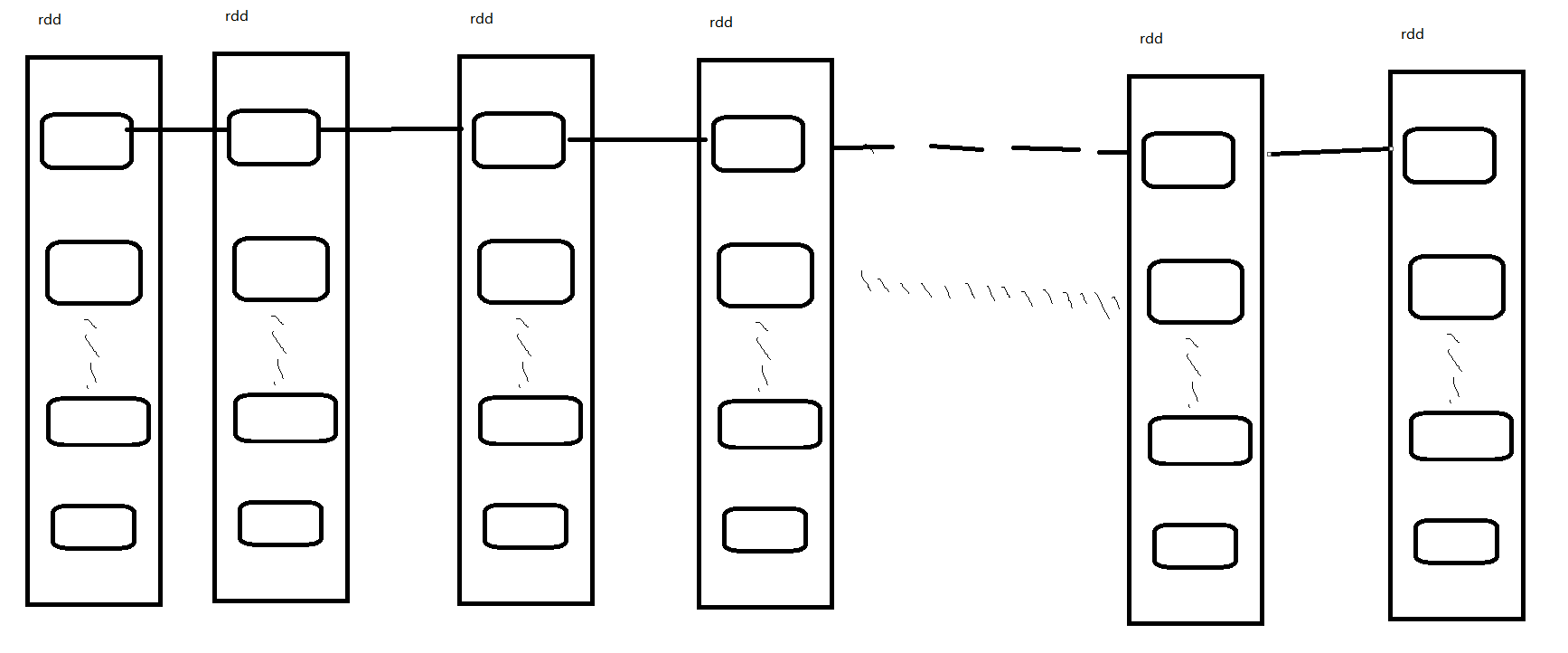
Spark检查点、stage、sparksql

Spark检查点checkpoint

1. 实现数据备份的高可靠
2. 方便管理操作数据

为什么要使用checkpoint: spark缓存在内存或者磁盘，但是可靠性不高，一旦机器坏掉，spark虽然能够根据dag（rdd依赖关系）进行恢复数据，但是这样代价会比较高，尤其是机器学习、深度学习、图计算的场景。那么这种情况可以使用checkpoint，讲数据备份在HDFS文件系统里面，从而实现数据的高可靠，而且方便管理。



如何实现checkpoint:

1. 首先在sparkcontext里面设置checkpoint后数据存放地址

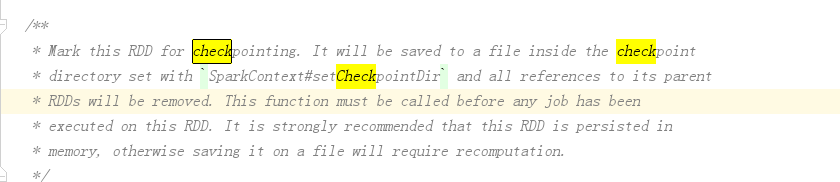
sc.setCheckpointDir("hdfs://bigdata01:9000/checkpoint20170211")

2.然后在需要checkpoint的rdd上面调用 checkpoint

Rdd.checkpoint

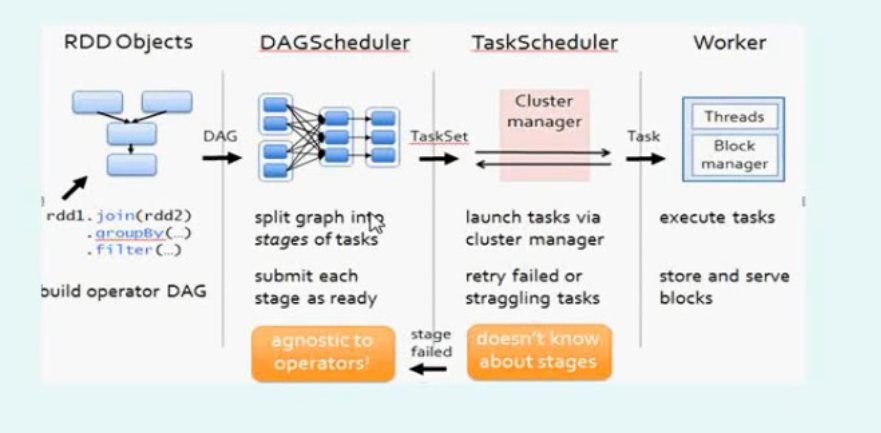
3.等到调用action方法后，checkpoint的rdd数据会存放到文件系统里面去

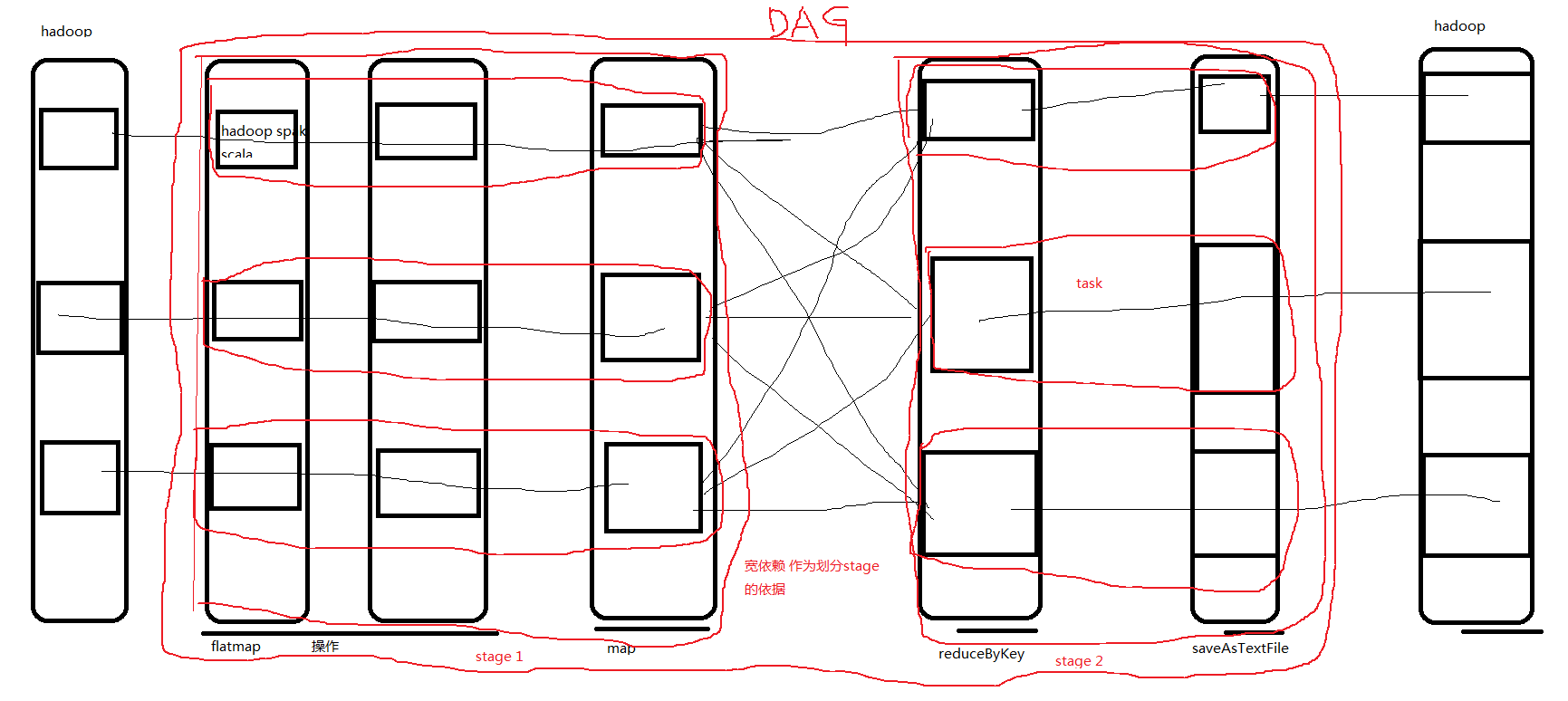
查看源码：



问题：请说说你对checkpoint的理解 ？

**Spark 任务提交流程**





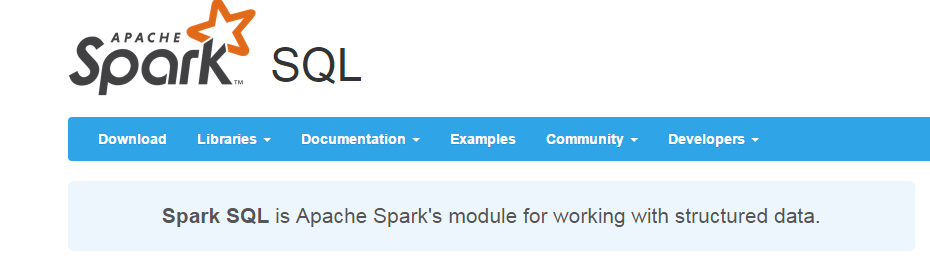
Rdd依赖关系参考文档Spark RDD.docx

DAG的生成参考文档Spark RDD.docx

后续看源码进阶提示：在spark-core里面找到各个流水线上的类

Spark Sql

官方文档：<http://spark.apache.org/sql/>



Spark SQL是Spark用来处理结构化数据的一个模块，它提供了一个编程抽象叫做DataFrame并且作为分布式SQL查询引擎的作用。

为什么要学习sparksql ?

1.易整合



1. 统一的数据访问方式



1. 兼容Hive



1. 标准的数据连接



## **DataFrames**

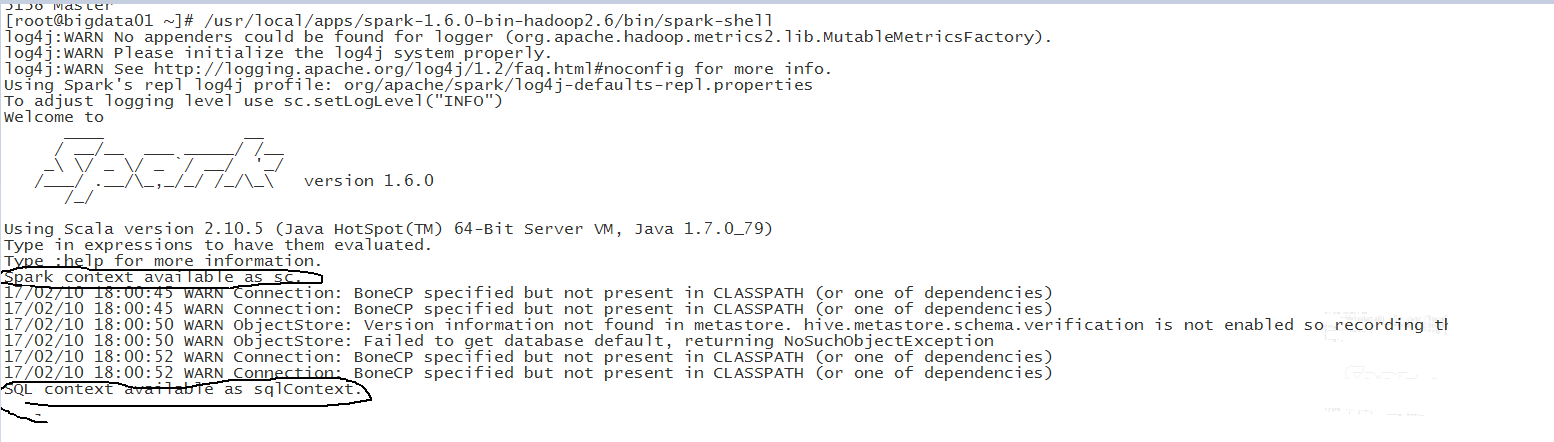
### 什么是DataFrames

与RDD类似，DataFrame也是一个分布式数据容器。然而DataFrame更像传统数据库的二维表格，除了数据以外，还记录数据的结构信息，即schema。同时，与Hive类似，DataFrame也支持嵌套数据类型（struct、array和map）。从API易用性的角度上 看，DataFrame API提供的是一套高层的关系操作，比函数式的RDD API要更加友好，门槛更低。由于与R和Pandas的DataFrame类似，Spark DataFrame很好地继承了传统单机数据分析的开发体验。



**创建dataframe:**

 在Spark SQL中SQLContext是创建DataFrames和执行SQL的入口，在spark-1.5.2中已经内置了一个**sqlContext**



1. 在本地创建一个文件，有三列，分别是id、name、age，用空格分隔，然后上传到hdfs上

hdfs dfs -put person.txt /

2.在spark shell执行下面命令，读取数据，将每一行的数据使用列分隔符分割

val lineRDD = sc.textFile("hdfs://bigdata01:9000/person.txt").map(\_.split(" "))

3.定义case class（相当于表的schema）

case class Person(id:Int, name:String, age:Int)

1. 将RDD和case class关联

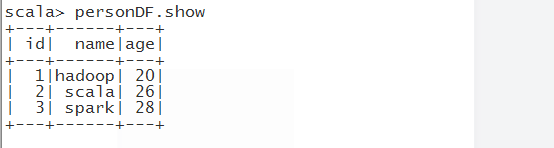
val personRDD = lineRDD.map(x => Person(x(0).toInt, x(1), x(2).toInt))

1. 将RDD转换成DataFrame

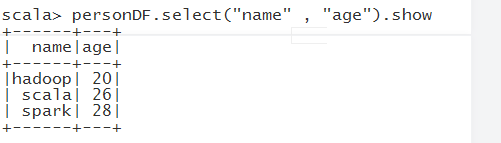
val personDF = personRDD.toDF

6.对DataFrame进行处理

personDF.show



1. 通过DataFrame查询2个字段(DSL风格)



### DSL风格语法（dataframe常用操作）

//查看DataFrame中的内容

personDF.show

//查看DataFrame部分列中的内容

personDF.select(personDF.col("name")).show

personDF.select(col("name"), col("age")).show

personDF.select("name").show

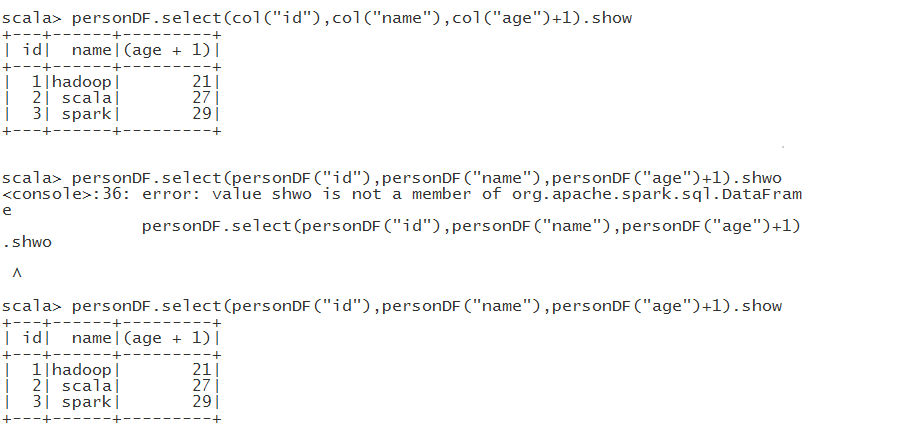
//打印DataFrame的Schema信息

personDF.printSchema

//查询所有的name和age，并将age+1

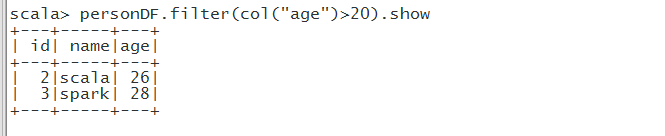
personDF.select(col("id"), col("name"), col("age") + 1).show

personDF.select(personDF("id"), personDF("name"), personDF("age") + 1).show



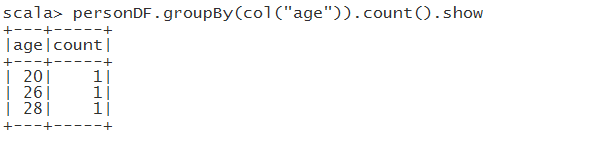
//过滤age大于等于20的

personDF.filter(col("age") >= 20).show



//按年龄进行分组并统计相同年龄的人数

personDF.groupBy("age").count().show()



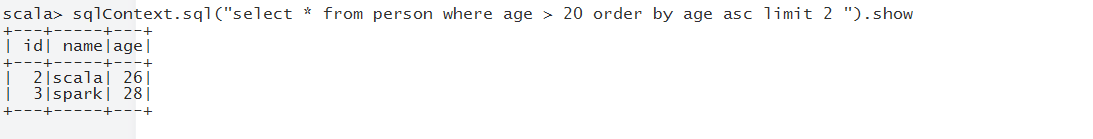
### SQL风格语法

如果想使用SQL风格的语法，需要将DataFrame注册成表

personDF.registerTempTable("person")

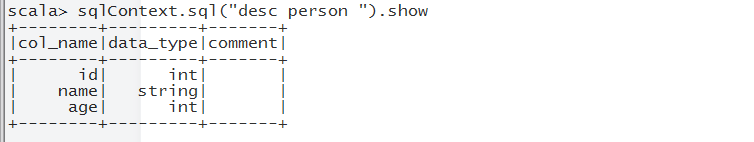
//查询年龄最大的前两名

sqlContext.sql("select \* from t\_person order by age desc limit 2").show



//显示表的Schema信息

sqlContext.sql("desc person").show



# 以编程方式执行Spark SQL查询

# **编写Spark SQL查询程序**

前面我们学习了如何在Spark Shell中使用SQL完成查询，现在我们来实现在自定义的程序中编写Spark SQL查询程序。首先在maven项目的pom.xml中添加Spark SQL的依赖

<dependency>  
 <groupId>org.apache.spark</groupId>  
 <artifactId>spark-sql\_2.10</artifactId>  
 <version>1.6.0</version>  
</dependency>

1. 第一种写法:通过反射推断Schema

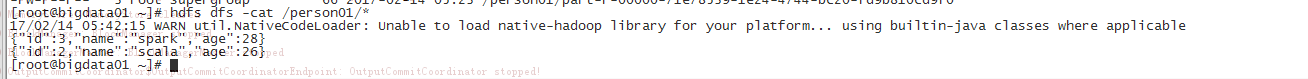
object SparkSQLLearn {  
 def main(args: Array[String]) {  
 // 这是用来设置本驱动程序的信息, 使用本地模式，本次演示的本地模式就是windows环境,启动2个线程  
 val sparkConf = new SparkConf().setAppName("SparkSQLLearn").setMaster("local[2]")  
 //负责和集群通信  
 val sparkContext = new SparkContext(sparkConf)  
 // spark sql是建立在sparkCores上面的，那么自然而然需要使用到sparkContext进行通信  
 val sqlContext = new SQLContext(sparkContext)  
  
 // 将数据进行切分  
 val rdd = sparkContext.textFile("G:\\spark\\sparksql\\person.txt").map(\_.split(" "))  
  
 //将样例类的信息放在rdd里面，等到触发action算子的时候执行  
 val personRDD = rdd.map(x => Person(x(0).toInt,x(1).toString,x(2).toInt))  
 //导入对从rdd到dataframe的转换支持  
 import sqlContext.implicits.\_  
 // 将rdd转换成dataframe  
 val personDf = personRDD.toDF  
 //将dataframe注册成为临时表  
 personDf.registerTempTable("person")  
 // 使用sql的方式  
 sqlContext.sql("select \* from person where age> 25 order by age desc limit 2").show()  
 sparkContext.stop()  
}  
}  
//case class一定要放到外面 , 必须要定义成为样例类  
case class Person(id: Int, name: String, age: Int)

打包到集群的方式：

def main(args: Array[String]) {  
 // 这是用来设置本驱动程序的信息, 使用本地模式，本次演示的本地模式就是windows环境  
 val sparkConf = new SparkConf().setAppName("SparkSQLLearn")  
 //负责和集群通信  
 val sparkContext = new SparkContext(sparkConf)  
 // spark sql是建立在sparkCores上面的，那么自然而然需要使用到sparkContext进行通信  
 val sqlContext = new SQLContext(sparkContext)  
  
 // 将数据进行切分  
 val rdd = sparkContext.textFile("hdfs://bigdata01:9000/person.txt").map(\_.split(" "))  
  
 //将样例类的信息放在rdd里面，等到触发action算子的时候执行  
 val personRDD = rdd.map(x => Person(x(0).toInt,x(1).toString,x(2).toInt))  
 //导入对从rdd到dataframe的转换支持  
 import sqlContext.implicits.\_  
 // 将rdd转换成dataframe  
 val personDf = personRDD.toDF  
 //将dataframe注册成为临时表  
 personDf.registerTempTable("person")  
 // 使用sql的方式  
 val df = sqlContext.sql("select \* from person where age> 25 order by age desc limit 2")  
 df.show()  
 df.write.json("hdfs://bigdata01:9000/person01")  
 sparkContext.stop()  
}  
}  
//case class一定要放到外面 , 必须要定义成为样例类  
//case class Person(id: Int, name: String, age: Int)

提交的指令：

/usr/local/apps/spark-1.6.0-bin-hadoop2.6/bin/spark-submit --master spark://bigdata01:7077 --class com.youe.spark.SparkSQLLearn /root/hello-spark-1.0.jar



### 通过StructType直接指定Schema

**d**ef main(args: Array[String]): Unit = {  
 //这是设置本驱动程序的信息 , 设置为本地模式，本次演示是在windows上面跑动，并启动4个线程  
 val sparkConf = new SparkConf().setAppName("SparkSqlContext").setMaster("local[4]")  
 //负责和集群通讯  
 val sparkContext = new SparkContext(sparkConf)  
 // 这是sql环境，sparksql会用到sparkcores 所以需要传入sparkContext和集群进行通信  
 val sparkSqlContext = new SQLContext(sparkContext)  
 //对每行数据进行切分  
 val rdd = sparkContext.textFile("G:\\spark\\sparksql\\person.txt").map(line => line.split(" "))  
 // 定义schema 表的元信息  
 val schema = StructType(List(  
 StructField("id" , IntegerType , false),  
 StructField("name" ,StringType,true),  
 StructField("age" ,IntegerType,true)  
 ))  
 // 将rdd转变为rowRDD  
 val rowRDD = rdd.map(x => Row(x(0).toInt , x(1).toString,x(2).toInt))  
 val personDF = sparkSqlContext.createDataFrame(rowRDD,schema)  
 personDF.registerTempTable("person")  
 val df = sparkSqlContext.sql("select \* from person order by age asc")  
 df.show()  
 sparkContext.stop()  
}

作业

1. 将课堂上的场景全部操作一遍（demo部分先照着做、理解，再背着写出来，再总结；源码部分要点进去看）
2. 理解stage依赖划分、宽依赖、窄依赖、DAG、任务task之间的关系以及过程

3. Sparksql、dataframe熟悉，能够使用