一：java代码开发工程

1. 在mysql 中新建一个数据库名称为spark（可以使用navicat）
2. 在spark数据库中创建nameandage,nameandscore,nameagescore。

创建nameandage

DROP TABLE IF EXISTS `nameandage`;

CREATE TABLE `nameandage` (

`name` varchar(255) DEFAULT NULL,

`age` int(10) DEFAULT NULL

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

INSERT INTO `nameandage` VALUES ('Michael', '20');

INSERT INTO `nameandage` VALUES ('Andy', '17');

INSERT INTO `nameandage` VALUES ('Justin', '19');

创建nameandscore

DROP TABLE IF EXISTS `nameandscore`;

CREATE TABLE `nameandscore` (

`name` varchar(255) DEFAULT NULL,

`score` int(12) DEFAULT NULL

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

INSERT INTO `nameandscore` VALUES ('Michael', '98');

INSERT INTO `nameandscore` VALUES ('Andy', '95');

INSERT INTO `nameandscore` VALUES ('Justin', '68');

创建nameagescore

DROP TABLE IF EXISTS `nameagescore`;

CREATE TABLE `nameagescore` (

`name` varchar(255) DEFAULT NULL,

`age` int(12) DEFAULT NULL,

`score` int(12) DEFAULT NULL

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

1. 在pom.xml文件中添加mysql的jar包依赖，如果不是maven的方式直接将jar包加入到工程即可。

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.35</version>

</dependency>

1. Java代码如下（在本地模式下运行）

package com.dt.spark.SparkApps.sql;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import org.apache.spark.SparkConf;

import org.apache.spark.api.java.JavaPairRDD;

import org.apache.spark.api.java.JavaRDD;

import org.apache.spark.api.java.JavaSparkContext;

import org.apache.spark.api.java.function.Function;

import org.apache.spark.api.java.function.PairFunction;

import org.apache.spark.api.java.function.VoidFunction;

import org.apache.spark.sql.DataFrame;

import org.apache.spark.sql.DataFrameReader;

import org.apache.spark.sql.Row;

import org.apache.spark.sql.RowFactory;

import org.apache.spark.sql.SQLContext;

import org.apache.spark.sql.types.DataTypes;

import org.apache.spark.sql.types.StructField;

import org.apache.spark.sql.types.StructType;

import scala.Tuple2;

public class SparkSQLJDBC2Mysql {

public static void main(String[] args) {

SparkConf conf = new SparkConf().setMaster("local").setAppName("SparkSQLJDBC2Mysql");

JavaSparkContext sc = new JavaSparkContext(conf);

SQLContext sqlContext = new SQLContext(sc);

/\*\*

\* 1.通过format（“jdbc”） 的方式说明sparksql操作的数据通过jdbc获得

\* jdbc 后端一般是数据库例如mysql 。oracle

\* 2.通过DataFrameReader的option的方法把方位的数据库的信息传递进去

\* url：代表数据库的jdbc链接地址

\* datable 具体指哪个数据库

\* 3. dirver 部分是是sparksql访问数据库的具体的驱动完整的包名和类名

\*

\* 4. 关于jdbc 的驱动jar，可以放在spark的lib 目录下，也可以在使用sparksubmit的使用指定的jar

\* (打包的时候)

\*/

DataFrameReader reader =sqlContext.read().format("jdbc");

reader.option("url", "jdbc:mysql://master:3306/spark");

reader.option("dbtable", "nameandscore");

reader.option("driver", "com.mysql.jdbc.Driver");

reader.option("user", "root");

reader.option("password", "123456");

/\*\*

\* 在实际的企业级开发环境中我们如果数据库中数据规模特别大，例如10亿条数据，此时采用传统的db 去处理的话，一般需要对数据

\* 分成很多批次处理例如分成100批（首受限于单台server的处理能力）且实际处理可能会非常复杂，通过传统的J2ee 等基石很难或者很不方便实现处理方法，此时

\* 使用sparksql获得数数据库中的数据并进行分布式处理就可以非常好的解决该问题，但是sparksql 加载数据需要时间，所以一边会在sparksql和db 之间加一个缓冲层

\* 例如中间使用redis，可以把spark的处理速度提高甚至45倍。

\*

\*

\*

\*/

DataFrame nameandscoremysqlDataSourceDF=reader.load(); //基于hive数据库生成的DataFrame

nameandscoremysqlDataSourceDF.show();

reader.option("dbtable", "nameandage");

DataFrame nameandagemysqlDataSourceDF=reader.load();//基于hive2数据库生成的DataFrame

nameandagemysqlDataSourceDF.show();

/\*\*

\* 把datafram 转化成rdd 并基于rdd 进行join操作

\*/

JavaPairRDD<String, Tuple2<Integer, Integer>> resultRDD = nameandscoremysqlDataSourceDF.javaRDD().mapToPair(new PairFunction<Row, String, Integer>() {

private static final long serialVersionUID = 1L;

@Override

public Tuple2<String, Integer> call(Row row) throws Exception {

return new Tuple2<String, Integer>(row.getAs("name").toString(), (int) row.getInt(1));

}

}).join(nameandagemysqlDataSourceDF.javaRDD().mapToPair(new PairFunction<Row, String, Integer>() {

private static final long serialVersionUID = 1L;

@Override

public Tuple2<String, Integer> call(Row row) throws Exception {

return new Tuple2<String, Integer>(row.getAs("name").toString(), (int) row.getInt(1));

}

}));

JavaRDD<Row> reusltRowRDD = resultRDD.map(new Function<Tuple2<String,Tuple2<Integer,Integer>>, Row>() {

@Override

public Row call(Tuple2<String, Tuple2<Integer, Integer>> tuple) throws Exception {

// TODO Auto-generated method stub

return RowFactory.create(tuple.\_1, tuple.\_2.\_2,tuple.\_2.\_1 );

}

});

List<StructField> structFields = new ArrayList<StructField>();

structFields.add(DataTypes.createStructField("name", DataTypes.StringType, true));

structFields.add(DataTypes.createStructField("age", DataTypes.IntegerType, true));

structFields.add(DataTypes.createStructField("score", DataTypes.IntegerType, true));

//构建StructType，用于最后DataFrame元数据的描述

StructType structType =DataTypes.createStructType(structFields);

DataFrame personsDF = sqlContext.createDataFrame(reusltRowRDD, structType);

personsDF.show();

/\*\*

\* 1.当dataframe要把通过spark sql、core、ml等复杂操作后的数据写入数据库的时候 首先是权限的问题，必须

\* 确保数据库授权了当前操作spark sql的用户

\* 2.Dataframe要写数据到db 的时候，一般都不可以直接写进去，而是要转成RDD，通过RDD写数据到db中

\*/

personsDF.javaRDD().foreachPartition(new VoidFunction<Iterator<Row>>(){

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

@Override

public void call(Iterator<Row> t) throws Exception{

Connection conn2mysql=null;

Statement statement =null;

try {

conn2mysql=DriverManager.getConnection("jdbc:mysql://master:3306/spark","root","123456");

statement=conn2mysql.createStatement();

while(t.hasNext()){

String sql="insert into nameagescore (name,age,score) values (";

Row row=t.next();

String name=row.getAs("name");

int age= row.getInt(1);

int score =row.getInt(2);

sql+="'"+name+"',"+"'"+age+"',"+"'"+score+"')";

statement.execute(sql);

}

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}finally{

if(conn2mysql !=null){

conn2mysql.close();

}

if(statement !=null){

statement.close();

}

}

}

});

}

}