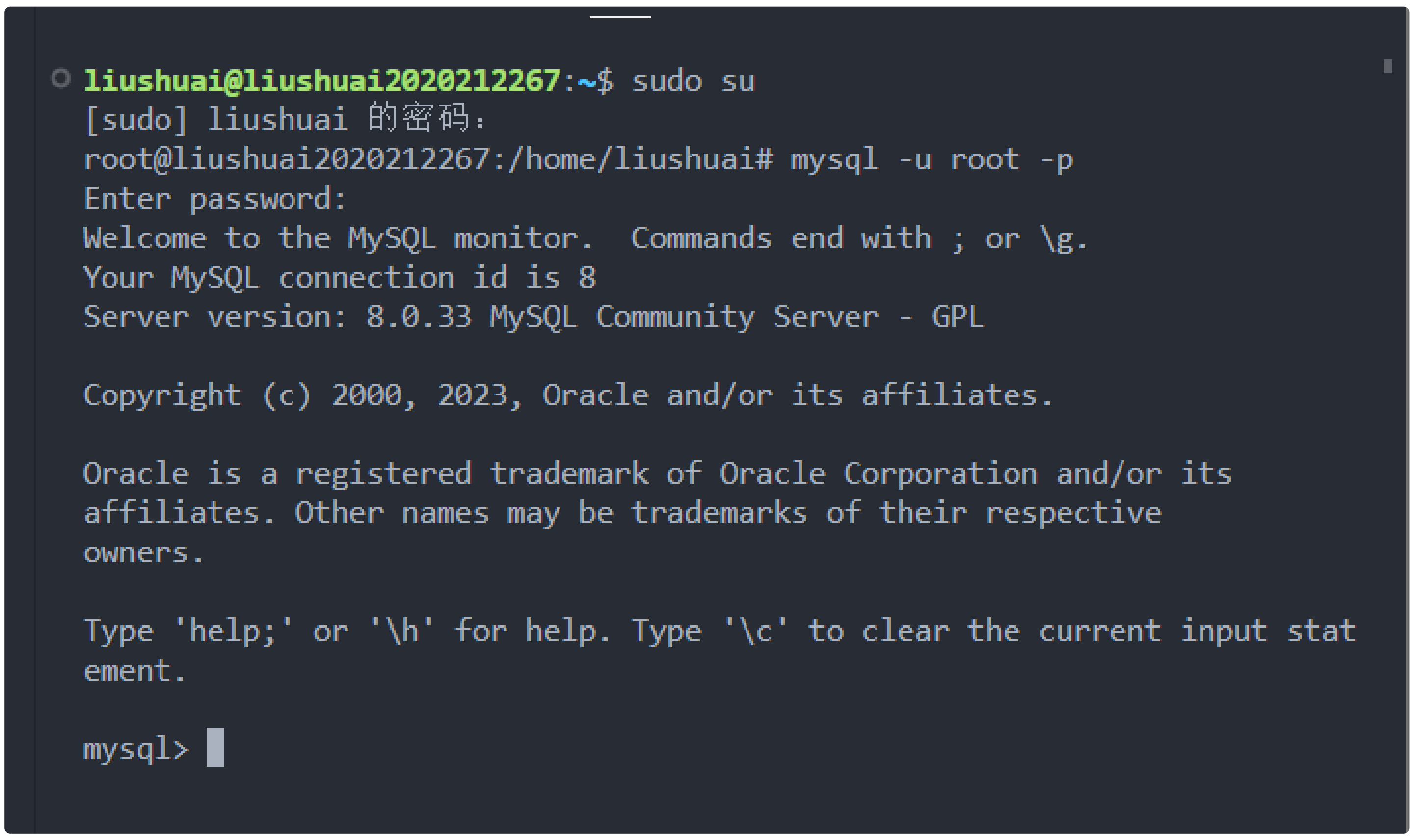


```
emitting matrix      :  81% |##################################
emitting matrix      :  82% |###################################
emitting matrix      :  83% |###################################
emitting matrix      :  84% |####################################
emitting matrix      :  85% |####################################
emitting matrix      :  86% |####################################
emitting matrix      :  87% |#####################################
emitting matrix      :  88% |#####################################
emitting matrix      :  89% |######################################
emitting matrix      :  90% |######################################
emitting matrix      :  91% |#######################################
emitting matrix      :  92% |#######################################
emitting matrix      :  93% |#######################################
emitting matrix      :  94% |########################################
emitting matrix      :  95% |########################################
emitting matrix      :  96% |#########################################
emitting matrix      :  97% |#########################################
emitting matrix      :  98% |#########################################
emitting matrix      :  99% |#########################################
emitting matrix      : 100% |#########################################
##|

done!
update-alternatives: 使用 /var/lib/mecab/dic/ipadic-utf8 来在自动模式
中提供 /var/lib/mecab/dic/debian (mecab-dictionary)
正在设置 mysql-community-client (8.0.33-1ubuntu18.04) ...
正在设置 mysql-client (8.0.33-1ubuntu18.04) ...
正在设置 mysql-community-server (8.0.33-1ubuntu18.04) ...
update-alternatives: 使用 /etc/mysql/mysql.cnf 来在自动模式中提供 /etc
/mysql/my.cnf (my.cnf)
Created symlink /etc/systemd/system/multi-user.target.wants/mysql.serv
ice → /lib/systemd/system/mysql.service.
正在处理用于 man-db (2.8.3-2ubuntu0.1) 的触发器 ...
正在处理用于 libc-bin (2.27-3ubuntu1.6) 的触发器 ...
root@liushuai2020212267:~# 
```

Kotlin

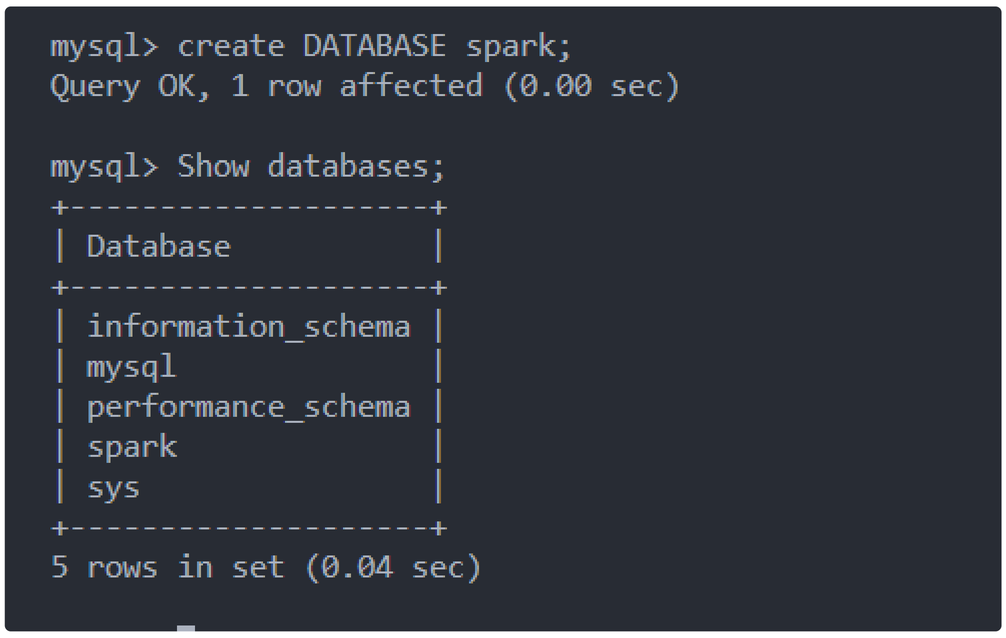
```
mysql -u root -p
```

1.2.3 输入以上指令登录 mysql



```
liushuai@liushuai2020212267:~$ sudo su
[sudo] liushuai 的密码：
root@liushuai2020212267:/home/liushuai# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.33 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input stat
ement.

mysql> 
```

1.2.4 创建一个名称为 spark 的数据库



```
mysql> create DATABASE spark;
Query OK, 1 row affected (0.00 sec)

mysql> Show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| spark              |
| sys                |
+--------------------+
5 rows in set (0.04 sec)
```

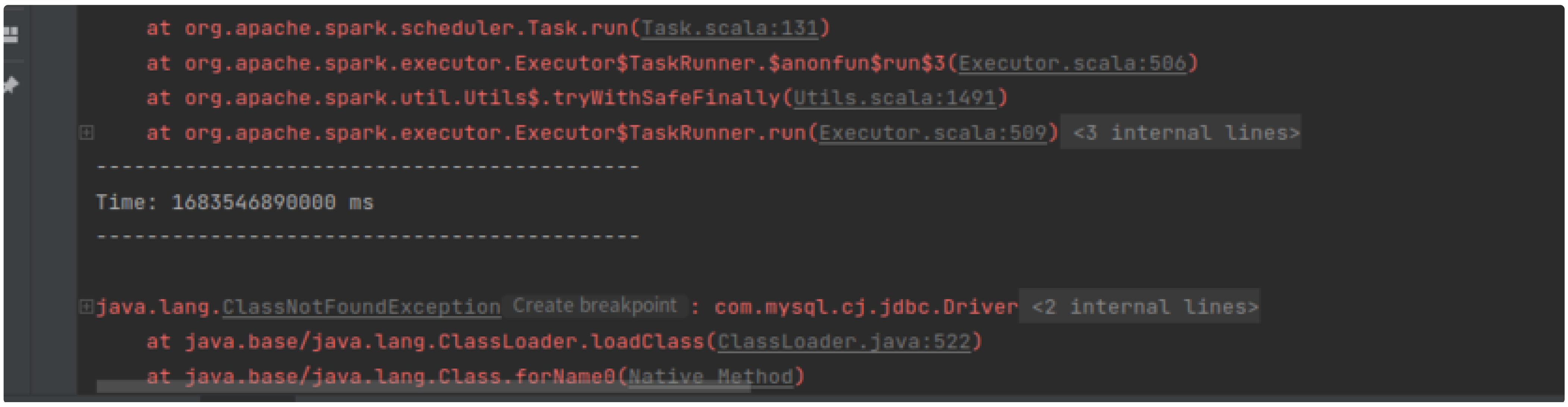
利用 spark 实现 FileStreamToMySQL
首先在 pom.xml 中添加

```Kotlin
<groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.33</version>
```

导入 mysql connector，其他 spark 以及 fstream 的依赖同上

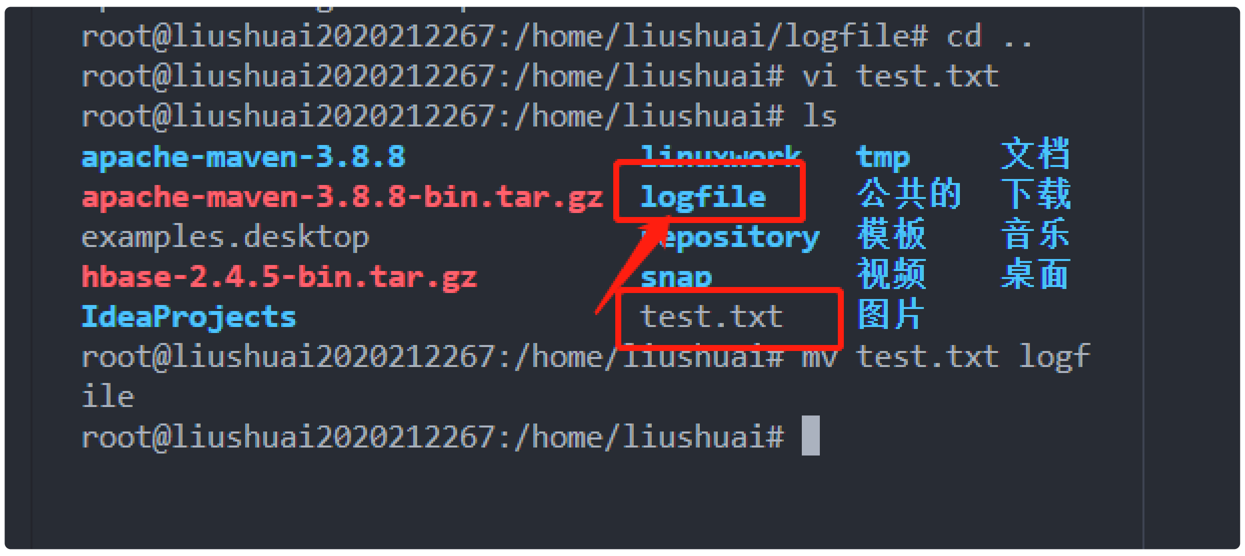
执行 FileStreamToMySQL 程序

```Kotlin
package org.liushuai
import org.apache.spark.streaming._
import org.apache.spark.SparkConf
import java.sql.{Connection, DriverManager, SQLException}
object FileStreamToMySQL {
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setAppName("FileStreamWordCountMySQL").setMaster("local[*]")
    val ssc = new StreamingContext(conf, Seconds(10))
    ssc.sparkContext.setLogLevel("WARN")
    val lines = ssc.textFileStream("file:///home/liushuai/logfile")
    val words = lines.flatMap(_.split(" "))
    val pairs = words.map(word => (word, 1))
    val wordCounts = pairs.reduceByKey(_ + _)
    wordCounts.print()
    wordCounts.foreachRDD(rdd => {
      rdd.foreachPartition(partition => {
        val url = "jdbc:mysql://localhost/spark"
        val driver = "com.mysql.cj.jdbc.Driver"
        val username = "root"
        val password = "DSAewq321"
        val tableName = "word_count"
        val wordColumn = "word"
        val countColumn = "occurrence"
        var connection: Connection = null
        try {
          Class.forName(driver)
          connection = DriverManager.getConnection(url, username, password)
          val statement = connection.createStatement()
          // 将数据写入表中
          partition.foreach(pair => {
            val word = pair._1
            val count = pair._2
            try {
              val update= "UPDATE %s SET %s = %s + %s WHERE %s ='%s'".format(tableName,countColumn,countColumn,count,wordColumn,word.replace("'", "''"))
              val affectedRows = statement.executeUpdate(update)
              if (affectedRows == 0) {
                val insert = "INSERT INTO %s (%s, %s) VALUES('% s', % s) ".format(tableName,wordColumn,countColumn,word.replace("'", "''"),count)
                statement.executeUpdate(insert)
              }
            } catch {
              case e: SQLException =>
                e.printStackTrace()
            }
          })
          statement.close()
        }
        catch {
          case e: Throwable => e.printStackTrace()
        }
        finally {
          if (connection != null) {
            connection.close()
          }
        }
      })
    })
    ssc.start()
    ssc.awaitTermination()
  }
}
```



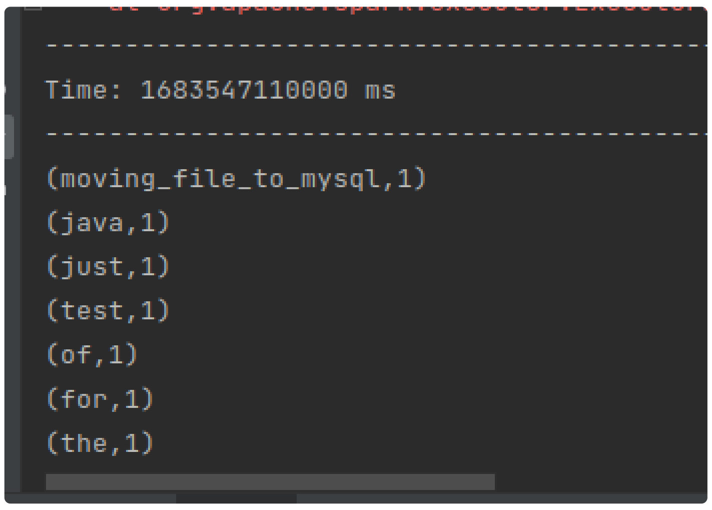
```
    at org.apache.spark.scheduler.Task.run(Task.scala:131)
    at org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$3(Executor.scala:506)
    at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1491)
    at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:509) <3 internal lines>
-------------------------------------------
Time: 1683546890000 ms
-------------------------------------------

java.lang.ClassNotFoundException Create breakpoint : com.mysql.cj.jdbc.Driver <2 internal lines>
    at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:522)
    at java.base/java.lang.Class.forName0(Native Method)
```

在 logfile 外部创建 test.txt 文件，并在运行 FileStreamToMySQL 的情况下将 txt 文件移入到 logfile 中



```
root@liushuai2020212267:/home/liushuai/logfile# cd ..
root@liushuai2020212267:/home/liushuai# vi test.txt
root@liushuai2020212267:/home/liushuai# ls
apache-maven-3.8.8               linuxwork   tmp     文档
apache-maven-3.8.8-bin.tar.gz    logfile     公共的  下载
examples.desktop                 repository  模板    音乐
hbase-2.4.5-bin.tar.gz           snap        视频    桌面
IdeaProjects                     test.txt    图片
root@liushuai2020212267:/home/liushuai# mv test.txt logf
ile
root@liushuai2020212267:/home/liushuai# 
```

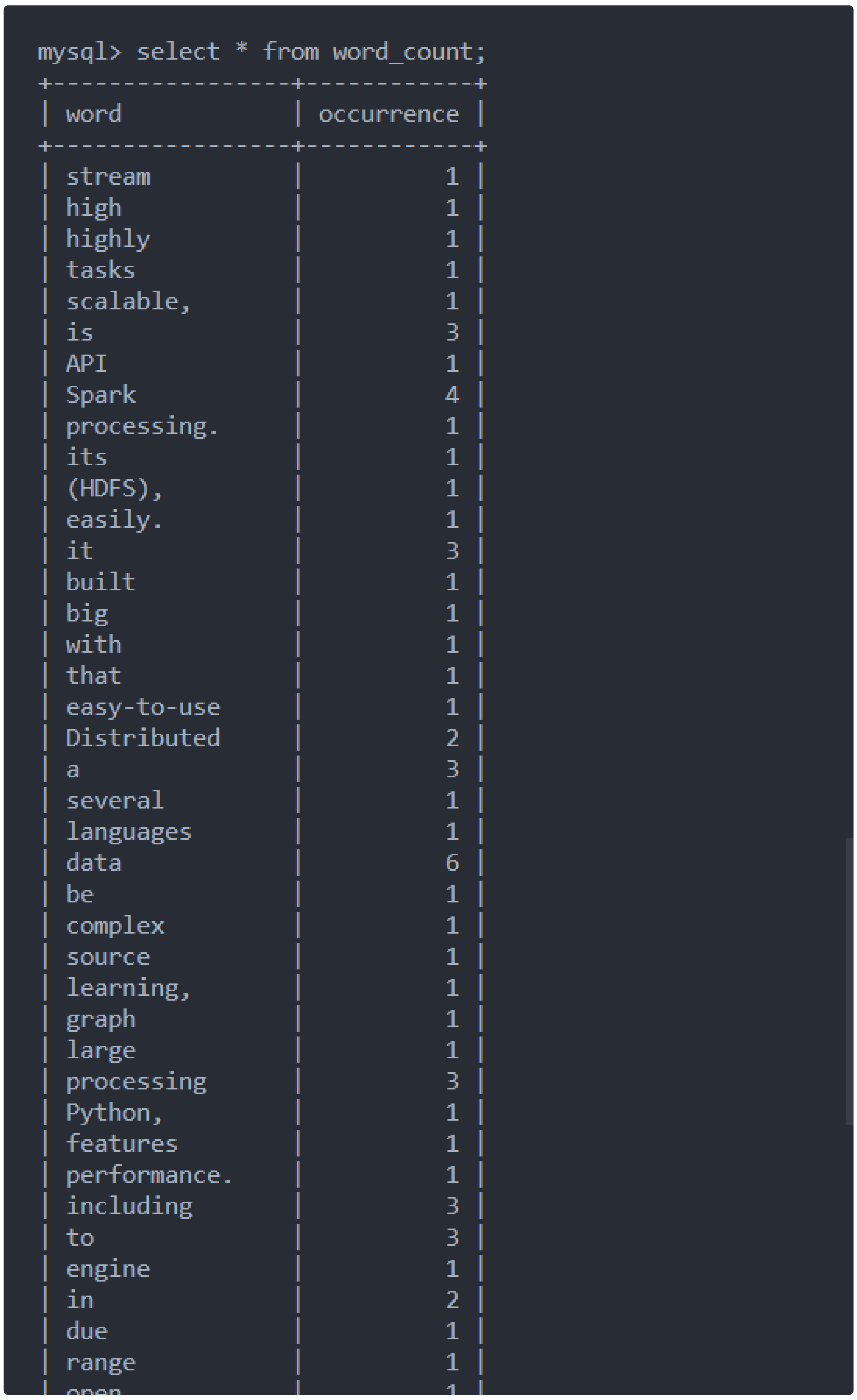
FileStreamToMySQL 的回显如下，证明此时已经进行了词频统计操作



```
-------------------------------------------
Time: 1683547110000 ms
-------------------------------------------
(moving_file_to_mysql,1)
(java,1)
(just,1)
(test,1)
(of,1)
(for,1)
(the,1)
```

```Kotlin
select * from word_count
```

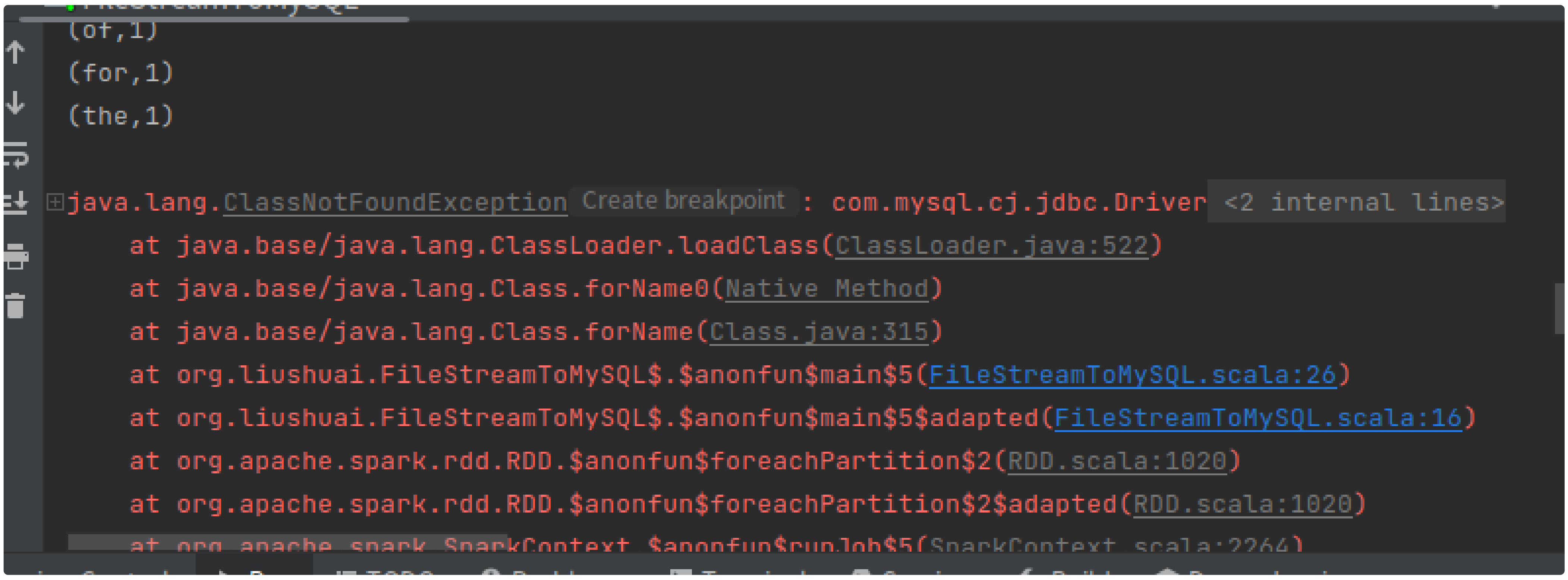
检查词频统计结果是否被存入到 mysql 中



```
mysql> select * from word_count;
+-----------------+------------+
| word            | occurrence |
+-----------------+------------+
| stream          |          1 |
| high            |          1 |
| highly          |          1 |
| tasks           |          1 |
| scalable,       |          1 |
| is              |          3 |
| API             |          1 |
| Spark           |          4 |
| processing.     |          1 |
| its             |          1 |
| (HDFS),         |          1 |
| easily.         |          1 |
| it              |          3 |
| built           |          1 |
| big             |          1 |
| with            |          1 |
| that            |          1 |
| easy-to-use     |          1 |
| Distributed     |          2 |
| a               |          3 |
| several         |          1 |
| languages       |          1 |
| data            |          6 |
| be              |          1 |
| complex         |          1 |
| source          |          1 |
| learning,       |          1 |
| graph           |          1 |
| large           |          1 |
| processing      |          3 |
| Python,         |          1 |
| features        |          1 |
| performance.    |          1 |
| including       |          3 |
| to              |          3 |
| engine          |          1 |
| in              |          2 |
| due             |          1 |
| range           |          1 |
| open            |          1 |
```

在实践过程中遇到的问题：

- 缺少 JDBC



```
FileStreamToMySQL
(of,1)
(for,1)
(the,1)

java.lang.ClassNotFoundException Create breakpoint : com.mysql.cj.jdbc.Driver <2 internal lines>
    at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:522)
    at java.base/java.lang.Class.forName0(Native Method)
    at java.base/java.lang.Class.forName(Class.java:315)
    at org.liushuai.FileStreamToMySQL$.$anonfun$main$5(FileStreamToMySQL.scala:26)
    at org.liushuai.FileStreamToMySQL$.$anonfun$main$5$adapted(FileStreamToMySQL.scala:16)
    at org.apache.spark.rdd.RDD.$anonfun$foreachPartition$2(RDD.scala:1020)
    at org.apache.spark.rdd.RDD.$anonfun$foreachPartition$2$adapted(RDD.scala:1020)
    at org.apache.spark.SparkContext.$anonfun$runJob$5(SparkContext.scala:2264)
```

在显示完词频过后，会发现以下数据并未存入到 mysql 当中，并返回未找到 jdbc.driver 的问题，该问题的原因是因为我们在 mvn install 后未将新添加的依赖

二、Spark Streaming 完全分布实现

2.1 完全分布下的 SparkStreaming 处理套接字

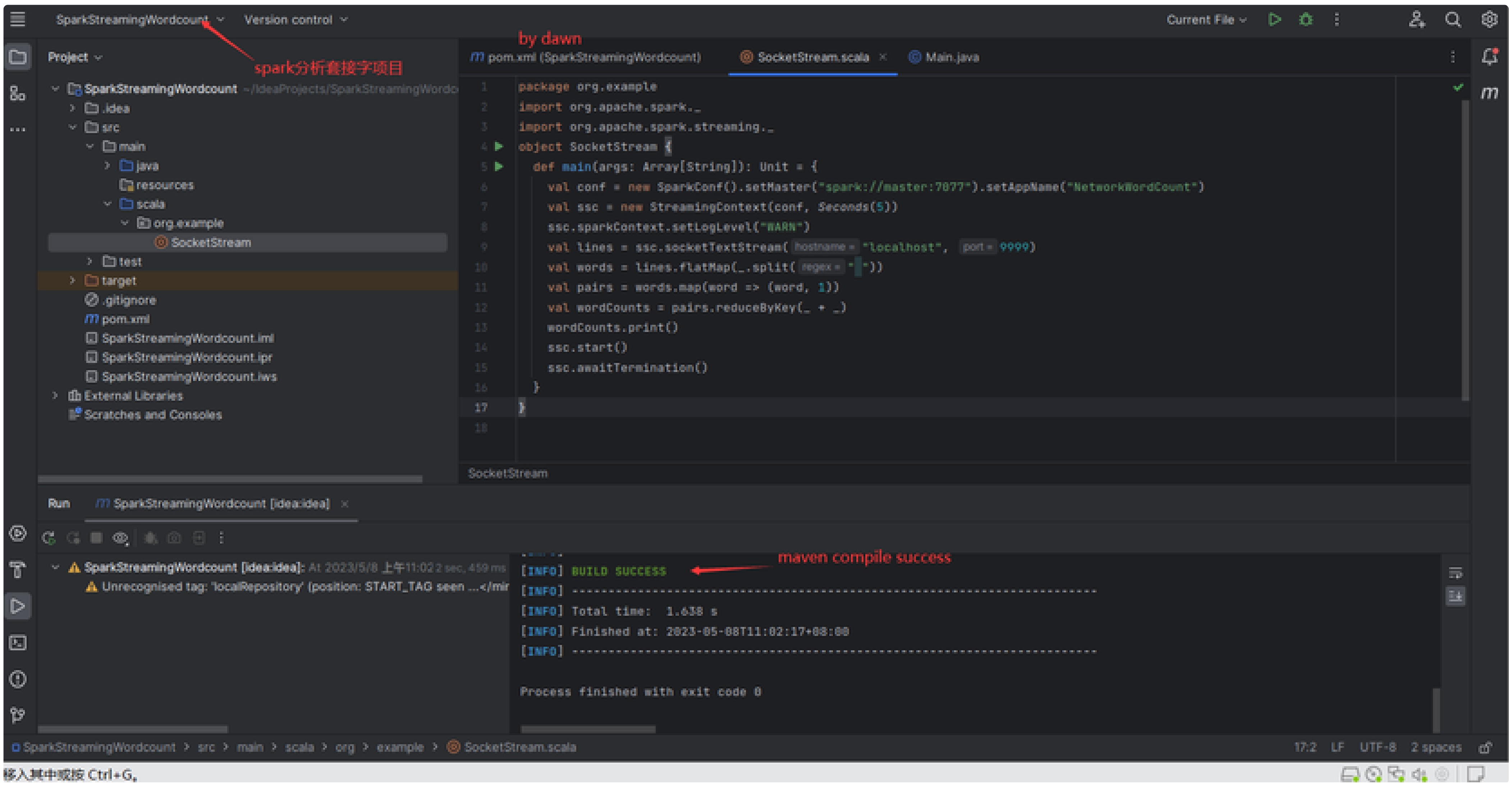
> 本次实验之前已基于个人热点局域网下桥接暴露 ip 配置好完全分布式集群 hadoop 和 spark 集群。

该测试的基本步骤是：在本地主机的 9999 端口上启动一个监听器，等待输入的数据流 命令行 。当有数据流输入时，nc 命令会将数据流传输到该端口，基于完全分布式集群的 Spark Streaming 应用程序将从该端口获取数据流并进行处理即记录日志。

Spark Streaming 可以通过 Socket 端口监听并接收数据，然后进行相应处理。

下面为实验过程记录包括说明：

1. 同实验三创建 `SparkStreamingWordcount` 项目，配置 maven 环境、scala 环境和 project `pom.xml` 文件，下载相关依赖包成功进行 `mvn compile`


by dawn
spark分析套接字项目
maven compile success
SparkStreamingWordcount
Version control
Current File
Project
pom.xml (SparkStreamingWordcount)
SocketStream.scala
Main.java
SparkStreamingWordcount ~/IdeaProjects/SparkStreamingWordco
.idea
src
main
java
resources
scala
org.example
SocketStream
test
target
.gitignore
pom.xml
SparkStreamingWordcount.iml
SparkStreamingWordcount.ipr
SparkStreamingWordcount.iws
External Libraries
Scratches and Consoles
Run
SparkStreamingWordcount [idea:idea]
SparkStreamingWordcount › src › main › scala › org › example › SocketStream.scala
17:2 LF UTF-8 2 spaces

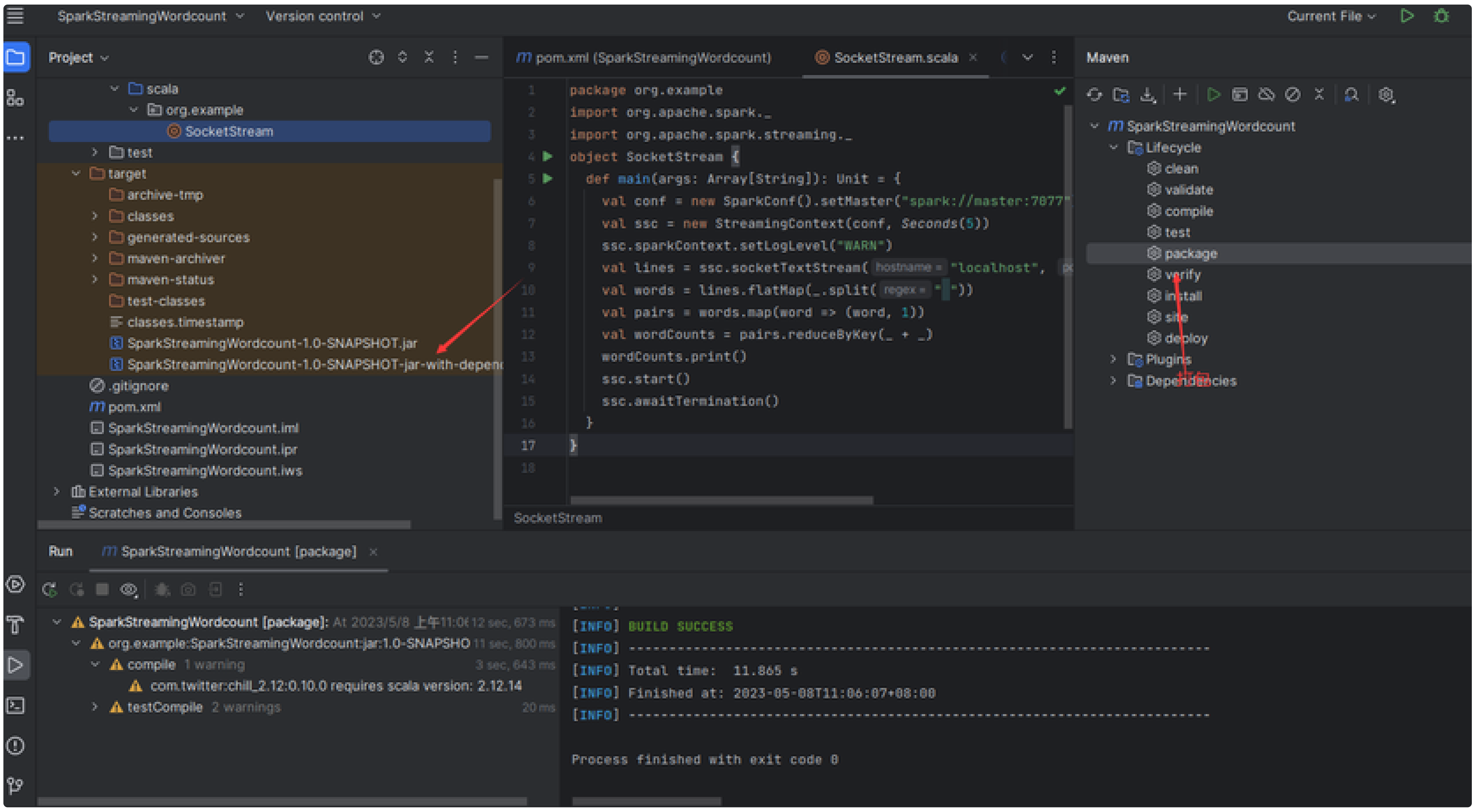
```scala
package org.example
import org.apache.spark._
import org.apache.spark.streaming._
object SocketStream {
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setMaster("spark://master:7077").setAppName("NetworkWordCount")
    val ssc = new StreamingContext(conf, Seconds(5))
    ssc.sparkContext.setLogLevel("WARN")
    val lines = ssc.socketTextStream( hostname = "localhost", port = 9999)
    val words = lines.flatMap(_.split( regex = " "))
    val pairs = words.map(word => (word, 1))
    val wordCounts = pairs.reduceByKey(_ + _)
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```

```
SparkStreamingWordcount [idea:idea]: At 2023/5/8 上午11:02 2 sec, 459 ms
  Unrecognised tag: 'localRepository' (position: START_TAG seen ...</mir
```

```
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.638 s
[INFO] Finished at: 2023-05-08T11:02:17+08:00
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

移入其中或按 Ctrl+G。

2. 复制示例代码文件并打包 spark streaming wordcount jar 项目：



```scala
package org.example
import org.apache.spark._
import org.apache.spark.streaming._
object SocketStream {
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setMaster("spark://master:7077"
    val ssc = new StreamingContext(conf, Seconds(5))
    ssc.sparkContext.setLogLevel("WARN")
    val lines = ssc.socketTextStream( hostname = "localhost",
    val words = lines.flatMap(_.split( regex = " "))
    val pairs = words.map(word => (word, 1))
    val wordCounts = pairs.reduceByKey(_ + _)
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```

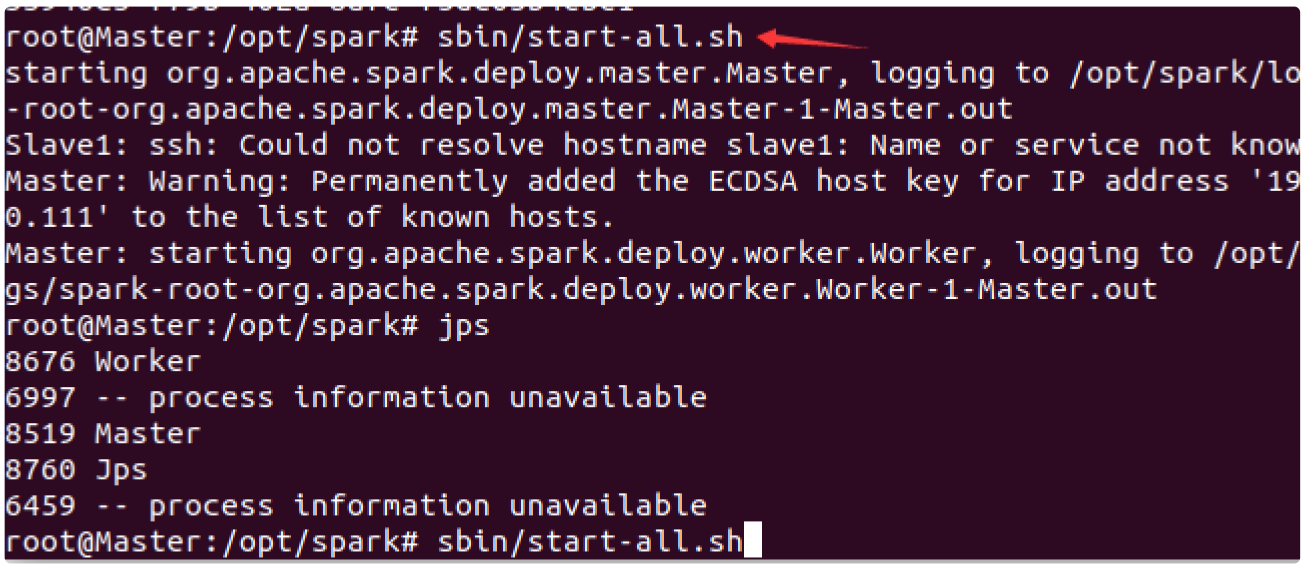
```
SparkStreamingWordcount [package]: At 2023/5/8 上午11:0( 12 sec, 673 ms
  org.example:SparkStreamingWordcount:jar:1.0-SNAPSHO 11 sec, 800 ms
    compile  1 warning                                 3 sec, 643 ms
      com.twitter:chill_2.12:0.10.0 requires scala version: 2.12.14
    testCompile  2 warnings                            20 ms
```

```
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  11.865 s
[INFO] Finished at: 2023-05-08T11:06:07+08:00
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

该 scala 示例代码文件的功能：创建一个 StreamingContext（Spark Streaming 应用程序的主入口点）并配置为每 5 秒接收一次本机端口为 `9999` 的数据流，并记录监听日志，后续将监听的任务可视化到前端界面 Master:4040 中 `4040应是Spark流处理配置文件默认的端口`。

3. 将带 depencies 的 jar 包移动到 `/opt/spark` 目录并重命名为 `SocketStream.jar`
4. 启动 Spark 集群并且向 Spark 提交 jar 包

- 启动 Spark 集群：

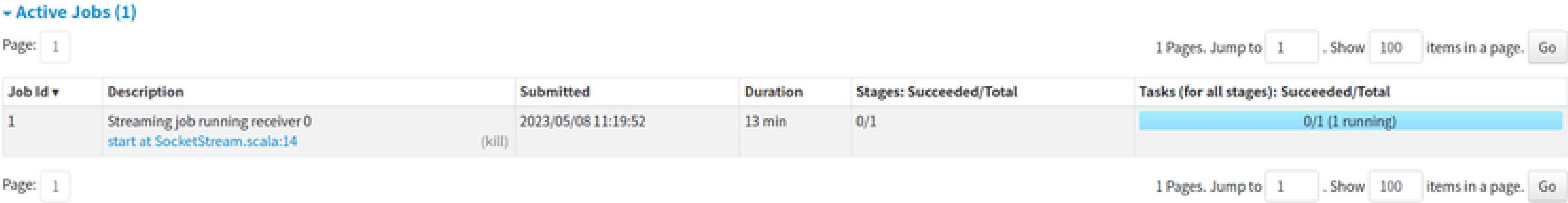


```
root@Master:/opt/spark# sbin/start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/lo
-root-org.apache.spark.deploy.master.Master-1-Master.out
Slave1: ssh: Could not resolve hostname slave1: Name or service not know
Master: Warning: Permanently added the ECDSA host key for IP address '19
0.111' to the list of known hosts.
Master: starting org.apache.spark.deploy.worker.Worker, logging to /opt/
gs/spark-root-org.apache.spark.deploy.worker.Worker-1-Master.out
root@Master:/opt/spark# jps
8676 Worker
6997 -- process information unavailable
8519 Master
8760 Jps
6459 -- process information unavailable
root@Master:/opt/spark# sbin/start-all.sh
```

- 在 `9999` 端口启动一个监听器
- 向 spark 提交 jar 包并在 Spark 集群上运行：

命令：spark-submit --class **你的主类** --master spark://**主节点** ip:7077 你的 jar 包路径

如图成功运行 SparkStreaming 运行程序 jar：



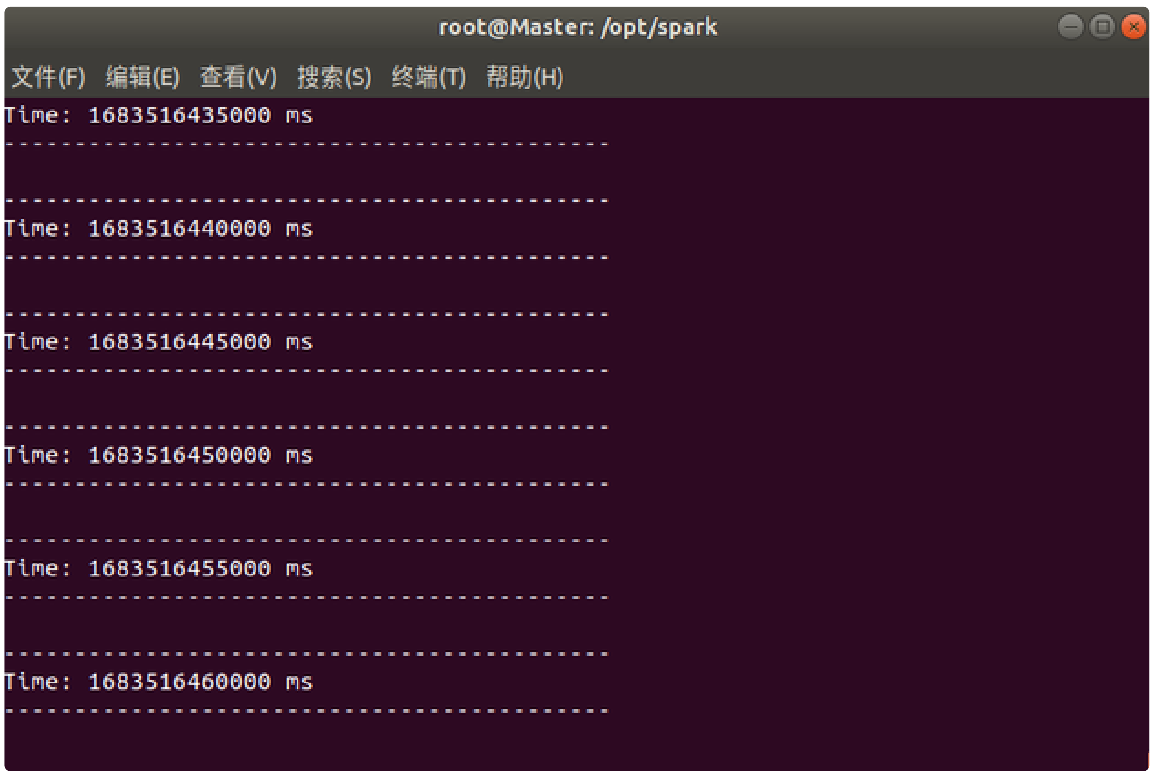
▾ Active Jobs (1)

Page: 1 　　　　1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	Streaming job running receiver 0 start at SocketStream.scala:14 (kill)	2023/05/08 11:19:52	13 min	0/1	0/1 (1 running)

Page: 1 　　　　1 Pages. Jump to 1 . Show 100 items in a page. Go

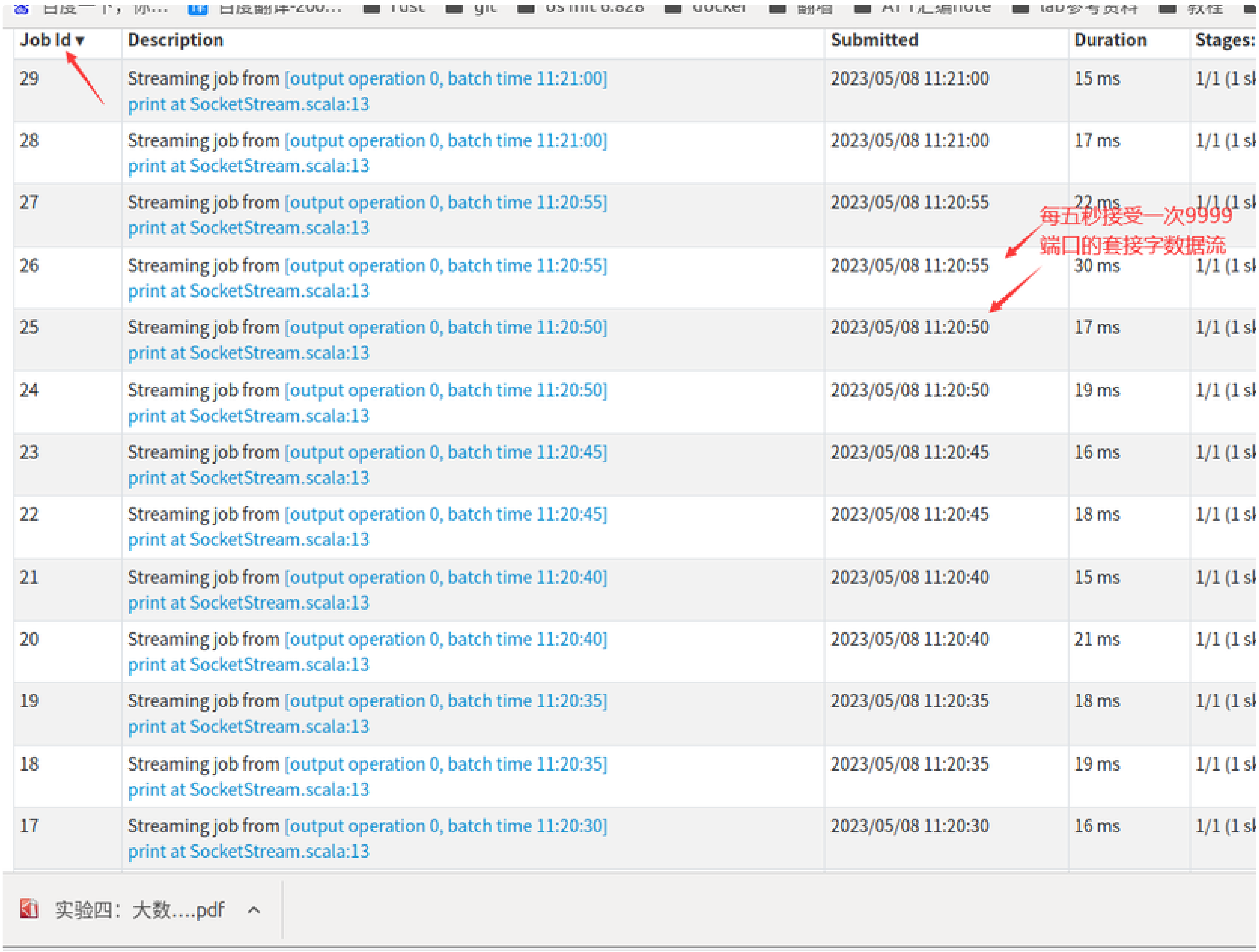
4040 端口 active jobs



```
Time: 1683516435000 ms
-------------------------------------------

-------------------------------------------
Time: 1683516440000 ms
-------------------------------------------

-------------------------------------------
Time: 1683516445000 ms
-------------------------------------------

-------------------------------------------
Time: 1683516450000 ms
-------------------------------------------

-------------------------------------------
Time: 1683516455000 ms
-------------------------------------------

-------------------------------------------
Time: 1683516460000 ms
-------------------------------------------
```

SparkStreaming monitoring

在前端界面 Master:4040 查看 sparkstreaming 正以 5 秒的时间间隔监听数据流：



Job Id ▾	Description	Submitted	Duration	Stages:
29	Streaming job from [output operation 0, batch time 11:21:00] print at SocketStream.scala:13	2023/05/08 11:21:00	15 ms	1/1 (1 sk
28	Streaming job from [output operation 0, batch time 11:21:00] print at SocketStream.scala:13	2023/05/08 11:21:00	17 ms	1/1 (1 sk
27	Streaming job from [output operation 0, batch time 11:20:55] print at SocketStream.scala:13	2023/05/08 11:20:55	22 ms	1/1 (1 sk
26	Streaming job from [output operation 0, batch time 11:20:55] print at SocketStream.scala:13	2023/05/08 11:20:55	30 ms	1/1 (1 sk
25	Streaming job from [output operation 0, batch time 11:20:50] print at SocketStream.scala:13	2023/05/08 11:20:50	17 ms	1/1 (1 sk
24	Streaming job from [output operation 0, batch time 11:20:50] print at SocketStream.scala:13	2023/05/08 11:20:50	19 ms	1/1 (1 sk
23	Streaming job from [output operation 0, batch time 11:20:45] print at SocketStream.scala:13	2023/05/08 11:20:45	16 ms	1/1 (1 sk
22	Streaming job from [output operation 0, batch time 11:20:45] print at SocketStream.scala:13	2023/05/08 11:20:45	18 ms	1/1 (1 sk
21	Streaming job from [output operation 0, batch time 11:20:40] print at SocketStream.scala:13	2023/05/08 11:20:40	15 ms	1/1 (1 sk
20	Streaming job from [output operation 0, batch time 11:20:40] print at SocketStream.scala:13	2023/05/08 11:20:40	21 ms	1/1 (1 sk
19	Streaming job from [output operation 0, batch time 11:20:35] print at SocketStream.scala:13	2023/05/08 11:20:35	18 ms	1/1 (1 sk
18	Streaming job from [output operation 0, batch time 11:20:35] print at SocketStream.scala:13	2023/05/08 11:20:35	19 ms	1/1 (1 sk
17	Streaming job from [output operation 0, batch time 11:20:30] print at SocketStream.scala:13	2023/05/08 11:20:30	16 ms	1/1 (1 sk

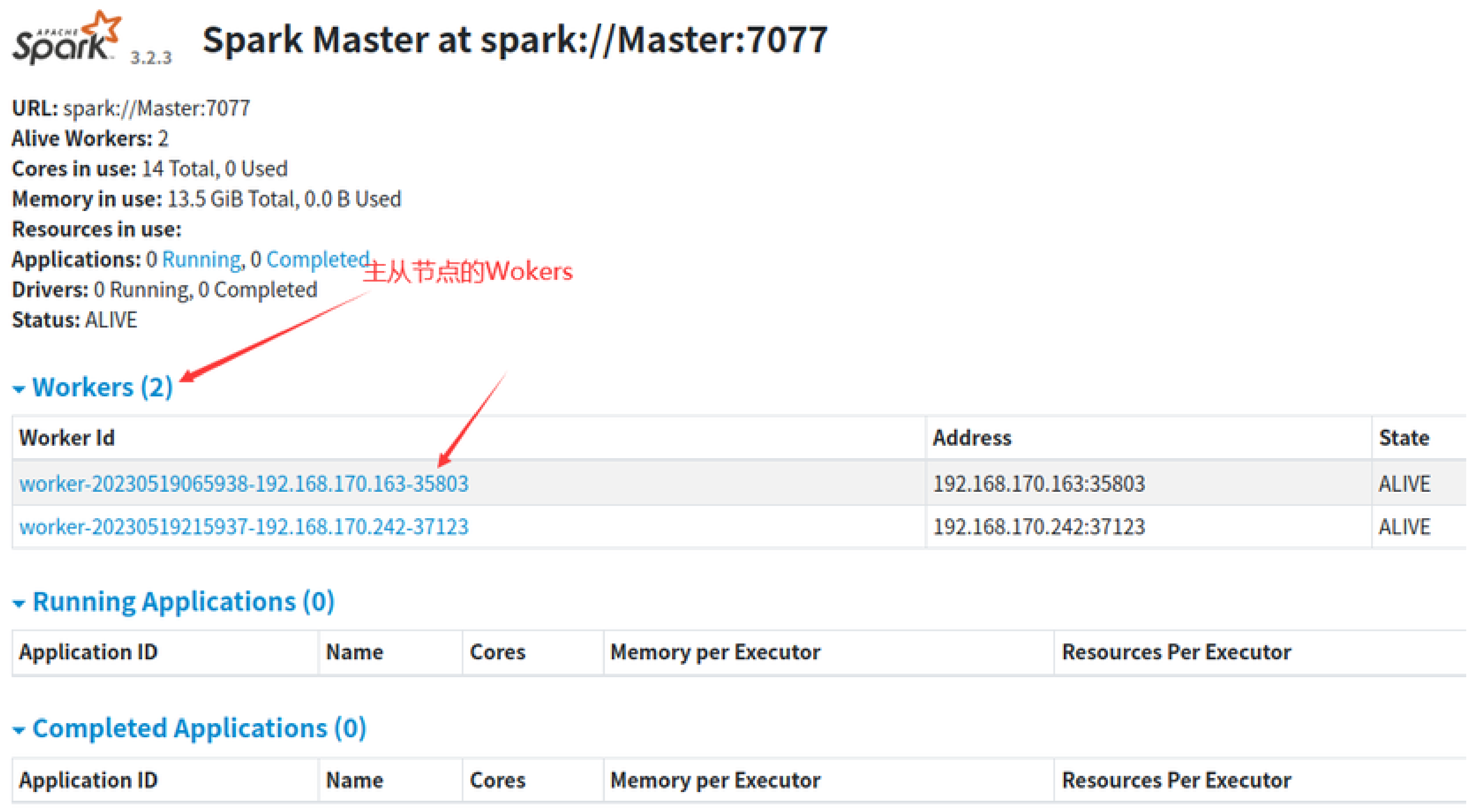
每五秒接受一次9999
端口的套接字数据流

- 在前端界面检查完全分布式 Spark 集群是否成功包括：1. 存活的执行器(Live Executors)的数量应该大于 1；2. Worker 页面,运行应用程序的节点应该不止一个，即 Master 和 slaves。

在 Spark 集群模式下,应用程序将在集群中的多个节点上运行。每个节点上都会启动一个 Spark
执行器(Executor)来运行应用程序中的任务。

✓ ~~待添加完全分布的执行器图示。~~

我们的完全分布式 Spark 仅有一个从节点 Slave1，如图共有两个 workers 和 executors：



Spark Master at spark://Master:7077

URL: spark://Master:7077
Alive Workers: 2
Cores in use: 14 Total, 0 Used
Memory in use: 13.5 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (2)

Worker Id	Address	State
worker-20230519065938-192.168.170.163-35803	192.168.170.163:35803	ALIVE
worker-20230519215937-192.168.170.242-37123	192.168.170.242:37123	ALIVE

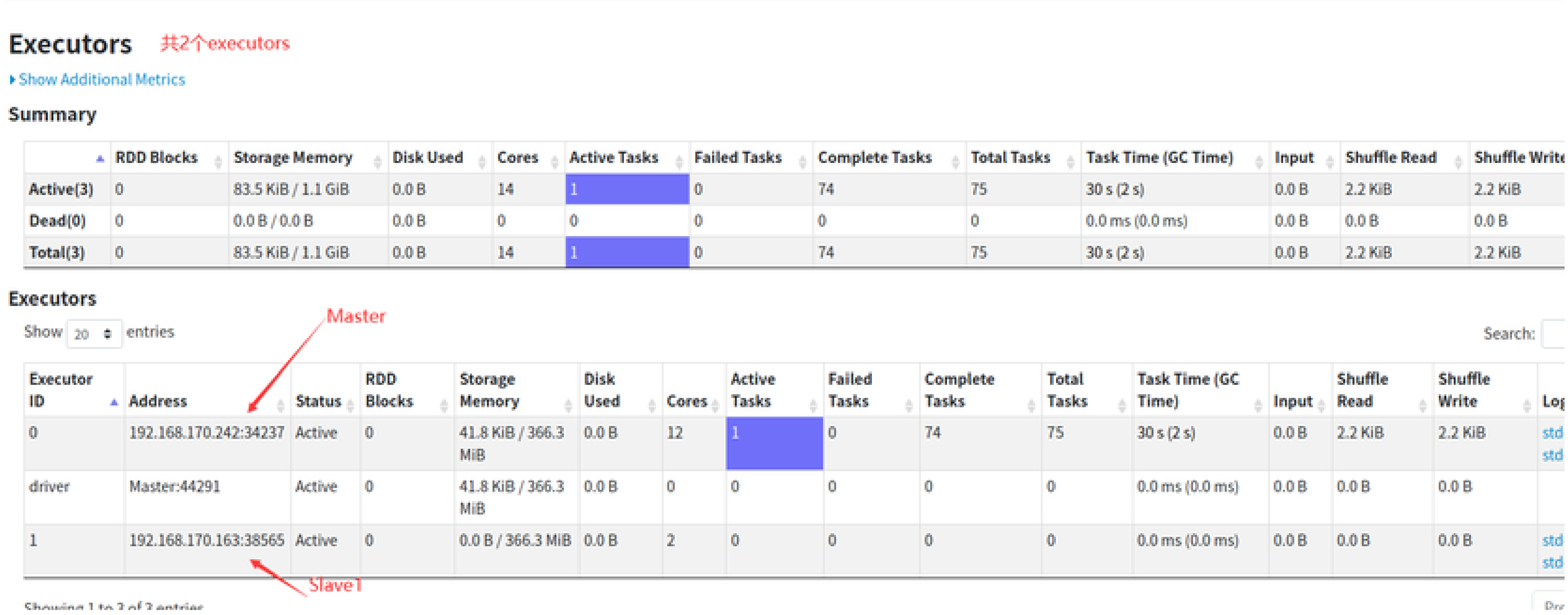
Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor

workers



Executors

▸ Show Additional Metrics

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
Active(3)	0	83.5 KiB / 1.1 GiB	0.0 B	14	1	0	74	75	30 s (2 s)	0.0 B	2.2 KiB	2.2 KiB
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B
Total(3)	0	83.5 KiB / 1.1 GiB	0.0 B	14	1	0	74	75	30 s (2 s)	0.0 B	2.2 KiB	2.2 KiB

Executors

Show 20 entries   Search:

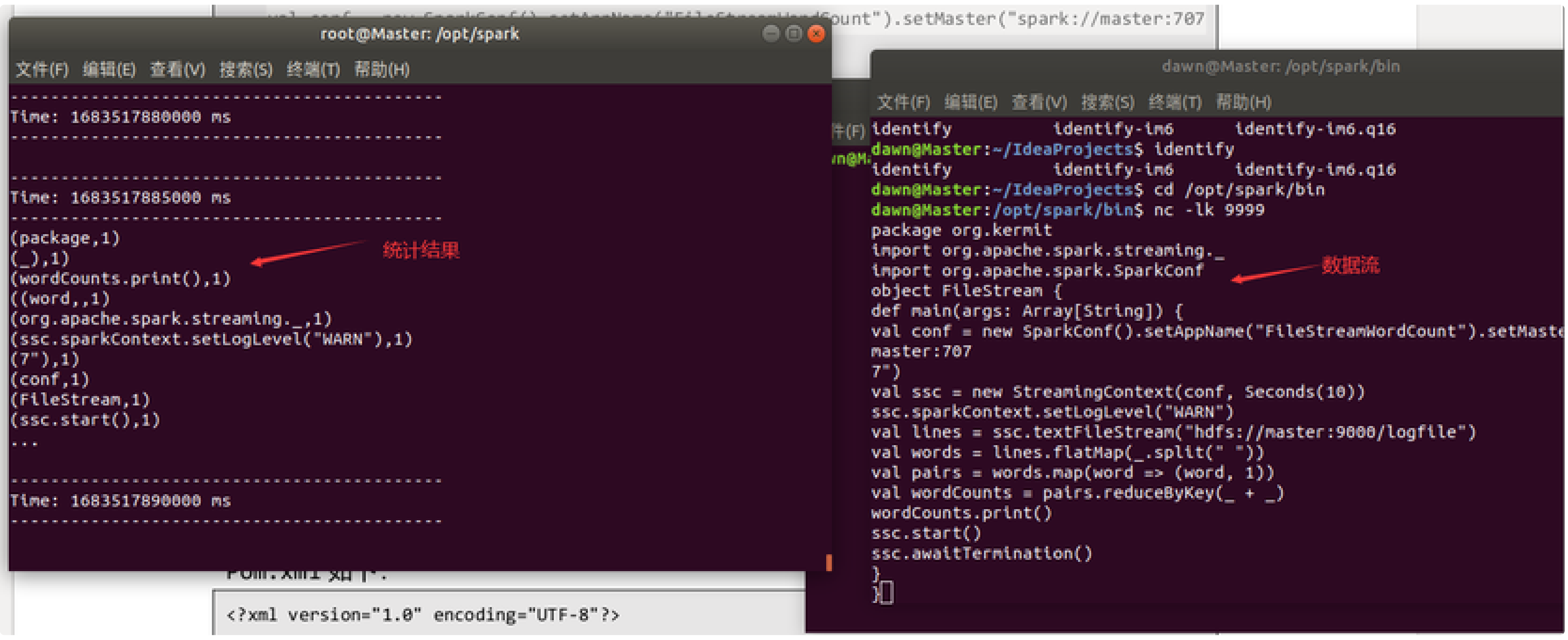
Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Log
0	192.168.170.242:34237	Active	0	41.8 KiB / 366.3 MiB	0.0 B	12	1	0	74	75	30 s (2 s)	0.0 B	2.2 KiB	2.2 KiB	std std
driver	Master:44291	Active	0	41.8 KiB / 366.3 MiB	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	
1	192.168.170.163:38565	Active	0	0.0 B / 366.3 MiB	0.0 B	2	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	std std

Showing 1 to 3 of 3 entries

executors

注意：后续实验截图 workers 和 executors 就略。

- 在 netcat 的控制台输入文字，集群程序给出 word count 结果


root@Master: /opt/spark
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
dawn@Master: /opt/spark/bin
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
-------------------------------------------
Time: 1683517880000 ms
-------------------------------------------

-------------------------------------------
Time: 1683517885000 ms
-------------------------------------------
(package,1)
(_),1)
(wordCounts.print(),1)
((word,,1)
(org.apache.spark.streaming._,1)
(ssc.sparkContext.setLogLevel("WARN"),1)
(7"),1)
(conf,1)
(FileStream,1)
(ssc.start(),1)
...

-------------------------------------------
Time: 1683517890000 ms
-------------------------------------------
```

统计结果

```
identify            identify-im6        identify-im6.q16
dawn@Master:~/IdeaProjects$ identify
identify            identify-im6        identify-im6.q16
dawn@Master:~/IdeaProjects$ cd /opt/spark/bin
dawn@Master:/opt/spark/bin$ nc -lk 9999
package org.kermit
import org.apache.spark.streaming._
import org.apache.spark.SparkConf
object FileStream {
def main(args: Array[String]) {
val conf = new SparkConf().setAppName("FileStreamWordCount").setMaste
master:707
7")
val ssc = new StreamingContext(conf, Seconds(10))
ssc.sparkContext.setLogLevel("WARN")
val lines = ssc.textFileStream("hdfs://master:9000/logfile")
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)
wordCounts.print()
ssc.start()
ssc.awaitTermination()
}
}
```

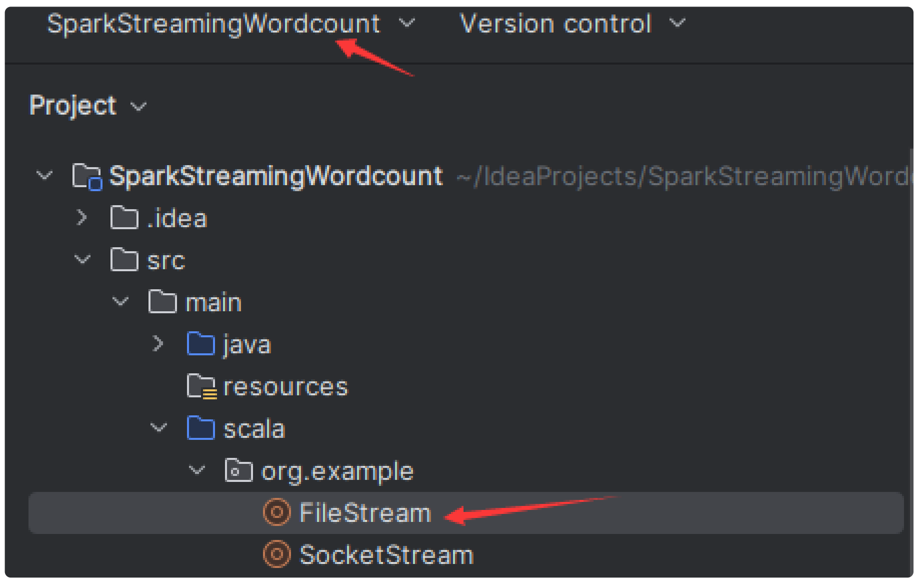
数据流

2.2 完全分布式下 Spark Streaming 处理 HDFS 文件流

该部分是基于 SparkStreaming 处理 hdfs 的文件流，实验五是创建一个数据流端口实时监听处理，该部分即 SparkStreaming 实时监听 hdfs 的文件流进行处理。

下面是实验过程和记录：

1. 配置相关只需在同一个 project 下创建一个名为 `FileStream` 主类即可，因为相关依赖包是一致的，注意需要更改 `pom.xml` 的主类。



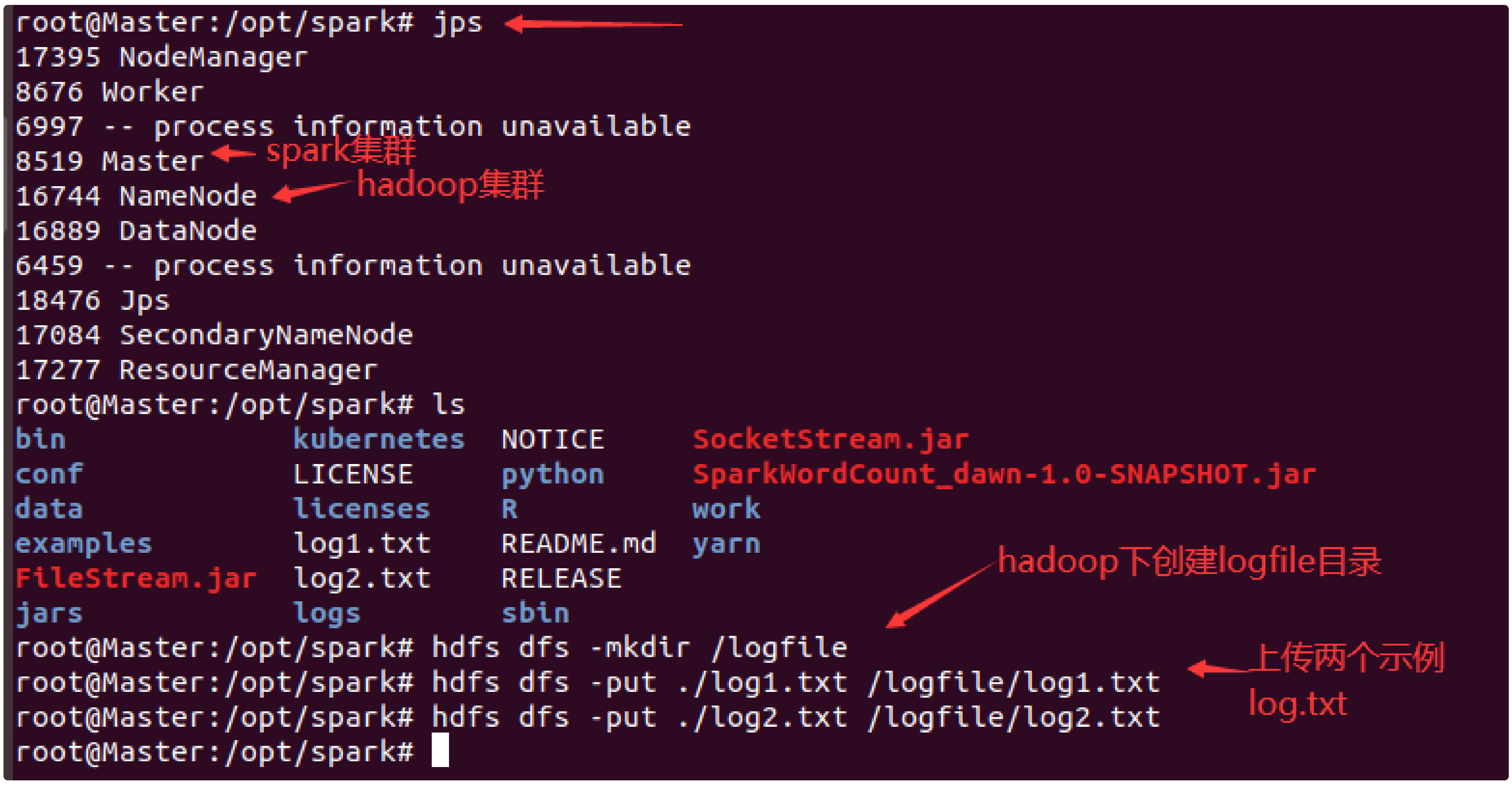
FileStream

2. 同五打包 `FileStream` 带依赖的 jar 包重命名并 mv 到 `/opt/spark` 目录下

3. 将 `FileStream.jar` jar 包提交到 Spark 集群上运行

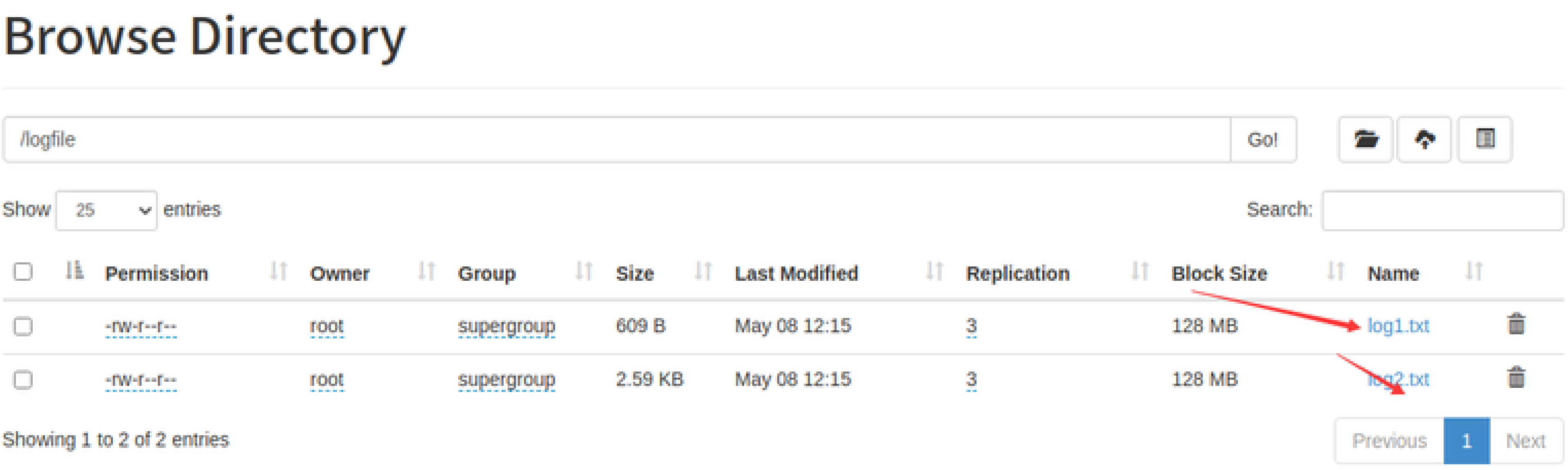
此时 SparkStreaming 正在监听 `hdfs://master:9000/logfile` 的实时上传文件流

4. 准备两个测试文件 `log1.txt` 和 `log2.txt`，启动 hadoop，上传到 hadoop 的 `/logfile/` 目录下，作为 sparkStreaming 处理的 hdfs 文件流，如图：



```
root@Master:/opt/spark# jps
17395 NodeManager
8676 Worker
6997 -- process information unavailable
8519 Master
16744 NameNode
16889 DataNode
6459 -- process information unavailable
18476 Jps
17084 SecondaryNameNode
17277 ResourceManager
root@Master:/opt/spark# ls
bin             kubernetes   NOTICE     SocketStream.jar
conf            LICENSE      python     SparkWordCount_dawn-1.0-SNAPSHOT.jar
data            licenses     R          work
examples        log1.txt     README.md  yarn
FileStream.jar  log2.txt     RELEASE
jars            logs         sbin
root@Master:/opt/spark# hdfs dfs -mkdir /logfile
root@Master:/opt/spark# hdfs dfs -put ./log1.txt /logfile/log1.txt
root@Master:/opt/spark# hdfs dfs -put ./log2.txt /logfile/log2.txt
root@Master:/opt/spark# 
```

在 hdfs 前端查看文件：



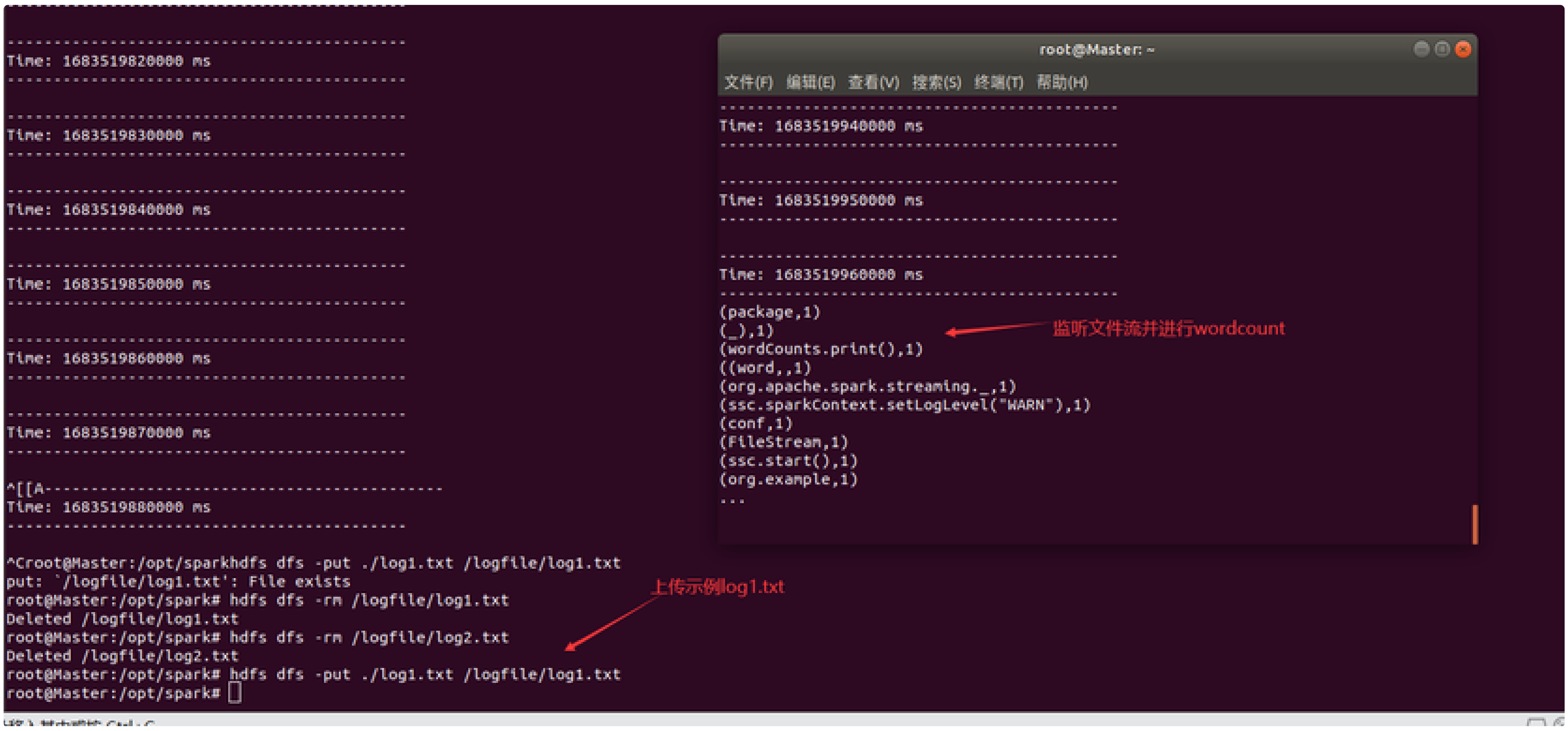
Browse Directory

/logfile

□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
□	-rw-r--r--	root	supergroup	609 B	May 08 12:15	3	128 MB	log1.txt	🗑
□	-rw-r--r--	root	supergroup	2.59 KB	May 08 12:15	3	128 MB	log2.txt	🗑

Showing 1 to 2 of 2 entries

查看运行的 SparkStreaming 程序：



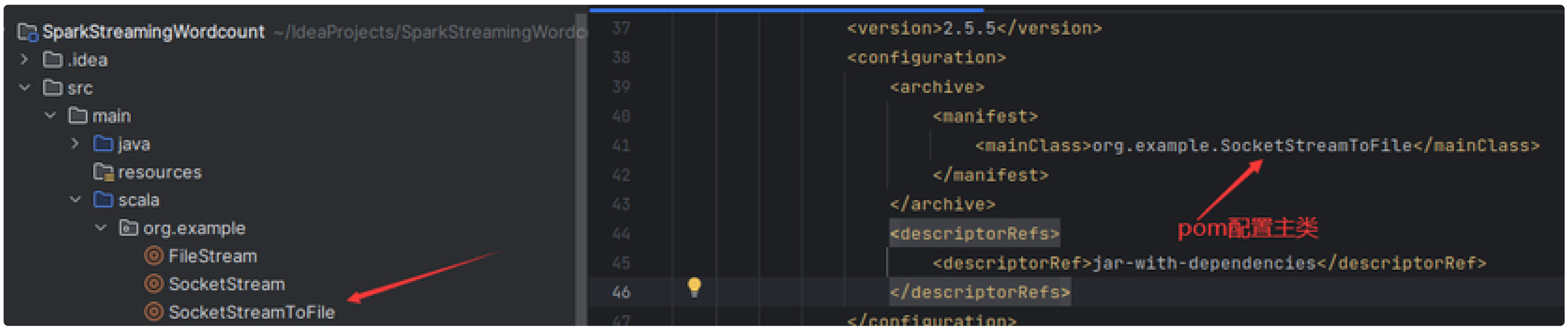
```
-------------------------------------------
Time: 1683519820000 ms
-------------------------------------------

-------------------------------------------
Time: 1683519830000 ms
-------------------------------------------

-------------------------------------------
Time: 1683519840000 ms
-------------------------------------------

-------------------------------------------
Time: 1683519850000 ms
-------------------------------------------

-------------------------------------------
Time: 1683519860000 ms
-------------------------------------------

-------------------------------------------
Time: 1683519870000 ms
-------------------------------------------

^[[A-------------------------------------------
Time: 1683519880000 ms
-------------------------------------------

^Croot@Master:/opt/sparkhdfs dfs -put ./log1.txt /logfile/log1.txt
put: `/logfile/log1.txt': File exists
root@Master:/opt/spark# hdfs dfs -rm /logfile/log1.txt
Deleted /logfile/log1.txt
root@Master:/opt/spark# hdfs dfs -rm /logfile/log2.txt
Deleted /logfile/log2.txt
root@Master:/opt/spark# hdfs dfs -put ./log1.txt /logfile/log1.txt
root@Master:/opt/spark# 
```

```
-------------------------------------------
Time: 1683519940000 ms
-------------------------------------------

-------------------------------------------
Time: 1683519950000 ms
-------------------------------------------

-------------------------------------------
Time: 1683519960000 ms
-------------------------------------------
(package,1)
(_),1)
(wordCounts.print(),1)
((word,,1)
(org.apache.spark.streaming._,1)
(ssc.sparkContext.setLogLevel("WARN"),1)
(conf,1)
(FileStream,1)
(ssc.start(),1)
(org.example,1)
...
```

2.3 完全分布式 Socket 处理结果保存到 HDFS

该部分相当于结合五和六，首先基于完全分布式的 sparkstreaming 实时监听处理创建的 `9999` 端口暴露的数据流，然后将处理结果(wordcount 结果)保存到完全分布式集群的 hadoop 分布式数据库中。

1. 配置文件、jar 包等同上，不再赘述。

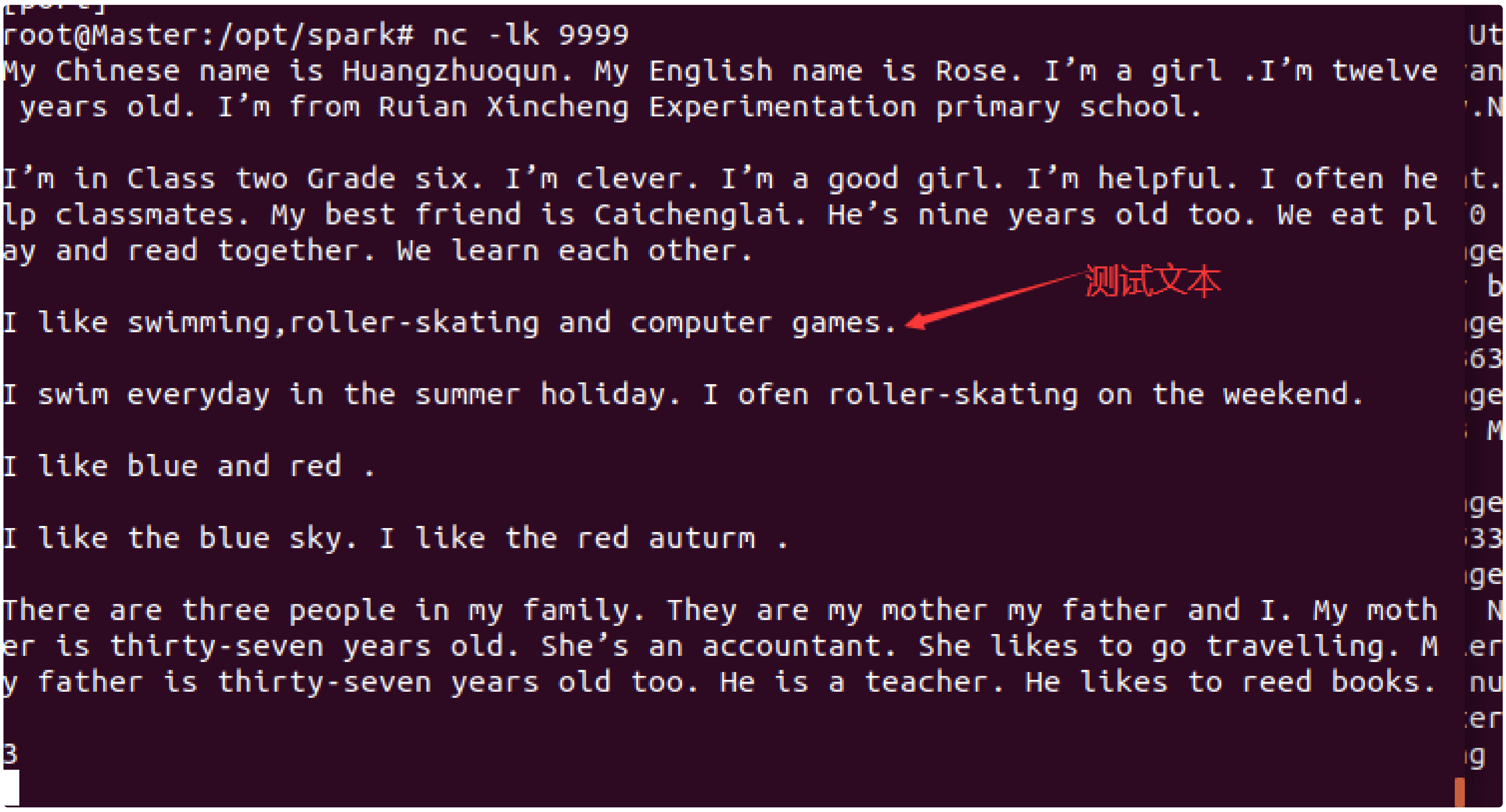


```
SparkStreamingWordcount ~/IdeaProjects/SparkStreamingWordc
  > .idea
  v src
    v main
      > java
        resources
      v scala
        v org.example
            FileStream
            SocketStream
            SocketStreamToFile
```

```
                <version>2.5.5</version>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>org.example.SocketStreamToFile</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
```

2. 先启动 `9999` 端口，再运行 SocketStreamToFile jar 包开始监听。

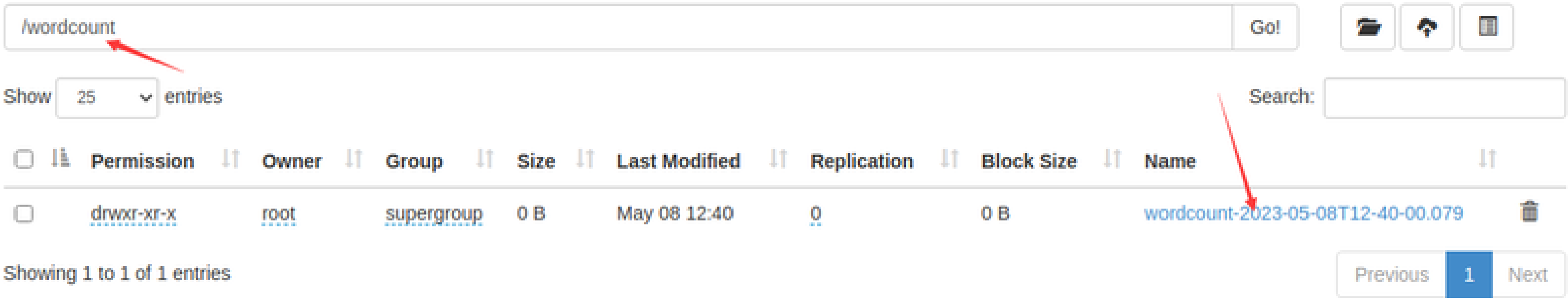
3. 在 `9999` 监听端口输入测试文本数据。



```
[port]
root@Master:/opt/spark# nc -lk 9999
My Chinese name is Huangzhuoqun. My English name is Rose. I’m a girl .I’m twelve
 years old. I’m from Ruian Xincheng Experimentation primary school.

I’m in Class two Grade six. I’m clever. I’m a good girl. I’m helpful. I often he
lp classmates. My best friend is Caichenglai. He’s nine years old too. We eat pl
ay and read together. We learn each other.

I like swimming,roller-skating and computer games.

I swim everyday in the summer holiday. I ofen roller-skating on the weekend.

I like blue and red .

I like the blue sky. I like the red auturm .

There are three people in my family. They are my mother my father and I. My moth
er is thirty-seven years old. She’s an accountant. She likes to go travelling. M
y father is thirty-seven years old too. He is a teacher. He likes to reed books.

3
```

4. 在 hdfs 前端查看 wordcount 保存结果

此时正在完全分布式集群运行的 spark 程序应该从 `9999` 端口中接收到了数据流，并将处理结果保存到了 hdfs 数据库中，在 hdfs 控制界面可以看到创建了一个 wordcount 文件夹，输入文本的 wordcount 结果也被保存到了一个包含时间的文件夹中：

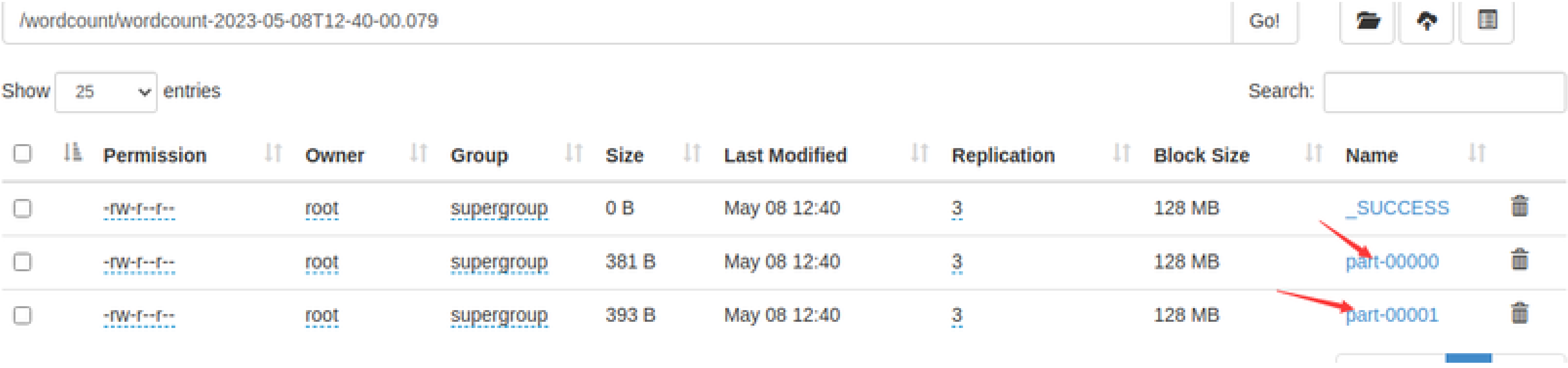


/wordcount Go!

Show 25 entries Search:

☐	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	drwxr-xr-x	root	supergroup	0 B	May 08 12:40	0	0 B	wordcount-2023-05-08T12-40-00.079

Showing 1 to 1 of 1 entries

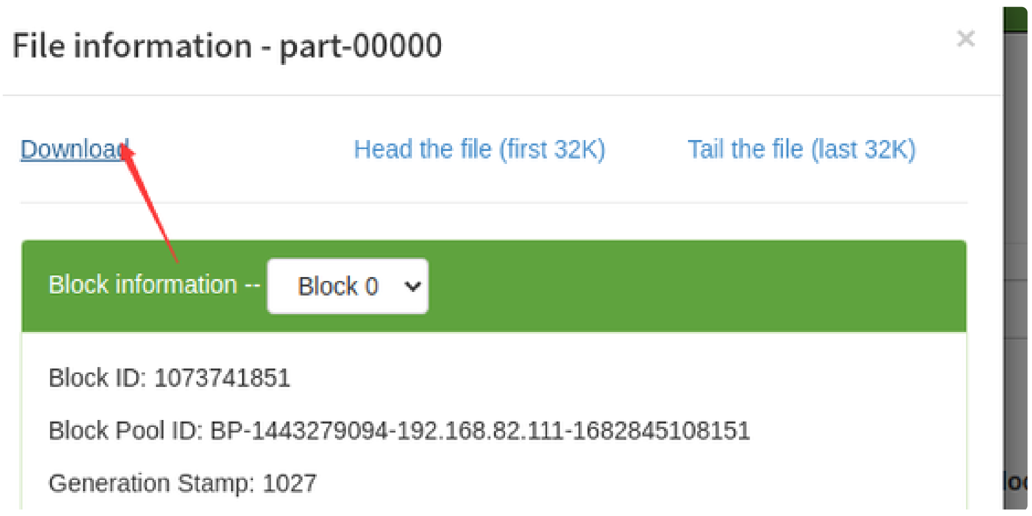


/wordcount/wordcount-2023-05-08T12-40-00.079 Go!

Show 25 entries Search:

☐	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	-rw-r--r--	root	supergroup	0 B	May 08 12:40	3	128 MB	_SUCCESS
☐	-rw-r--r--	root	supergroup	381 B	May 08 12:40	3	128 MB	part-00000
☐	-rw-r--r--	root	supergroup	393 B	May 08 12:40	3	128 MB	part-00001

浏览 part0 和 part1 查看 count 结果，我是下载到本地查看：

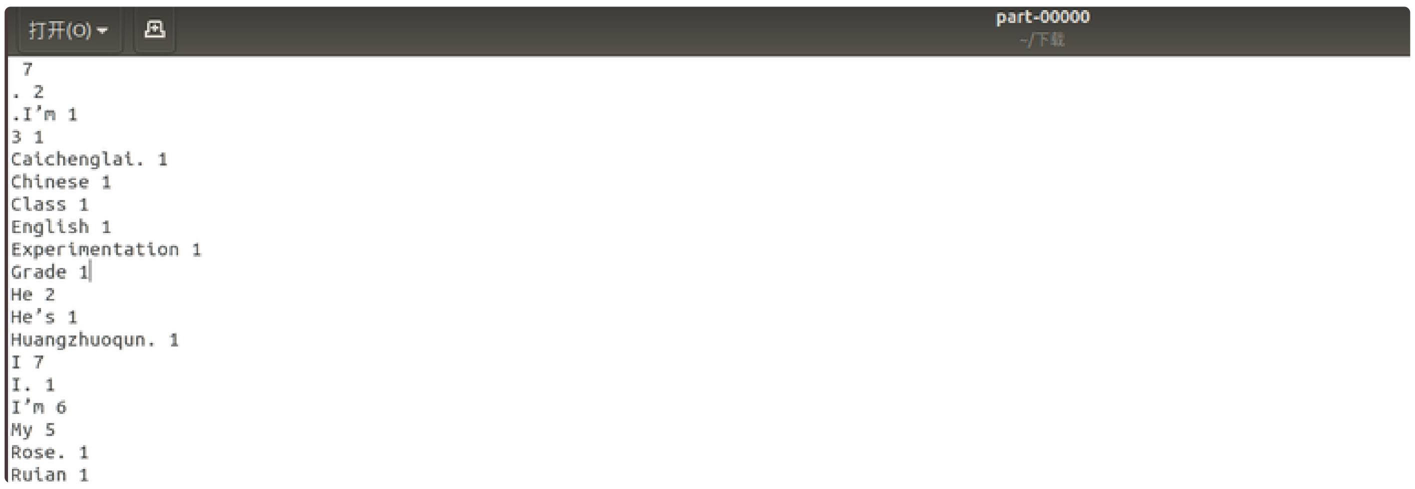


File information - part-00000

Block ID: 1073741851

Block Pool ID: BP-1443279094-192.168.82.111-1682845108151

Generation Stamp: 1027



```
 7
. 2
.I'm 1
3 1
Caichenglai. 1
Chinese 1
Class 1
English 1
Experimentation 1
Grade 1
He 2
He's 1
Huangzhuoqun. 1
I 7
I. 1
I'm 6
My 5
Rose. 1
Ruian 1
```

本地 count

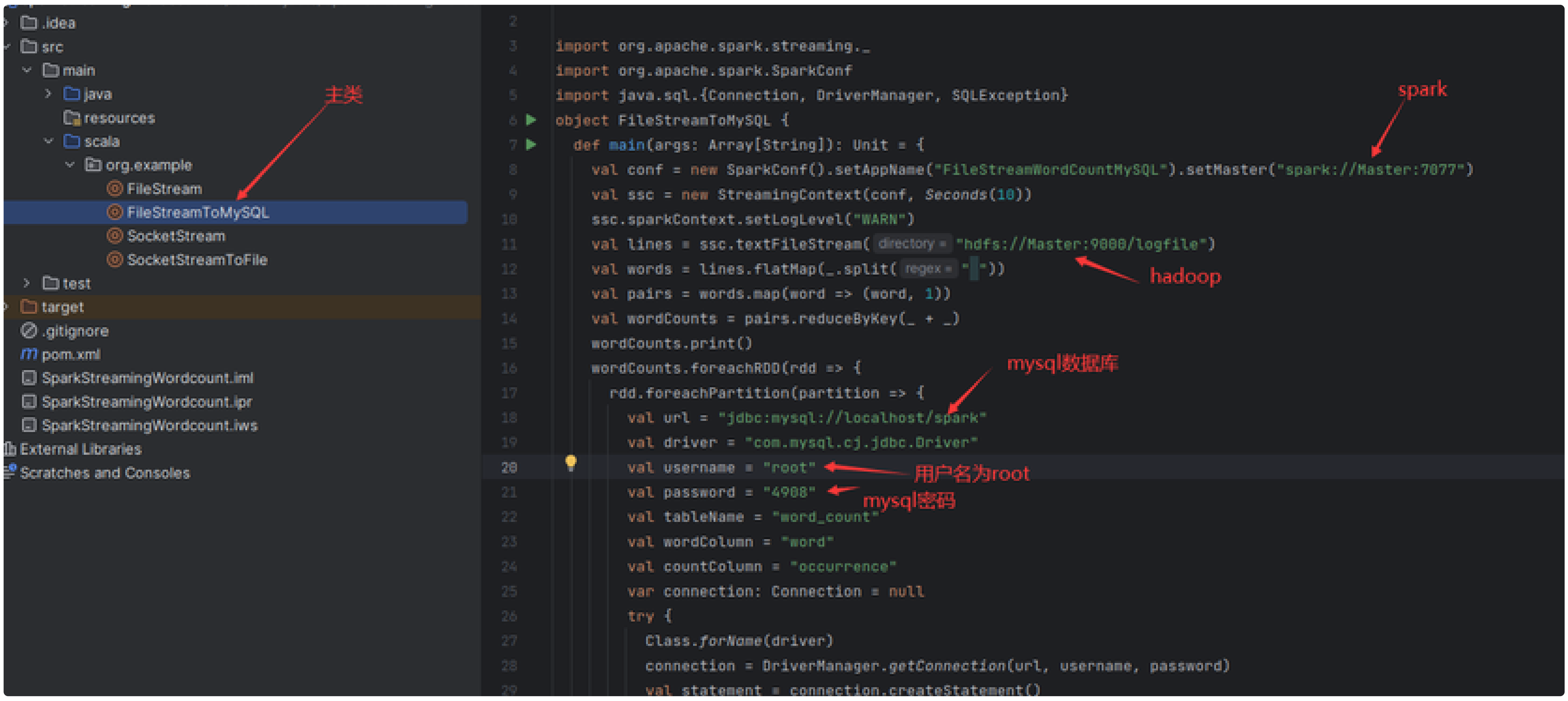
- □ 待插入完全分布 spark 或者 hdfs 的截图

2.4 完全 Spark 处理 hdfs 文件保存到 sql 数据库

mysql 数据库相关配置环境和安装同四的伪分布。该部分也类似，基于完全分布式的 spark 监听处理 hdfs 文件流(10s 一次)，然后最后保存到 mysql 数据库内。

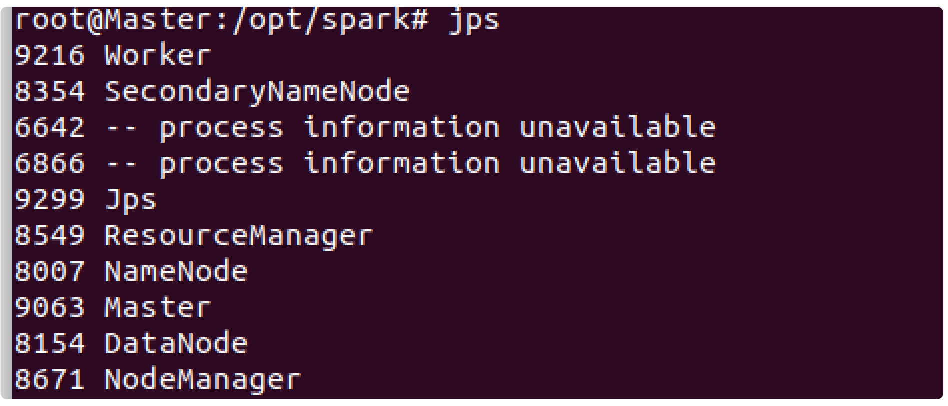
下面是实验过程记录和说明：

1. 创建一个 sql 数据库，并创建用于 word count 的表
2. 在项目中创建 `FileStreamToMySQL` 主类，同上配置好 `pom.xml`，如图：

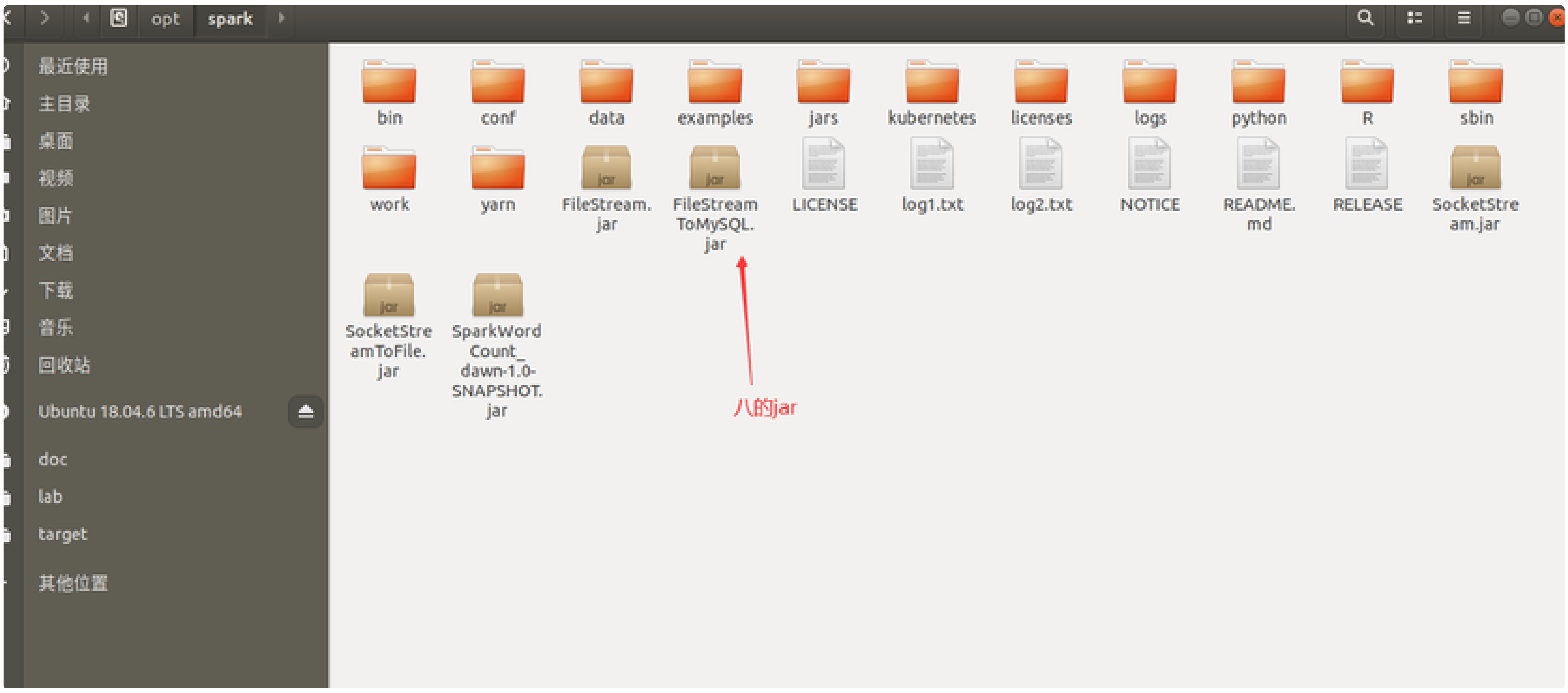


```scala

import org.apache.spark.streaming._
import org.apache.spark.SparkConf
import java.sql.{Connection, DriverManager, SQLException}
object FileStreamToMySQL {
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setAppName("FileStreamWordCountMySQL").setMaster("spark://Master:7077")
    val ssc = new StreamingContext(conf, Seconds(10))
    ssc.sparkContext.setLogLevel("WARN")
    val lines = ssc.textFileStream( directory = "hdfs://Master:9000/logfile")
    val words = lines.flatMap(_.split( regex = " "))
    val pairs = words.map(word => (word, 1))
    val wordCounts = pairs.reduceByKey(_ + _)
    wordCounts.print()
    wordCounts.foreachRDD(rdd => {
      rdd.foreachPartition(partition => {
        val url = "jdbc:mysql://localhost/spark"
        val driver = "com.mysql.cj.jdbc.Driver"
        val username = "root"
        val password = "4908"
        val tableName = "word_count"
        val wordColumn = "word"
        val countColumn = "occurrence"
        var connection: Connection = null
        try {
          Class.forName(driver)
          connection = DriverManager.getConnection(url, username, password)
          val statement = connection.createStatement()
```

3. 同样，打包 jar 程序，重命名 mv 到 `/opt/spark` 目录，运行 hadoop、spark 集群



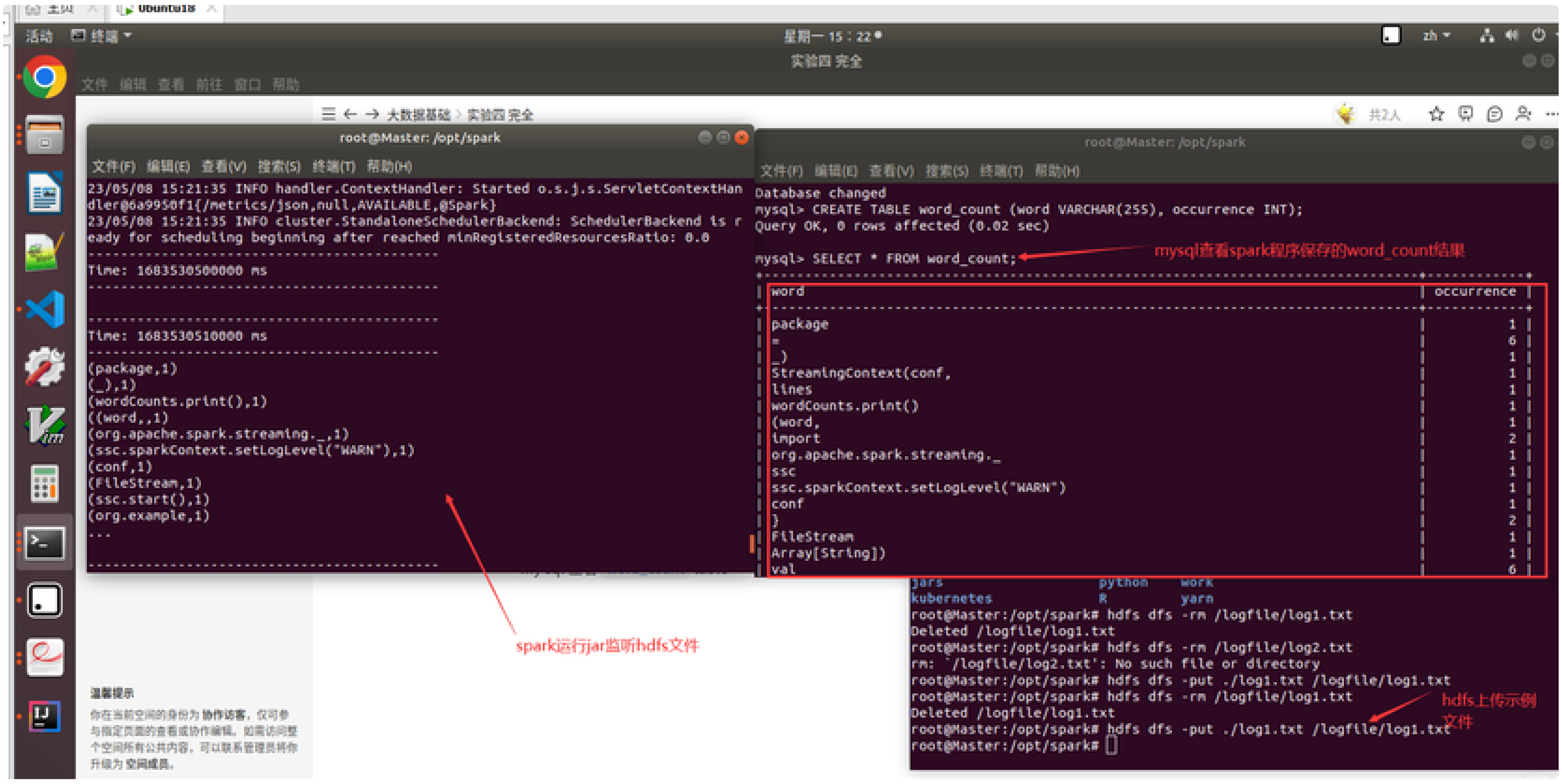
```
root@Master:/opt/spark# jps
9216 Worker
8354 SecondaryNameNode
6642 -- process information unavailable
6866 -- process information unavailable
9299 Jps
8549 ResourceManager
8007 NameNode
9063 Master
8154 DataNode
8671 NodeManager
```



八的jar

4. 把 jar 提交给 spark 运行监听文件流，然后 hadoop 上传文件，最后再查看处理结果，并查看 mysql 是否存储有 word_count 结果。

- spark 运行 jar 包
- hadoop 上传 `log1.txt`
- my sql 查看 `word_count` table



```
23/05/08 15:21:35 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHan
dler@6a9950f1{/metrics/json,null,AVAILABLE,@Spark}
23/05/08 15:21:35 INFO cluster.StandaloneSchedulerBackend: SchedulerBackend is r
eady for scheduling beginning after reached minRegisteredResourcesRatio: 0.0
-------------------------------------------
Time: 1683530500000 ms
-------------------------------------------

-------------------------------------------
Time: 1683530510000 ms
-------------------------------------------
(package,1)
(_),1)
(wordCounts.print(),1)
((word,,1)
(org.apache.spark.streaming._,1)
(ssc.sparkContext.setLogLevel("WARN"),1)
(conf,1)
(FileStream,1)
(ssc.start(),1)
(org.example,1)
...

-------------------------------------------
```

spark运行jar监听hdfs文件

```
Database changed
mysql> CREATE TABLE word_count (word VARCHAR(255), occurrence INT);
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT * FROM word_count;
+-------------------------------------------------------------+------------+
| word                                                        | occurrence |
+-------------------------------------------------------------+------------+
| package                                                     |          1 |
| =                                                           |          6 |
| _)                                                          |          1 |
| StreamingContext(conf,                                      |          1 |
| lines                                                       |          1 |
| wordCounts.print()                                          |          1 |
| (word,                                                      |          1 |
| import                                                      |          2 |
| org.apache.spark.streaming._                                |          1 |
| ssc                                                         |          1 |
| ssc.sparkContext.setLogLevel("WARN")                        |          1 |
| conf                                                        |          1 |
| }                                                           |          2 |
| FileStream                                                  |          1 |
| Array[String])                                              |          1 |
| val                                                         |          6 |
```

mysql查看spark程序保存的word_count结果

```
jars                    python    work
kubernetes              R         yarn
root@Master:/opt/spark# hdfs dfs -rm /logfile/log1.txt
Deleted /logfile/log1.txt
root@Master:/opt/spark# hdfs dfs -rm /logfile/log2.txt
rm: `/logfile/log2.txt': No such file or directory
root@Master:/opt/spark# hdfs dfs -put ./log1.txt /logfile/log1.txt
root@Master:/opt/spark# hdfs dfs -rm /logfile/log1.txt
Deleted /logfile/log1.txt
root@Master:/opt/spark# hdfs dfs -put ./log1.txt /logfile/log1.txt
root@Master:/opt/spark# 
```

hdfs上传示例文件

总结

本次实验，主要学习了如何用 Spark 并行处理框架处理实时的数据流，即使用 SparkStreaming 分别监听 Socket 套接字数据流或者 hdfs 文件数据流运行样例 scala 程序实时对输入流进行词频统计 word_count，具体包括 spark scala 项目的环境配置，打包，spark 结果验证。通过本次实验，对 Spark Streaming 的基本概念和使用有了更深入的了解，包括：

1. Spark Streaming 是 Spark 的一个扩展库，可以让 Spark 处理实时数据流。
2. Spark Streaming 可以通过多种数据源接收实时数据流，如 Kafka、Flume、HDFS、Socket 等。在本次实验中，我们使用 Socket 套接字数据流和 HDFS 文件数据流作为输入源。

3. Spark Streaming 的处理过程包括数据输入、数据转换、数据输出。在本次实验中，我们对输入的数据流进行词频统计，即对每个批处理作业中的单词进行计数，并将结果输出到控制台或 HDFS 文件中。
4. 在 Spark 中，我们可以使用 Scala 或 Java 等语言编写 Spark 应用程序。本次实验使用 Scala 语言编写了 Spark Streaming 应用程序，通过 Spark 的 API 实现了数据的输入、转换和输出等操作。

总之，通过本次实验，学会了如何使用 Spark Streaming 处理实时数据流并配置了集群的 spark，且对 Spark Streaming 的基本概念和使用有了更深入的了解。