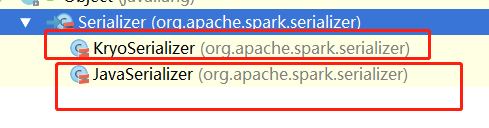
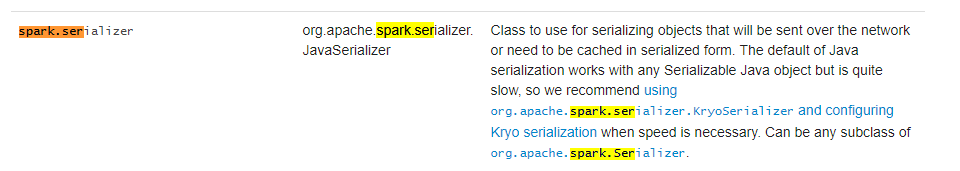
# 序列化的类补充：



<http://spark.apache.org/docs/latest/configuration.html>



默认使用的是java的序列化：

|  |
| --- |
| **class** TestDemo **object** TestDemo {  **def** main(args: Array[String]): Unit = {  **val** conf = **new** SparkConf()  conf.set(**"spark.serializer"**, **"org.apache.spark.serializer.KryoSerializer"**)  *// 如果使用该序列化方式，必须注册类* conf.registerKryoClasses(*Array*(*classOf*[TestDemo]))  **val** sc = **new** SparkContext(conf)  sc.stop()  } } |

# zookeeper快速上手

## 简介

zookeeper 是一个一致性的协调服务。 记录状态信息，维护共享数据。状态切换。

spark HA 集群 维护spark master的状态

kafka 依然需要zookeeper

xxxx

zookeeper存储的数据都是 k-v 类型的。

每一个数据节点称之为 znode

/ab 数据

zookeeper的集群规模，是奇数台。

上千台规模的集群， 3-5台即可。

## lead选举机制

端口：2181:2888:3888

2181： 客户端连接端口

2888： 通信端口

3888： 选举端口

假定3台节点：

lead选举的机制： ZAB协议 Paxos 算法

64 高低位

## zookeeper的集群搭建

必须要做的：各server节点的时间同步

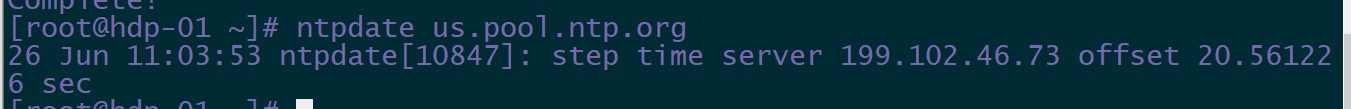
spark on yarn 中 yarn的时间，也是要求同步的。

date -s 2018/06/26

date -s 11:01:00

# ntpdate 时间同步的服务器

没有命令： yum -y install ntpdate



|  |
| --- |
| ntpdate us.pool.ntp.org 该命令作用：利用ntpdate同步标准时间  1.cn.pool.ntp.org  2.cn.pool.ntp.org  3.cn.pool.ntp.org  0.cn.pool.ntp.org  cn.pool.ntp.org  tw.pool.ntp.org  0.tw.pool.ntp.org  1.tw.pool.ntp.org  2.tw.pool.ntp.org  3.tw.pool.ntp.org |

1,看机器是否能连通外网。 2，多尝试几次。

官网：

<http://zookeeper.apache.org/releases.html>

zookeeper 3.4.6 版本

server： hdp-01 hdp-02 hdp-03

解压：

[root@hdp-01 ~]# tar -zxvf zookeeper-3.4.6.tar.gz -C apps/

修改conf目录下的zoo\_sample.cfg

# mv zoo\_sample.cfg zoo.cfg

修改配置文件：

dataLogDir=/root/zklogdata

|  |
| --- |
| dataDir=/root/zkdata  server.1=hdp-01:2888:3888  server.2=hdp-02:2888:3888  server.3=hdp-03:2888:3888 |

分发安装包：

|  |
| --- |
| cd /root/apps  for i in 2 3;do scp -r zookeeper-3.4.6/ hdp-0$i:$PWD ;done |

在**每一台server节点**上：

创建一个数据目录/root/zkdata

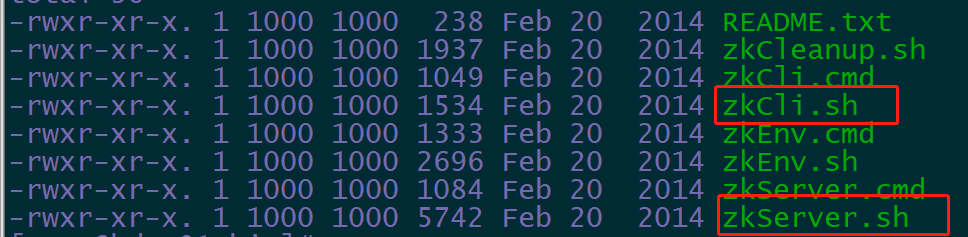
mkdir /root/zkdata

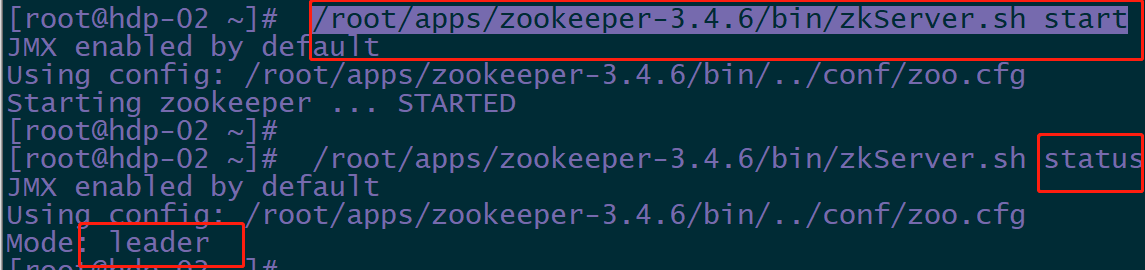
在每一台server中， 的/root/zkdata中创建一个文件，myid，值分别对应着配置文件中的id。

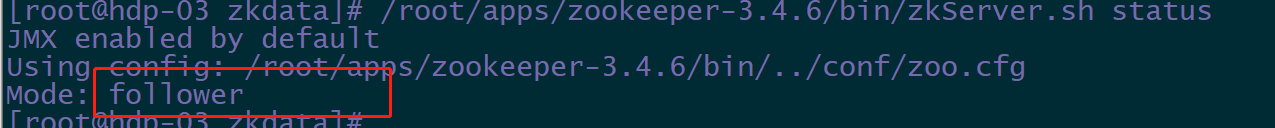
|  |
| --- |
| echo 1 > /root/zkdata/myid  ssh hdp-02 "echo 2 > /root/zkdata/myid"  ssh hdp-03 "echo 3 > /root/zkdata/myid" |

验证每一台机器上的myid文件是否有数据。

### 启动：

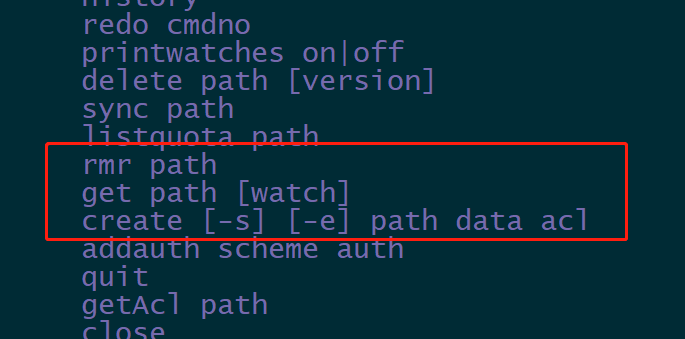








直接输入help查看帮助信息：



# sparkHA的集群

spark standalone的HA集群

HA : high avaliable 高可用 7\*24 H 不会挂掉

spark的standalone的集群： 单个Master的 多个worker 单点故障

搞2个master

zookeeper集群。

spark-env.sh 添加 spark的恢复模式为zookeeper模式即可。

把之前的host注释掉，写新的：

export SPARK\_DAEMON\_JAVA\_OPTS="-Dspark.deploy.recoveryMode=ZOOKEEPER -Dspark.d

eploy.zookeeper.url=hdp-01:2181,hdp-02:2181,hdp-03:2181 -Dspark.deploy.zookeep

er.dir=/spark"



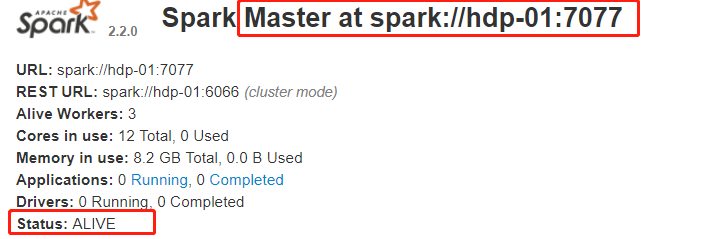
把配置分发到其他的节点中：

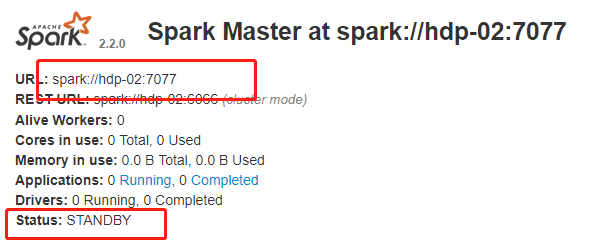
[root@hdp-01 conf]# for i in 2 3 4 ; do scp spark-env.sh hdp-0$i:$PWD ;done

启动：2个Master。

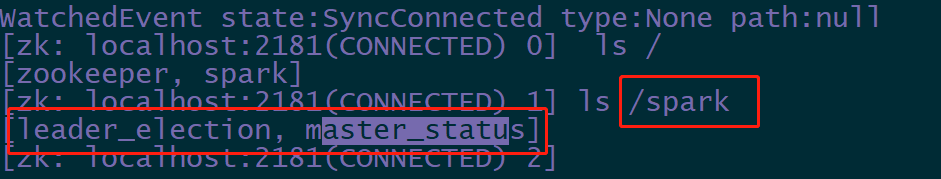
当正确启动集群之后， 再单独启动一个Master即可。

1. 在hdp-01机器上，启动 start-all.sh
2. 在hdp-02上，单独启动master: start-master.sh



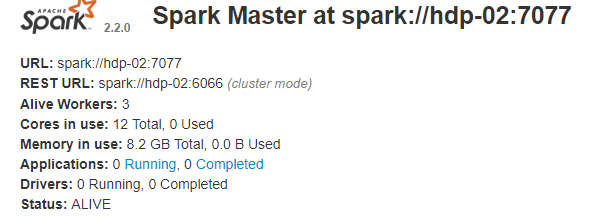


查看zookeeper中的注册节点数据：



验证HA：

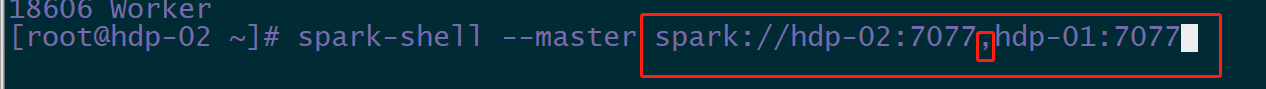
杀掉hdp-01上的Master。查看hdp-02上的Master状态。



新提交的任务，处于等待状态。

不支持抢占，当再次启动hdp-01的master的时候，处于standby状态。

使用方式： 指定多个master的url地址。



# sparksql

## 是什么？



spark sql是spark中处理结构化数据的一个模块。spark sql 也是一个**sql解析引擎**。

“” -🡪 spark 任务 运行在spark的集群中。

## sparksql 的优点

### 易整合

1. SQL DataFrame API 多种开发语言都支持

### 统一的数据访问形式

json parquet jdbc

.read().json()

.read().jdbc()

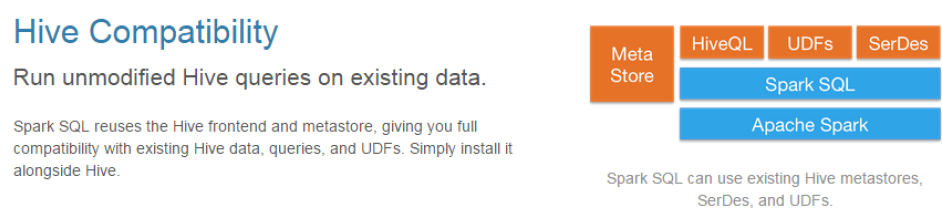
write

### 兼容hive

hive -🡪 mapreduce

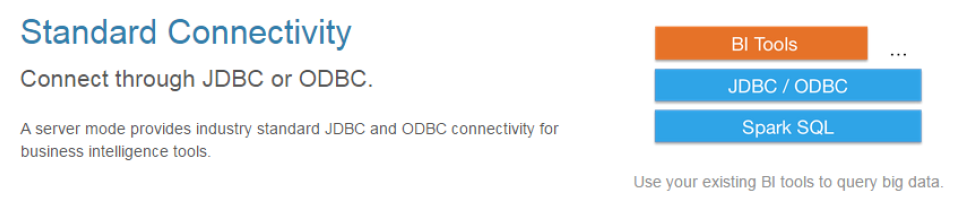
hive ----》 spark 程序运行

HQL



### 标准的数据连接

BI Tools 企业的应用服务器。



# sparkSQL编程

## spark sql的发展史

DataFrame 编程API spark1.3

Dataset 编程API spark1.6 引入

Dataset + DataFrame = Dataset spark 2.x 之后，合并了

DataFame = Dataset[Row]

spark core中 RDD ---》 基于RDD的算子

SparkSQL 中 DataFrame Dataset

## DataFrame

SparkSql 中的DataFrame 就相当于 SparkCore中的RDD

DataFrame依然是一个抽象的数据集。基于RDD之上的更高层次的封装。

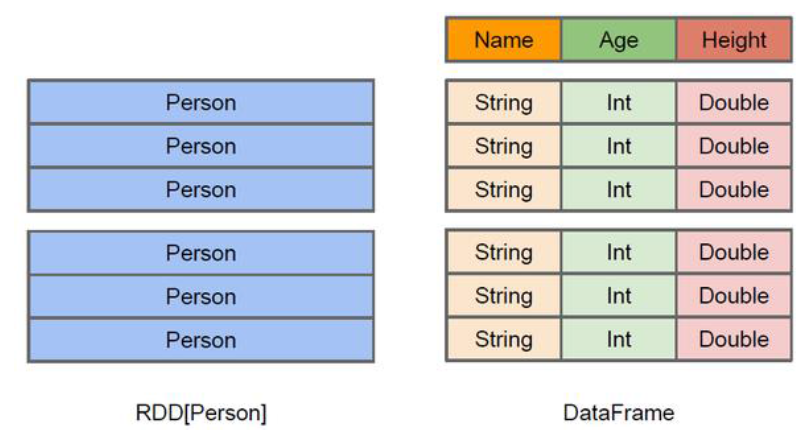
DataFrame是由 结构的。

表结构信息： 字段名称 字段的类型 是否为空等信息。

schema

DataFrame是带着schema信息的RDD。

DataFrame = RDD + schema



直接从一个rdd中，指定schema ===》 DataFrame

## sparkshell中操作dataframe

person.txt -🡪 RDD[String] -🡪 case class RDD[Person] -🡪 DataFrame

|  |
| --- |
| scala> val rdd1 = sc.textFile("hdfs://hdp-01:9000/person.txt")  rdd1: org.apache.spark.rdd.RDD[String] = hdfs://hdp-01:9000/person.txt MapPartitionsRDD[1] at textFile at <console>:24  scala> case class Person(name:String,age:Int,fv:Int)  defined class Person  scala> val rdd2 = rdd1.map(\_.split(" ")).map(arr => Person(arr(0),arr(1).toInt,arr(2).toInt))  rdd2: org.apache.spark.rdd.RDD[Person] = MapPartitionsRDD[3] at map at <console>:28  scala> val pdf = rdd2.toDF()  pdf: org.apache.spark.sql.DataFrame = [name: string, age: int ... 1 more field]  scala> pdf.schema  res0: org.apache.spark.sql.types.StructType = StructType(StructField(name,StringType,true), StructField(age,IntegerType,false), StructField(fv,IntegerType,false))  scala> pdf.printSchema  root  |-- name: string (nullable = true)  |-- age: integer (nullable = false)  |-- fv: integer (nullable = false)  scala> val pdf2 = pdf.select("age")  pdf2: org.apache.spark.sql.DataFrame = [age: int]  scala> pdf2.show()  +---+  |age|  +---+  | 20|  | 23|  | 25|  +---+  scala> val pdf3 = pdf.where("age > 22") |

## IDEA中操作DataFrame

### 配置maven工程：

导入sparksql的依赖jar包

|  |
| --- |
| *<!--导入spark sql的依赖jar包-->* <**dependency**>  <**groupId**>org.apache.spark</**groupId**>  <**artifactId**>spark-sql\_2.11</**artifactId**>  <**version**>${spark.version}</**version**> </**dependency**> |

spark cores RDD SparkContext

sparksql DataFrame SQLContext

|  |
| --- |
| **def** main(args: Array[String]): Unit = {   **val** conf = **new** SparkConf()  .setMaster(**"local[\*]"**)  .setAppName(**this**.getClass.getSimpleName)  **val** sc = **new** SparkContext(conf)   *// 操作sql专用实例* **val** sqlContext = **new** SQLContext(sc)   *// 导入sql实例上的隐式转换* **import** sqlContext.implicits.\_    **val** file = sc.textFile(**"person.txt"**)   *// 数据预处理* **val** personRdd: RDD[Person] = file.map(t => {  **val** split = t.split(**" "**)  **val** name1 = split(0)  **val** age1 = split(1).toInt  **val** fv1 = split(2).toInt  *Person*(name1, age1, fv1)  })   *// 需要导入隐式转换 DataFrame* **val** pdf: DataFrame = personRdd.toDF()   *// 表结构就是通过反射，利用类的字段来获取的。* pdf.printSchema()  */\*\*  \* root  \* |-- name: string (nullable = true)  \* |-- age: integer (nullable = false)  \* |-- fv: integer (nullable = false)  \*/   // 只有获取到DataFrame的一个对象，才能使用DataFrame的API来执行sparksql的操作   // 有2种语法风格的操作。  // sql语法  // 先必须把dataframe 注册成临时表   /\*pdf.registerTempTable("t\_person")   val sql: DataFrame = sqlContext.sql("select age from t\_person where age > 25")   // 打印结果  sql.show() // 默认显示20行\*/   // DSL语法 DSL（Domain Specified Language）   // 选择 select* pdf.select(pdf.col(**"name"**),pdf.col(**"age"**),pdf.col(**"fv"**))  pdf.select(pdf(**"name"**))  pdf.select(**"name"**)    **val** result2 = pdf.where(**"age > 25"**).select(**"age"**)   result2.show()  sc.stop()  } }   **case class** Person(name: String, age: Int, fv: Int) |

## DataFrame

### 获取DataFrame:

1,RDD[Person] 利用Person的类结构 来反射获取schema信息的。 toDF

2,自定义schema信息。

|  |
| --- |
| *// 需要导入隐式转换 DataFrame* **val** pdf: DataFrame = personRdd.toDF()  *println*(pdf.schema)  *// 表结构就是通过反射，利用类的字段来获取的。* pdf.printSchema() */\*\*  \* root  \* |-- name: string (nullable = true)  \* |-- age: integer (nullable = false)  \* |-- fv: integer (nullable = false)  \*/* |

StructType(StructField(name,StringType,true), StructField(age,IntegerType,false), StructField(fv,IntegerType,false))

### 自定义shema信息

|  |
| --- |
| **val** conf = **new** SparkConf()  .setMaster(**"local[\*]"**)  .setAppName(**this**.getClass.getSimpleName) **val** sc = **new** SparkContext(conf)  *// 操作sql专用实例* **val** sqlContext = **new** SQLContext(sc)  **val** file: RDD[String] = sc.textFile(**"person.txt"**)  **val** rowRdd: RDD[Row] = file.map(t => {  **val** split = t.split(**" "**)  *Row*(split(0), split(1).toInt, split(2).toInt) })   *// createDataFrame(rowRDD: RDD[Row], schema: StructType) // sqlContext.createDataFrame()* **val** schema = *StructType*(  *List*(  *// 是否为空这个字段可以省略  StructField*(**"name2"**, StringType),  *StructField*(**"age2"**, IntegerType),  *StructField*(**"fv"**, IntegerType, **false**)  ))  **val** pdf: DataFrame = sqlContext.createDataFrame(rowRdd, schema)  *println*(pdf.schema)  *// 2种操作风格 SQL DSL* sc.stop() |

### DataFrame版本的wordcount

|  |
| --- |
| **def** main(args: Array[String]): Unit = {   **val** conf = **new** SparkConf()  .setMaster(**"local[\*]"**)  .setAppName(**this**.getClass.getSimpleName)  **val** sc = **new** SparkContext(conf)  **val** sqlContext = **new** SQLContext(sc)   *// 隐式转换* **import** sqlContext.implicits.\_   **val** file = sc.textFile(**"wc.txt"**)   *// 把数据用类封装* **val** wordRdd: RDD[Word] = file.flatMap(\_.split(**" "**)).map(*Word*(\_))    **val** wdf: DataFrame = wordRdd.toDF()    *// sql* wdf.registerTempTable(**"t\_word"**)  **val** result: DataFrame = sqlContext.sql(**"select word,count(\*) as cnts from t\_word group by word order by cnts desc limit 3"**)  *// result.printSchema()  // result.show()    // DSL* **val** result2 = wdf.groupBy(**"word"**).count()  result2.printSchema()   result2.show()    sc.stop()  } }   **case class** Word(word: String) |

### DataFrame操作的流程：

1. sparkSql 导入sparksql的依赖jar包
2. new SQLContext(sc)
3. 读数据进行预处理
   1. 定义case class rdd ----》 RDD[类] 导入隐式转换 toDF方法
   2. 自定义shema信息 rdd -🡪 RDD[Row] createDataFrame(rowRdd,schema)

4,得到一个DataFrame对象

5,调用API编程

* 1. sql语法 注册临时表 sqlContext.sql(“sql语句”)
  2. DSL语法 调用专用的方法

6，查看结果

resultDF.show()

## Dataset

### 简介

spark1.6版本引入的 spark2.x dataset 和 dataframe合并了

Dataset，和dataFrame类似，依然是一个分布式的数据集合。

dataset默认自带schema信息。

spark cores RDD SparkContext

sparksql DataFrame SQLContext

sparksql Dataset SparkSession

### SparkSession

spark2.x 之后，提供了一个新的编程API，SparkSession

|  |
| --- |
| *// 存在拿来使用，不存在，再创建 // spark session* **val** spark: SparkSession = SparkSession.*builder*()  *// local模式下2个参数配置* .master(**"local[\*]"**)  .appName(**this**.getClass.getSimpleName)  .getOrCreate() |

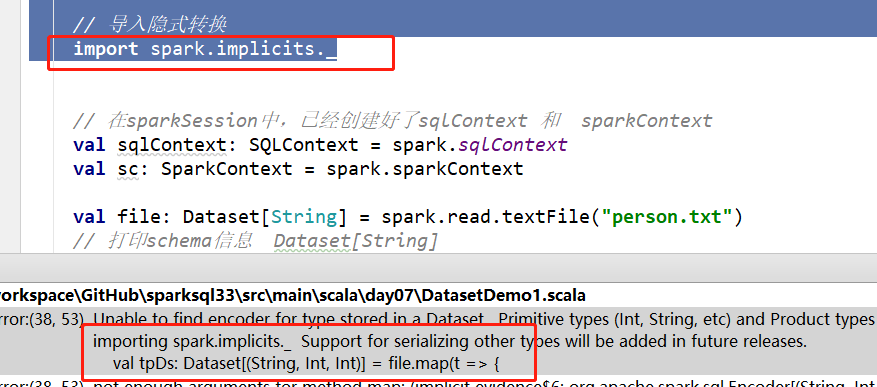
### Dataset的schema

|  |
| --- |
| **val** file: Dataset[String] = spark.read.textFile(**"person.txt"**) *// 打印schema信息 Dataset[String]* file.printSchema() */\*\*  \* root  \* |-- value: string (nullable = true)  \*/* **val** tpDs: Dataset[(String, Int, Int)] = file.map(t => {  **val** split = t.split(**" "**)  (split(0), split(1).toInt, split(2).toInt) })  tpDs.printSchema() */\*\*  \* root 名称同元组的取值方式 类型同 数据的类型  \* |-- \_1: string (nullable = true)  \* |-- \_2: integer (nullable = false)  \* |-- \_3: integer (nullable = false)  \*/* |

**type** DataFrame = Dataset[Row]

|  |
| --- |
| **val** pdf: DataFrame = tpDs.toDF(**"name"**,**"age"**,**"fv"**) **val** pdf2: Dataset[Row] = tpDs.toDF(**"name"**,**"age"**,**"fv"**) |

### SQL语法风格



解决方案：

import spark.implicits.\_ 注意：这里的spark是创建的SparkSession的实例名称。

|  |
| --- |
| *// 存在拿来使用，不存在，再创建  // spark session* **val** spark: SparkSession = SparkSession.*builder*()  *// local模式下2个参数配置* .master(**"local[\*]"**)  .appName(**this**.getClass.getSimpleName)  .getOrCreate()   *// 导入隐式转换* **import** spark.implicits.\_  *// 在sparkSession中，已经创建好了sqlContext 和 sparkContext* **val** sqlContext: SQLContext = spark.*sqlContext* **val** sc: SparkContext = spark.sparkContext   **val** file: Dataset[String] = spark.read.textFile(**"person.txt"**)  *// 打印schema信息 Dataset[String]* file.printSchema()  */\*\*  \* root  \* |-- value: string (nullable = true)  \*/* **val** tpDs: Dataset[(String, Int, Int)] = file.map(t => {  **val** split = t.split(**" "**)  (split(0), split(1).toInt, split(2).toInt)  })  *// tpDs.printSchema()  /\*\*  \* root 名称同元组的取值方式 类型同 数据的类型  \* |-- \_1: string (nullable = true)  \* |-- \_2: integer (nullable = false)  \* |-- \_3: integer (nullable = false)  \*/* **val** pdf: DataFrame = tpDs.toDF(**"name"**,**"age"**,**"fv"**)  **val** pdf2: Dataset[Row] = tpDs.toDF(**"name"**,**"age"**,**"fv"**)   pdf.printSchema()  *// sql  // 注册临时视图* pdf.createTempView(**"v\_person"**)   *// sql方法 写sql* **val** result1: DataFrame = spark.sql(**"select age from v\_person where age > 23 "**)   result1.show() |

### DSL语法操作

|  |
| --- |
| *// 存在拿来使用，不存在，再创建  // spark session* **val** spark: SparkSession = SparkSession.*builder*()  *// local模式下2个参数配置* .master(**"local[\*]"**)  .appName(**this**.getClass.getSimpleName)  .getOrCreate()   *// 导入隐式转换* **import** spark.implicits.\_   **val** file: Dataset[String] = spark.read.textFile(**"person.txt"**)   **val** tpDs: Dataset[(String, Int, Int)] = file.map(t => {  **val** split = t.split(**" "**)  (split(0), split(1).toInt, split(2).toInt)  })   **val** pdf: DataFrame = tpDs.toDF(**"name"**, **"age"**, **"fv"**)  *// ("select age from v\_person where age > 23 ")   // DSL 对应着sql中的一些操作  // select 选择* pdf.select(**"age"**,**"name"**) *//.show()   // where 是sql中的写法 filter where 调用的就是filter* pdf.where(**"age > 23"**) *// .show()   // 排序 orderBy sort 同一个API 默认是升序* pdf.orderBy(**$"age"** desc) *// .show()   // 分组聚合  // 统计次数 count(\*)* pdf.groupBy(**"age"**).count() *// .show()   // 导入函数* **import** org.apache.spark.sql.functions.\_  pdf.groupBy(**"age"**).agg(*count*(**"\*"**) as **"cnts"**) *// .show()* **val** sumfv = pdf.groupBy(**"age"**).agg(*sum*(**"fv"**))*// .show()* pdf.groupBy(**"age"**).agg(*min*(**"fv"**) as **"minfv"**).orderBy(**"minfv"**)*// .show()* sumfv.printSchema()  *// 如何修改字段的名称* sumfv.withColumnRenamed(**"sum(fv)"**,**"sumFV"**).show()   *//实际 SQl 语法风格* spark.stop() |

# Dataset和dataFrame都可以转化成rdd

|  |
| --- |
| **val** file: Dataset[String] = spark.read.textFile(**"person.txt"**)   **val** tpDs: Dataset[(String, Int, Int)] = file.map(t => {  **val** split = t.split(**" "**)  (split(0), split(1).toInt, split(2).toInt)  })   **val** rdd: RDD[(String, Int, Int)] = tpDs.*rdd* **val** pdf: DataFrame = tpDs.toDF(**"name"**, **"age"**, **"fv"**)   **val** rdd1: RDD[Row] = pdf.*rdd* rdd1.map(t=>{  *//* ***索引从0开始*** *// t.getString(0)* **val** name = t.getAs[String](**"name"**)  **val** age = t.getAs[Int](**"age"**)  (name,age)  }) |

# 今日重点：

zookeeper 集群安装

代码，还需要完成

Dataset 的wordcount

sparkSql：

DataFrame Dataset -🡪 rdd

dataframe shema信息如何设定

SQL DSL

dataset

SparkSession：

默认的schema ---》 自定义 toDF(“”)

DSL : 用方法实现sql中的基本操作。

join操作