

实验五：Spark SQL

刘帅 2020212267

一、加载数据成为分布式表

1、mysql数据

1.1 从数据库读取数据，并进行查询数据。

1.1.1 首先执行mysql命令，在root用户下打开管理员模式。

创建名为MyDB的数据库

```
create database MyDB;
```

切换至MyDB数据库：

```
use MyDB;
```

创建一个名为 StudentInfo 的表格，包含四个字段：ID、Name、Gender 和 birthday。其中，ID 是整数类型，长度为 20；Name 是字符串类型，长度为 20；Gender 是字符类型，长度为 1；birthday 是日期类型。

```
create table StudentInfo(ID int(20),Name varchar(20),Gender char(1),birthday date);
```

写入数据

```
insert into StudentInfo values(1,"A","F","1996-09-12"),(2,"b","M","1995-12-23"),(3,"C","M","1996-10-29"),(4,"D","M","1995-02-25"),(5,"E","F","1997-06-06");
```

1.1.2 从studentinfo读出数据

```

mysql> Bye
root@liushuai_2020212267:/home/forstant/IdeaProjects# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
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 statement.

mysql> select * from StudentInfo;
ERROR 1046 (3D000): No database selected
mysql> use MyDB;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from StudentInfo;
+-----+-----+-----+-----+
| ID | Name | Gender | birthday |
+-----+-----+-----+-----+
| 1 | A | F | 1996-09-12 |
| 2 | b | M | 1995-12-23 |
| 3 | C | M | 1996-10-29 |
| 4 | D | M | 1995-02-25 |
| 5 | E | F | 1997-06-06 |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)

```

可见 我们已经成功将MyDB的studentinfo属性读出来了。

1.2 操作mysql表

1.2.1 修改pom.xml表

导入spark-core, spark-sql和mysql-connector依赖, 从而进行spark分布式计算和分布式sql查询数据处理。

```

<?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.liushuai</groupId>
    <artifactId>SparkSQL</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>3.2.3</version>
        </dependency>
    </dependencies>

```

```

    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-sql_2.12</artifactId>
      <version>3.2.3</version>
    </dependency>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.33</version>
    </dependency>
  </dependencies>
</project>

```

1.2.2 利用SparkSQL从MySQL中读取学生信息

```

package org.liushuai
import org.apache.spark.sql.SparkSession
object code1 {
  def main(args: Array[String]): Unit = {
    // SparkSession
    val spark = SparkSession.builder()
      .appName("mysql_test")
      .master("local")
      .getOrCreate()
    //implicits 隐式转换包, toDF、show 方法等都需要这个包
    //jdbc 连接, 传入上面设置的参数
    val mysqlDF = spark.read
      .format("jdbc")
      .option("url", "jdbc:mysql://127.0.0.1:3306/MyDB") //jdbc 连接的地址
      .option("driver", "com.mysql.cj.jdbc.Driver") //驱动
      .option("dbtable", "StudentInfo") //学生信息表
      .option("user", "root") //用户名
      .option("password", "liushuai") //密码
      .load()
    // 使用 sql 语句要注册成临时表
    mysqlDF.createOrReplaceTempView("studentTable")
    // 输出整个表
    spark.sql("select * from studentTable").show
    spark.sql("select Name, Gender, ID from studentTable where Gender='M' and ID >
1").show
  }
}

```

执行code1结果如下

1. 查询一 展示整个表

```
select * from studentTable
```

```
23/05/28 14:01:29 INFO CodeGenerator: Code generated in 22.525124 ms
+---+---+---+---+
| ID|Name|Gender| birthday|
+---+---+---+---+
| 1|  A|    F|1996-09-12|
| 2|  b|    M|1995-12-23|
| 3|  C|    M|1996-10-29|
| 4|  D|    M|1995-02-25|
| 5|  E|    F|1997-06-06|
+---+---+---+---+
```

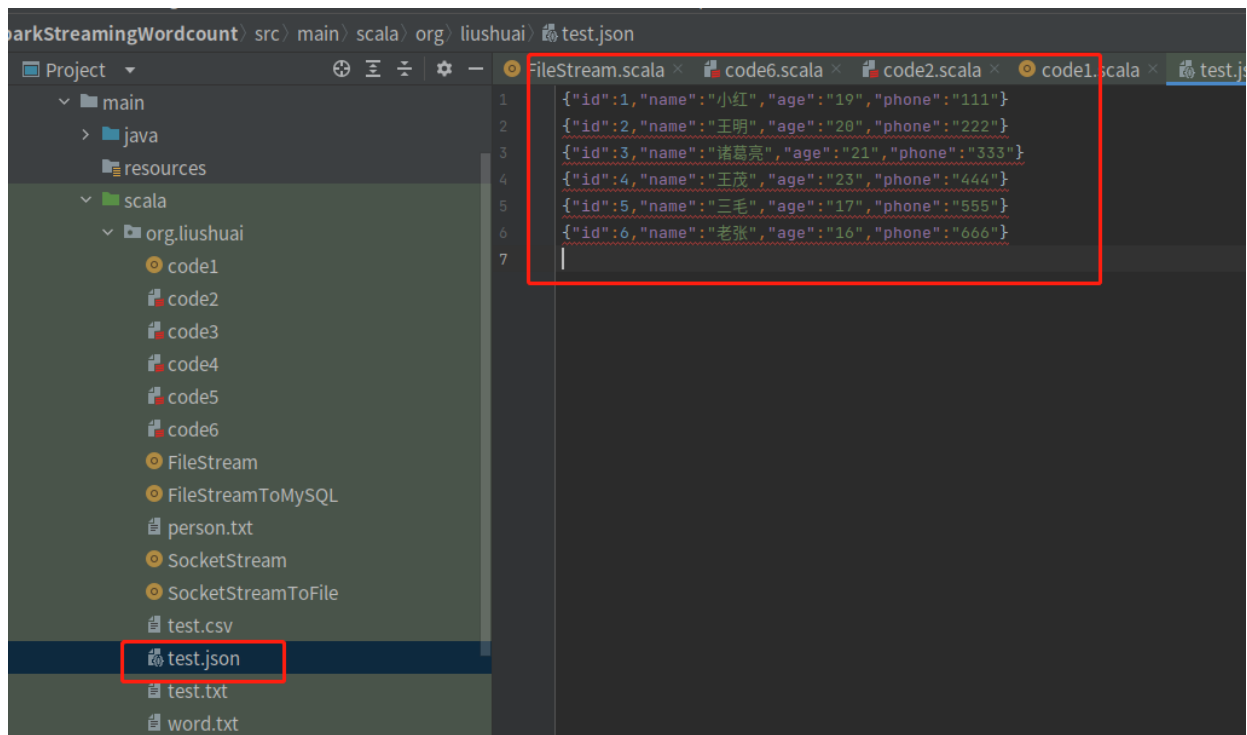
2. 查询二：输出ID 大于1 的男同学的姓名、性别、学号

```
select Name, Gender, ID from studentTable where Gender='M' and ID > 1
```

```
23/05/28 14:01:30 INFO DAGScheduler:
23/05/28 14:01:30 INFO CodeGenerator:
+---+---+---+---+
| Name|Gender| ID|
+---+---+---+---+
|  b|    M|  2|
|  C|    M|  3|
|  D|    M|  4|
+---+---+---+---+
```

1.3 从本地读取数据，将查询结果存入mysql库

准备数据，将数据存入test.json，我们将数据存入scala的项目目录中，和code在相同目录下。



pom.xml的依赖同code1，在此我们只利用sparksql，spark-core和mysqlconnector环境。

在接下来的任务中，我们需要提取年龄小于等于20 的人的所有信息并存入MyDB 数据库的personTable 表中

```
package org.liushuai
import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}
import java.util.Properties
object code2 {
  def main(args: Array[String]): Unit = {
    // zhun bei huan jing
    val spark: SparkSession =
SparkSession.builder().appName("sparksql").master("local[*]").getOrCreate()
    // load data
    val df1: DataFrame =
spark.read.json("/home/liushuai/IdeaProjects/SparkStreamingWordcount/src/main/scala/
org/example/test.json")
    //implicits 隐式转换包，toDF、show 方法等都需要这个包
    import spark.implicits._
    df1.createOrReplaceTempView("personTable")
    // 提取年龄小于等于 20 的人的所有信息并存入 MyDB 数据库的 personTable 表中
    val result: DataFrame = spark.sql("select * from personTable where age <= 20")
    result.show()
    val prop = new Properties()
    prop.put("user","root")
    prop.put("password","liushuai")
    val jdbc: Unit = result.write.mode(SaveMode.Append).jdbc("jdbc:mysql://127.0.0.
1:3306/MyDB","personTable",prop)
  }
}
```

使用 `spark.sql()` 执行 SQL 查询语句，选择所有年龄小于等于 20 的人的信息，并将结果存储到一个新的 DataFrame 中。并使用 `result.write.mode(SaveMode.Append).jdbc()` 方法将 DataFrame 中的数据保存到 MySQL 数据库中的 MyDB 数据库的 personTable 表中。其中，`SaveMode.Append` 表示将数据追加到已有数据的末尾，`jdbc()` 方法用于将 DataFrame 写入 JDBC 数据源。

```
23/05/28 14:10:36 INFO SparkEnv: Registering OutputCommitCoordinator
23/05/28 14:10:37 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
23/05/28 14:10:37 INFO Utils: Successfully started service 'SparkUI' on port 4041.
23/05/28 14:10:37 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://liushuai2020212267:4041
23/05/28 14:10:37 INFO Executor: Starting executor ID driver on host liushuai2020212267
23/05/28 14:10:37 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 45637.
23/05/28 14:10:37 INFO NettyBlockTransferService: Server created on liushuai2020212267:45637
23/05/28 14:10:37 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
23/05/28 14:10:37 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, liushuai2020212267, 45637, None)
23/05/28 14:10:37 INFO BlockManagerMasterEndpoint: Registering block manager liushuai2020212267:45637 with 1949.1 MiB RAM, BlockManagerId(driver, liushuai2020212267, 45637, None)
23/05/28 14:10:37 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, liushuai2020212267, 45637, None)
23/05/28 14:10:37 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, liushuai2020212267, 45637, None)
```

```
23/05/22 11:11:00 INFO CodeGenerator: Code generated in 11.257094 ms
+-----+
|age| id|  name|phone|
+-----+
| 19|  1|  小红|  111|
| 20|  2|  王明|  222|
| 21|  3|  诸葛亮| 333|
| 23|  4|  王茂|  444|
| 17|  5|  三毛|  555|
| 16|  6|  老张|  666|
+-----+

23/05/22 11:11:00 INFO FileSourceStrategy: Pushed Filters:
23/05/22 11:11:00 INFO FileSourceStrategy: Post-Scan Filters:
23/05/22 11:11:00 INFO FileSourceStrategy: Output Data Schema: struct<_c0: string, _c1: string, _c2: stri
```

可见，按照年龄所筛选出的目标的信息成功回显。并且sparkUI成功绑定4041地址，可以用于监控和调试应用程序的运行状况。

利用mysql查看表格是否被成功写入MyDB:

```
e2 > mysql> show tables;
+-----+
| Tables_in_MyDB |
+-----+
| StudentInfo    |
| personTable    |
+-----+
2 rows in set (0.01 sec)

mysql>
finished: jdbc at code2.scala:19, took 0.560169 s
stop() from shutdown hook
web UI at http://liushuai2020212267:4040
ndpoint: MapOutputTrackerMasterEndpoint stopped!
```

查看personTable表格内容:

```
select * from personTable;
```

```
mysql> select * from personTable
->
-> ;
+-----+-----+-----+-----+
| age | id | name | phone |
+-----+-----+-----+-----+
| 19  | 1  | 小红 | 111   |
| 20  | 2  | 王明 | 222   |
| 17  | 5  | 三毛 | 555   |
| 16  | 6  | 老张 | 666   |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

1.4 读取其他数据类型文件(.txt,.csv,.json)

1.4.1 修改pom.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.liushuai</groupId>
  <artifactId>SparkSQL</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>3.2.3</version>
    </dependency>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-sql_2.12</artifactId>
      <version>3.2.3</version>
    </dependency>
  </dependencies>
</project>
```

1.4.2 同时加载 .txt,.csv,.json 的数据

```
package org.liushuai
import org.apache.spark.sql
import org.apache.spark.sql.{DataFrame, SparkSession}
object code3 {
  def main(args: Array[String]): Unit = {
    // 准备环境
    val spark: SparkSession =
      SparkSession.builder().appName("sparksql").master("local[*]").getOrCreate()
    // 加载数据
    val df1: DataFrame =
      spark.read.json("/home/liushuai/IdeaProjects/SparkStreamingWordcount/src/main/scala/org/liushuai/test.json")
    val df2: DataFrame =
      spark.read.csv("/home/liushuai/IdeaProjects/SparkStreamingWordcount/src/main/scala/org/liushuai/test.txt")
    val df3: DataFrame =
      spark.read.text("/home/liushuai/IdeaProjects/SparkStreamingWordcount/src/main/scala/org/liushuai/test.csv")
```

```

df1.printSchema()
df2.printSchema()
df3.printSchema()
df1.show()
df2.show()
df3.show()
// 关闭资源
spark.stop()
}
}

```

1.4.3 运行结果

数据data中每列的名称和类型

```

20/05/20 14:20:47 INFO InMemoryFileIndex: 10000
root
 |-- age: string (nullable = true)
 |-- id: long (nullable = true)
 |-- name: string (nullable = true)
 |-- phone: string (nullable = true)

root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)

root
 |-- value: string (nullable = true)

```

输出文件内容，不同文件类型有不同形式

1. json文件的输出形式

```

+---+---+-----+---+
|age| id|  name|phone|
+---+---+-----+---+
| 19|  1|  小红|  111|
| 20|  2|  王明|  222|
| 21|  3|诸葛亮|  333|
| 23|  4|  王茂|  444|
| 17|  5|  三毛|  555|
| 16|  6|  老张|  666|
+---+---+-----+---+

```

2. txt的输出形式


```

23/05/28 14:26:49 INFO FileSourceStrategy: Output Data Schema: struct<value: string>
+-----+
|_c0|    |_c1|_c2|
+-----+
| 1|zhangsan| 20|
| 2|    lisi| 30|
| 3|   wangwu| 25|
+-----+

23/05/28 14:26:49 INFO MemoryStore: Block broadcast_9 stored as values in memory (estimated size 337.3 KiB, free 1946.9 MiB)
23/05/28 14:26:49 INFO MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 32.3 KiB, free 1946.9 MiB)
23/05/28 14:26:49 INFO BlockManagerInfo: Added broadcast_9_piece0 in memory on liushuai2020212267:43691 (size: 32.3 KiB, free: 1948.9 MiB)
23/05/28 14:26:49 INFO SparkContext: Created broadcast 9 from show at code7.scala:17

```

3. csv的输出形式

```

23/05/28 14:26:50 INFO CodeGenerator: Code generated in 7.523483 ms
23/05/28 14:26:50 INFO SparkUI: Stopped Spark web UI at http://liushuai2020212267:4040
+-----+
|      value|
+-----+
|1,zhangsan,20|
|  2,lisi,30|
|  3,wangwu,25|
+-----+

```

二、spark sql 实现信息查询

在SparkSQL 模块中，将结构化数据封装到DataFrame 或Dataset 集合中后，提供了两种方式分析处理数据:

- 1) **SQL 编程**，将DataFrame/Dataset 注册为临时视图或表，编写SQL 语句，类似HiveQL;
- 2) **DSL (domain-specific language)编程**，（类似于面向对象）调用DataFrame/Dataset API 类似RDD 中的函数;

两种分析处理数据的区别：

- SQL 编程：使用 SQL 语言进行数据分析和处理，首先需要将 DataFrame 或 Dataset 注册为临时表或视图，然后编写 SQL 查询语句进行数据操作。这种方法类似于 HiveQL，语法简洁、易于理解，适合熟悉 SQL 语言的开发人员使用。SQL 编程的优点是可以直接使用 SQL 语句进行数据分析和处理，具有较高的灵活性和可读性，代码简洁易懂。缺点是不太灵活，只能使用 SQL 语言提供的常规操作进行数据处理，无法进行复杂的算法实现。
- DSL 编程：使用函数式 API 进行数据分析和处理，通过调用 DataFrame 或 Dataset API 中的函数实现。这种方法类似于面向对象的编程风格，使用 Scala 或 Java 语言进行编写，适合熟悉编程语言的开发人员使用。DSL 编程的优点是可以使用强类型的 API 进行数据处理，具有较高的灵活性和可扩展性，可以进行复杂的算法实现。缺点是需要熟悉 API 的使用方法和语法，代码相对较长，可读性较差。

2.1 查询文件person.txt（文件位置可以自定义，同时修改Code 中文件的位置）

```

1 zhangsan 20
2 lisi 29
3 wangwu 25
4 zhaoliu 30
5 tianqi 35
6 qishi 40

```

查询需求：

- 1、查看name 字段的数据
- 2、查看name 和age 字段数据

- 3、查询所有的name 和age, 并将age+1
- 4、过滤age 大于等于25 的
- 5、统计年龄大于30 的人数
- 6、按年龄进行分组并统计相同年龄的人数
- 7、查询姓名==张三的

2.2 sql查询模式的java代码如下:

```
import org.apache.spark.SparkContext
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.{DataFrame, SparkSession}
object code4 {
    def main(args: Array[String]): Unit = {
        //TODO 0.准备环境
        val spark: SparkSession =
SparkSession.builder().appName("sparksql").master("local[*]").getOrCreate()
        val sc: SparkContext = spark.sparkContext
        sc.setLogLevel("WARN")
        //TODO 1.加载数据: 改成自己的路径
        val lines: RDD[String] =
sc.textFile("/home/liushuai/IdeaProjects/SparkStreamingwordcount/src/main/scala/org/
liushuai/person.txt")
        //TODO 2.处理数据
        val personRDD: RDD[Person] = lines.map(line => {
            val arr: Array[String] = line.split(" ")
            Person(arr(0).toInt, arr(1), arr(2).toInt)
        })
        //RDD-->DF
        import spark.implicits._
        val personDF: DataFrame = personRDD.toDF()
        personDF.printSchema()
        personDF.show()
        //TODO =====SQL=====
        personDF.createOrReplaceTempView("t_person")//创建临时的,当前 SparkSession 也可以用
        //1.查看 name 字段的数据
        spark.sql("select name from t_person").show()
        //2.查看 name 和 age 字段数据
        spark.sql("select name,age from t_person").show()
        //3.查询所有的 name 和 age, 并将 age+1
        spark.sql("select name,age,age+1 from t_person").show()
        //4.过滤 age 大于等于 25 的
        spark.sql("select name,age from t_person where age >= 25").show()
        //5.统计年龄大于 30 的人数
        spark.sql("select count(*) from t_person where age > 30").show()
        //6.按年龄进行分组并统计相同年龄的人数
        spark.sql("select age,count(*) from t_person group by age").show()
        //7.查询姓名=张三的
        spark.sql("select * from t_person where name = 'zhangsan']").show()
        spark.stop()
    }
}
case class Person(id:Int,name:String,age:Int)
```

```
}
```

2.2.1 printSchema打印信息

personDF.printSchema()作用是打印 DataFrame 的结构信息，包括每个列的名称、数据类型和是否可空等元数据信息。具体来说，该函数会输出 DataFrame 的结构信息，包括列名和数据类型，以及每个列是否可为空。

personDF.printSchema()的结果如下

```
23/05/28 14:36:12 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, liushuai2020212267, 35753, None)
23/05/28 14:36:12 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, liushuai2020212267, 35753, None)
root
 |-- id: integer (nullable = false)
 |-- name: string (nullable = true)
 |-- age: integer (nullable = false)
```

2.2.2 person DF.show()的结果如下

```
+---+-----+---+
| id|   name|age|
+---+-----+---+
|  1|zhangsan| 20|
|  2|   lisi| 29|
|  3| wangwu| 25|
|  4| zhaoliu| 30|
|  5| tianqi| 35|
|  6|  qishi| 40|
+---+-----+---+
```

2.2.3 查看name字段的数据

```
+-----+
|   name|
+-----+
|zhangsan|
|   lisi|
| wangwu|
| zhaoliu|
| tianqi|
|  qishi|
+-----+
```

2.2.4 查看name和age字段数据

```
+-----+---+
|   name|age|
+-----+---+
|zhangsan| 20|
|   lisi| 29|
| wangwu| 25|
| zhaoliu| 30|
|  tianqi| 35|
|   qishi| 40|
+-----+---+
```

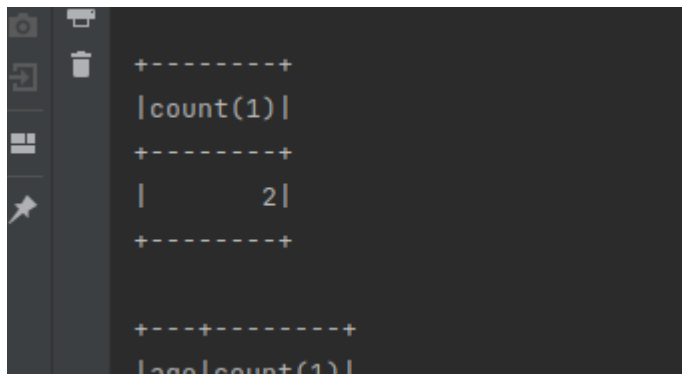
2.2.5 查看所有name和age，并将age+1的结果输出

```
+-----+---+-----+
|   name|age|(age + 1)|
+-----+---+-----+
|zhangsan| 20|      21|
|   lisi| 29|      30|
| wangwu| 25|      26|
| zhaoliu| 30|      31|
|  tianqi| 35|      36|
|   qishi| 40|      41|
+-----+---+-----+
```

2.2.6 过滤age大于等于25的

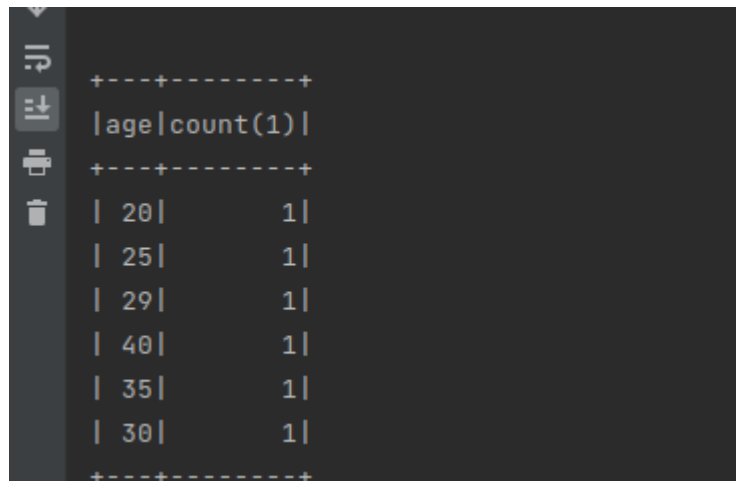
```
+-----+---+
|   name|age|
+-----+---+
|   lisi| 29|
| wangwu| 25|
| zhaoliu| 30|
|  tianqi| 35|
|   qishi| 40|
+-----+---+
```

2.2.7 统计年龄大于30岁的人数



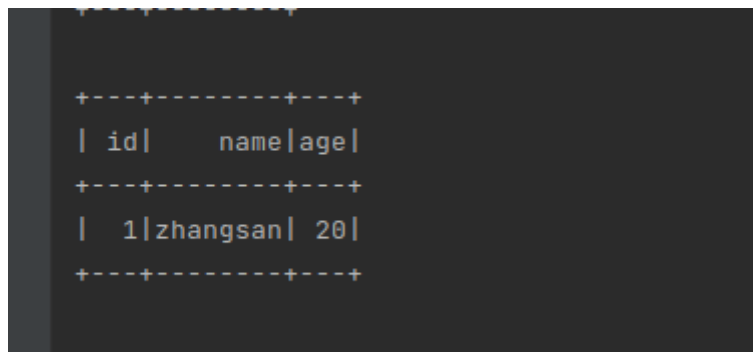
count(1)	
2	

2.2.8 按年龄进行分组并统计相同年龄的人数



age	count(1)
20	1
25	1
29	1
40	1
35	1
30	1

2.2.9 查询姓名为"zhangsan"的



id	name	age
1	zhangsan	20

2.3 dsl查询模式的java代码如下:

```
package org.liushuai
import org.apache.spark.SparkContext
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.{DataFrame, SparkSession, Dataset}
object code5 {
  def main(args: Array[String]): Unit = {
    //TODO 0.准备环境
    val spark: SparkSession =
      SparkSession.builder().appName("sparksq1").master("local[*]").getOrCreate()
    val sc: SparkContext = spark.sparkContext
    sc.setLogLevel("WARN")
    //TODO 1.加载数据: 路径改成自己的
```

```

val lines: RDD[String] =
sc.textFile("/home/liushuai/IdeaProjects/SparkStreamingwordcount/src/main/scala/org/
liushuai/person.txt")
//TODO 2.处理数据
val personRDD: RDD[Person] = lines.map(line => {
    val arr: Array[String] = line.split(" ")
    Person(arr(0).toInt, arr(1), arr(2).toInt)
})
//RDD-->DF
import spark.implicits._
val personDF: DataFrame = personRDD.toDF()
personDF.printSchema()
personDF.show()
//=1.查看 name 字段的数据
//personDF.select(personDF.col("name"))
personDF.select("name").show()
//=2.查看 name 和 age 字段数据
personDF.select("name","age").show()
//=3.查询所有的 name 和 age, 并将 age+1
//personDF.select("name","age","age+1").show()//错误的:cannot resolve '`age+1`'
given input columns: [age, id, name];
//注意$是把字符串转为了 Column 列对象
personDF.select($"name", $"age", $"age" + 1).show()
//注意'是把列名转为了 Column 列对象
personDF.select('name,'age,'age + 1).show()
//=4.过滤 age 大于等于 25 的
personDF.filter("age >= 25").show()
personDF.filter($"age" >= 25).show()
personDF.filter('age >= 25).show()
//=5.统计年龄大于 30 的人数
val count: Long = personDF.where('age > 30).count() //where 底层 filter
println("年龄大于 30 的人数为:"+count)
//=6.按年龄进行分组并统计相同年龄的人数
personDF.groupBy('age).count().show()
//=7.查询姓名=张三的
personDF.filter("name = 'zhangsan").show()
personDF.filter($"name"==="zhangsan").show()
personDF.filter('name==="zhangsan").show()
personDF.filter('name != "zhangsan").show()
//TODO 3.输出结果
//TODO 4.关闭资源
spark.stop()
}
case class Person(id:Int,name:String,age:Int)
}

```

输出结果同上:

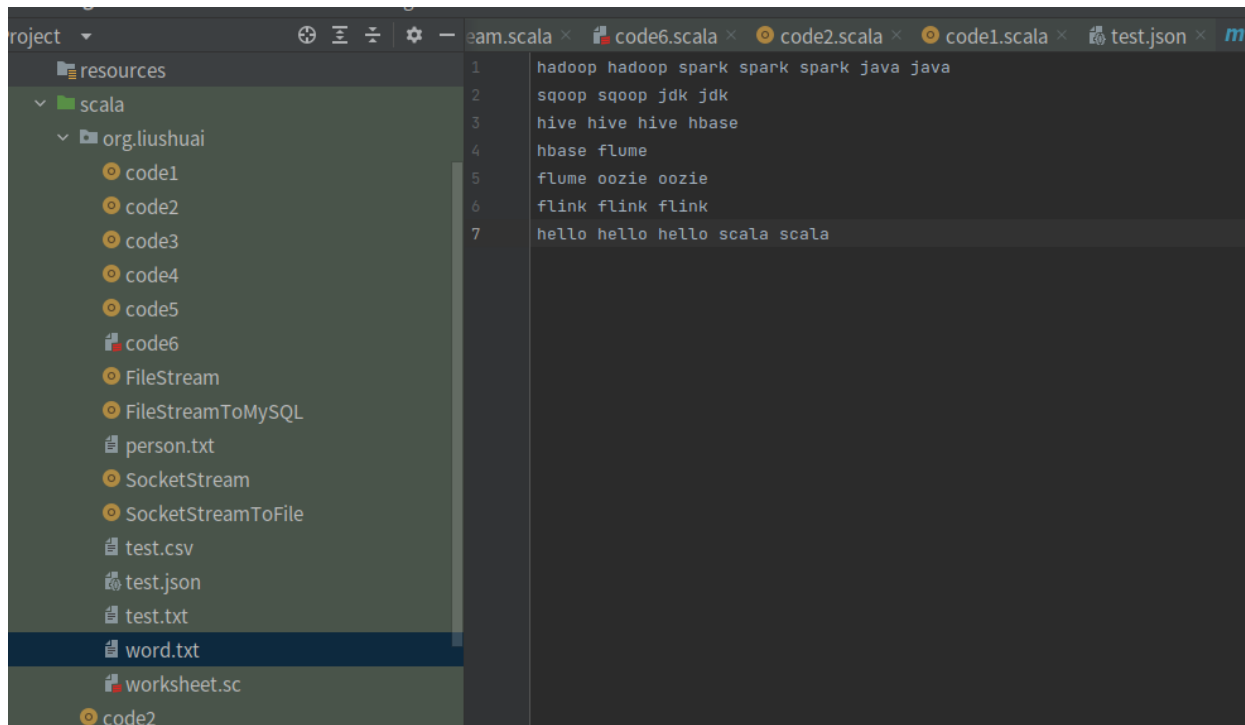
```
resources  code5  main(array[...])
code5 x
↑
|   tianqi| 35|
|   qishi| 40|
+-----+
↑
|   name|age|(age + 1)|
+-----+
|zhangsan| 20|      21|
|   lisi| 29|      30|
| wangwu| 25|      26|
| zhaoliu| 30|     31|
|   tianqi| 35|     36|
|   qishi| 40|     41|
+-----+

|   name|age|(age + 1)|
+-----+
|zhangsan| 20|      21|
|   lisi| 29|      30|
| wangwu| 25|      26|
| zhaoliu| 30|     31|
|   tianqi| 35|     36|
|   qishi| 40|     41|
+-----+

| id| name|age|
+---+-----+-----+
Version Control  Run  TODO  Problems  Terminal  Services  Build  Dep
Build completed successfully in 3 sec, 94 ms (2 minutes ago)
```

三、spark sql实现wordcount

3.1 编辑测试文件并构建目录格式



实现wordcount代码如下

```
package org.liushuai

import org.apache.spark.SparkContext
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.{DataFrame, SparkSession, Dataset}
object code6 {
  def main(args: Array[String]): Unit = {
    //1.创建 SparkSession
    val spark: SparkSession =
      SparkSession.builder().master("local[*]").appName("SparkSQL").getOrCreate()
    val sc: SparkContext = spark.sparkContext
    sc.setLogLevel("WARN")
    //2.读取文件
    val fileDS: Dataset[String] =
      spark.read.textFile("/home/liushuai/IdeaProjects/SparkStreamingWordcount/src/main/sc
      ala/org/liushuai/word.txt")
    //3.对每一行按照空格进行切分并压平
    import spark.implicits._
    val wordDS: Dataset[String] = fileDS.flatMap(_.split(" "))//注意:正确,因为 DS 有泛
    型,知道_是 String
    //wordDS.show()
    //4.对上面的数据进行 wordCount
    wordDS.createOrReplaceTempView("t_word")
    val sql =
      """
        |select value ,count(value) as count
        |from t_word
        |group by value
        |order by count desc
      """
    .stripMargin
```



```

spark.sql(sql).show()
sc.stop()
spark.stop()
}
}

```

```

23/05/28 14:47:53 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, liushuai2020212267, 38217, None)
23/05/28 14:47:53 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, liushuai2020212267, 38217, None)
+-----+
| value|count|
+-----+
| hive|    3|
| hello|  3|
| spark|  3|
| flink|  3|
| hbase|  2|
| flume|  2|
| jdk|   2|
| sqoop|  2|
| oozie|  2|
| hadoop| 2|
| scala|  2|
| java|   2|
+-----+

```

四、Sqark SQL集群形式运行wordcount

4.1 启动hdfs和spark集群

首先执行 start-dfs.sh 与start-yarn.sh脚本启动hdfs和spark集群

4.2 将测试文件上传至hdfs中

我们首先将test文件放到hadoop的根目录下

并利用hadoop fs -ls /对于文件进行查询，可见 我们此时已成功将test.txt放至hadoop集群。

```

root@liushuai2020212267:~# hadoop fs -put /home/matianchi/IdeaProjects/SparkStreamingWordcount/src/main/scala/org/liushuai/test.txt /
root@liushuai2020212267:~# hadoop fs -ls /
Found 4 items
drwxr-xr-x   - root supergroup          0 2023-04-30 10:24 /MRoutput
-rw-r--r--   1 root supergroup    6747 2023-04-30 10:17 /ceshi.txt
drwxr-xr-x   - root supergroup          0 2023-04-27 13:55 /hbase
-rw-r--r--   1 root supergroup      35 2023-05-28 14:59 /test.txt
root@liushuai2020212267:~#

```

Browse Directory

Show entries
Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	<input type="checkbox"/>
<input type="checkbox"/>	-rw-r--r--	root	supergroup	6.59 KB	Apr 30 22:11	3	128 MB	ceshi.txt	<input type="checkbox"/>
<input type="checkbox"/>	drwxr-xr-x	root	supergroup	0 B	Apr 30 17:18	0	0 B	hbase	<input type="checkbox"/>
<input type="checkbox"/>	drwxr-xr-x	root	supergroup	0 B	May 08 15:21	0	0 B	logfile	<input type="checkbox"/>
<input type="checkbox"/>	-rw-r--r--	root	supergroup	3.03 KB	May 22 11:06	3	128 MB	word.txt	<input type="checkbox"/>
<input type="checkbox"/>	drwxr-xr-x	root	supergroup	0 B	May 08 12:40	0	0 B	wordcount	<input type="checkbox"/>

4.3 样例Scala Spark代码实现了一个简单的单词计数功能：

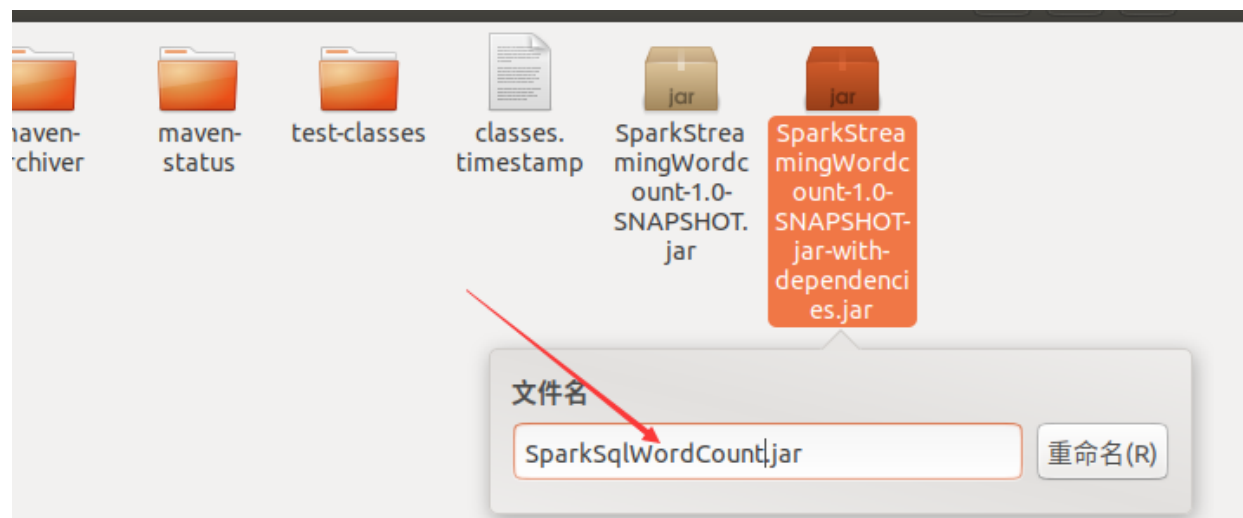
1. 创建一个名为“SparkSQL”的SparkSession实例：sparkSession是Spark 2.0版本后推出的一个统一的入口，可以用于创建DataFrame、Dataset、RDD以及执行Spark SQL等操作。
2. 读取HDFS文本文件“word.txt”，并将其转换为一个Dataset[String]类型的对象fileDS。
3. 对每一行进行按照空格切分并压平，得到一个新的Dataset[String]类型的对象wordDS。
4. 将wordDS注册为一个临时表t_word，并使用Spark SQL对该表进行查询并按照单词出现次数降序排序，最后输出结果。

整个过程可以简单概括为：读取文件、切分单词、进行单词计数、输出结果。

4.4 下面为实验过程记录：

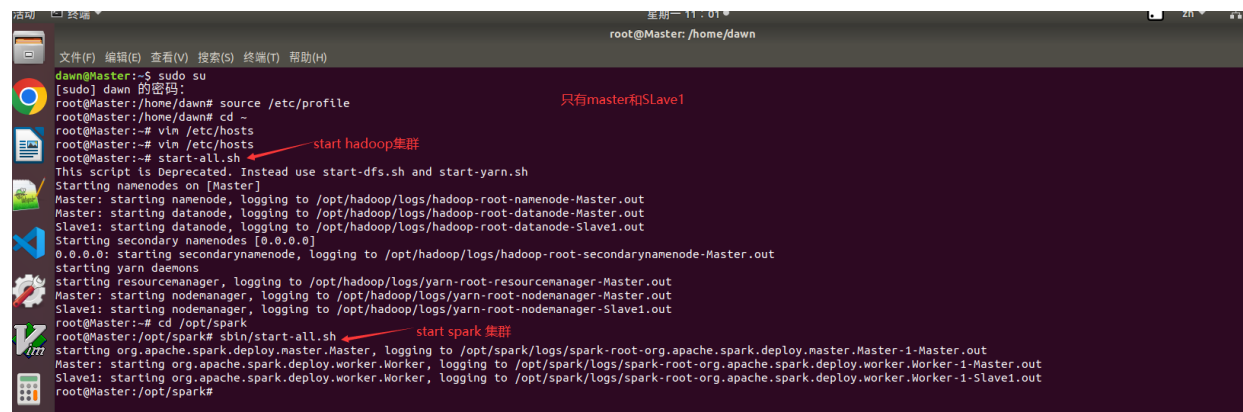
- 继续使用实验四的项目环境备份 pom.xml 文件后更改为实验五的配置，下载相关依赖打包样例程序为 jar，配置等同实验四，注意，样例程序中没有import Spark SQL中的Dataset类。

```
import org.apache.spark.sql.Dataset
```



- 如图打包后重命名移动到 /opt/spark 目录下。

1. 启动hdfs、spark集群



成功启动截图：

```
root@Master:/opt/spark# jps
9154 Worker
5253 -- process information unavailable
7990 NameNode
8646 NodeManager
5497 -- process information unavailable
8139 DataNode
9004 Master
8526 ResourceManager
9326 Jps
8334 SecondaryNameNode
```

- 新建一个word.txt并上传到hdfs

```
dawn@Master: ~/桌面/big data
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
lib/spawn.c      Code skeleton of the spawn library call.
You should run the pingpong, primes, and forktree test cases from lab 4 again af
ter merging in the new lab 5 code. You will need to comment out the ENV_CREATE(f
s_fs) line in kern/init.c because fs/fs.c tries to do some I/O, which JOS does n
ot allow yet. Similarly, temporarily comment out the call to close_all() in lib/
exit.c; this function calls subroutines that you will implement later in the lab
, and therefore will panic if called. If your lab 4 code doesn't contain any bug
s, the test cases should run fine. Don't proceed until they work. Don't forget t
o un-comment these lines when you start Exercise 1.

If they don't work, use git diff lab4 to review all the changes, making sure the
re isn't any code you wrote for lab4 (or before) missing from lab 5. Make sure t
hat lab 4 still works.

Lab Requirements
As before, you will need to do all of the regular exercises described in the lab
and at least one challenge problem. Additionally, you will need to write up bri
ef answers to the questions posed in the lab and a short (e.g., one or two parag
raph) description of what you did to solve your chosen challenge problem. If you
implement more than one challenge problem, you only need to describe one of the
m in the write-up, though of course you are welcome to do more. Place the write-
up in a file called answers-lab5.txt in the top level of your lab5 directory bef
ore handing in your work.
:wq
```

- upload Spark程序提交到spark集群中运行:

```
bin/spark-submit --class org.example.SqlWordCount --master spark://Master:7077
SparkSqlWordCount.jar
```

- 运行结果:

```
23/05/22 11:10:49 INFO handler.ContextHandler: Started o.s.j.s.ServletCont
23/05/22 11:10:49 INFO cluster.StandaloneSchedulerBackend: SchedulerBacken

+-----+-----+
| value|count|
+-----+-----+
| the| 25|
| to| 16|
| in| 13|
| file| 12|
| and| 10|
| a| 10|
| you| 10|
| of| 9|
| lab| 9|
| will| 7|
| | 6|
| athena%| 6|
| that| 6|
| git| 5|
| need| 5|
| lab5| 5|
| your| 5|
| for| 5|
| this| 5|
| system| 5|
+-----+-----+

only showing top 20 rows
```

- workers和executors截图:

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, 1 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

Workers (2)

Worker Id	Address
worker-20230521200120-192.168.170.163-34249	192.168.170.163:34249
worker-20230522110119-192.168.170.242-43031	192.168.170.242:43031

Executors

[Show Additional Metrics](#)

Summary

	▲ RDD Blocks ▼	▼ Storage Memory ▼	▼ Disk Used ▼	▼ Cores ▼	▼ Active Tasks ▼	▼ Failed Tasks ▼	▼ Complete Tasks ▼	▼ Total Tasks ▼
Active(3)	0	74.5 KiB / 1.1 GiB	0.0 B	14	1	0	0	1
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0
Total(3)	0	74.5 KiB / 1.1 GiB	0.0 B	14	1	0	0	1

Executors

Show entries

Executor ID	▲ Address ▼	▼ Status ▼	▼ RDD Blocks ▼	▼ Storage Memory ▼	▼ Disk Used ▼	▼ Cores ▼	▼ Active Tasks ▼	▼ Failed Tasks ▼	▼ Complete Tasks ▼	▼ Total Tasks ▼
0	192.168.170.242:39205	Active	0	17.8 KiB / 366.3 MiB	0.0 B	12	1	0	0	1
driver	Master:42777	Active	0	56.7 KiB / 366.3 MiB	0.0 B	0	0	0	0	0
1	192.168.170.163:40823	Active	0	0.0 B / 366.3 MiB	0.0 B	2	0	0	0	0

总结

本次实验，在实验四的基础上，我们使用Spark中的Sql模块进行数据处理和测试，最后也在spark集群上运行了Sql模块进行wordcount。

总体来说，Spark SQL是Spark用来处理结构化数据的一个模块，它提供了2个编程抽象：DataFrame和DataSet，并且可以利用Spark的分布式计算能力，可以处理海量数据。它将Hive SQL转换成MapReduce然后提交到集群上执行，大大简化了编写MapReduce程序的复杂性，由于MapReduce这种计算模型执行效率比较慢。所以Spark SQL的应运而生，它将Spark SQL转换成RDD，然后提交到集群执行。