## Udf和缺失值处理

import org.apache.spark.sql.functions.{udf => normalUdf}

import scala.reflect.runtime.universe.TypeTag

/\*\*

\* editor: xuhao

\* date: 2018.03.20 09:00:00

\*/

/\*\*

\* 用于处理sparkSQL中创建udf函数时null的输出问题，使udf函数可以实现null => null的映射

\* ----

\* 描述：

\* sparkSQL中的udf有以下弊端：

\* 输入可能带有null值，但输出不行，数据会发生信息损失。

\* eg,

\* {{{

\* Long(with null) => Double中不能有null => null的映射：

\* udf((s: Long) => s match {

\* case null => null

\* case e => e / width

\* } // not compile

\* }}}

\* ----

\* 示例：

\* NullableFunctions的udf暗含了的null => null的映射，数据不会损失信息

\* eg,

\* {{{

\* NullableFunctions.udf((s: Long) => s / width)

\* }}}

\* ----

\* The main source idea by Martin Senne's answer on Stack OverFlow.

\*/

object NullableFunctions {

def udf[RT: TypeTag, A1: TypeTag](f: Function1[A1, RT])

: UserDefinedFunction = normalUdf[Option[RT],A1](

(i: A1) => i match {

case null => None

case s => Some(f(s))

})

def udf[RT: TypeTag, A1: TypeTag, A2: TypeTag](f: Function2[A1, A2, RT])

: UserDefinedFunction = normalUdf[Option[RT], A1, A2](

(i1: A1, i2: A2) => (i1, i2) match {

case (null, \_) => None

case (\_, null) => None

case (s1, s2) => Some(f(s1, s2))

})

}

## 开窗函数

一个例子

import org.apache.spark.sql.Row

import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}

val lst = Array.range(10, 20, 2).flatMap(i => Array("A", "B", "C", "D").map(x => (x, i)))

val rdd = sc.parallelize(lst).map(Row.fromTuple)

val bindDF = hqlc.createDataFrame(rdd, StructType(Array(StructField("category", StringType), StructField("count", IntegerType))))

bindDF.registerTempTable("tableName")

val sqlExpr = "SELECT `category`, `count` from (SELECT `category`, `count`, rank() OVER (PARTITION BY category ORDER BY count DESC) as rank FROM `tableName`) tmp WHERE rank <= 2"

val resultDF = hqlc.sql(sqlExpr)

resultDF.show()

## Schema的apply方法

Schema中有apply方法注意用，要比fieldIndex在索引省劲

## groupBy之后重命名的问题

1）使用sql语句最方便

2）如果不使用sql语句，下面的方式可以

DF.agg(*sum*(*col*("count")).alias("count"))