6.5 从RDD转换到DataFrame及数据库操作

슬라이드 1

今天我们要学习一个非常重要的主题——如何从RDD转换得到DataFrame。

这是Spark SQL（Spark SQL）编程中的核心技能之一。

我们会学习两种转换方式：

第一种是利用反射机制（reflection mechanism）推断RDD模式（schema），

第二种是使用编程方式定义RDD模式。这两种方法都很实用，大家要掌握好。

슬라이드 2 6.5.1 利用反射机制推断RDD模式

现在我们来看第一种方法——利用反射机制推断RDD模式。

大家看，在我们的Spark安装目录下，有一个样例数据文件people.txt。这个文件位于"/usr/local/spark/examples/src/main/resources/"目录下。这是Spark安装时自带的测试数据。

我们来看看这个文件的内容，非常简单：

- Michael, 29
- Andy, 30
- Justin, 19

就是三个人的姓名和年龄，用逗号分隔。我们的任务是把这个文本文件加载到内存中，生成一个DataFrame（DataFrame），然后查询其中的数据。

这听起来很简单，但实际操作中有一些技巧，我们一步步来看。

슬라이드 3 6.5.1 利用反射机制推断RDD模式

好，现在我们来看具体的代码实现。大家注意看这段代码，我来逐行解释：

首先，我们需要导入Row类：

```
from pyspark.sql import Row
```

然后，我们通过一系列的map操作来处理数据：

```
people = spark.sparkContext.₩
textFile("file:///usr/local/spark/examples/src/main/resources/people.txt").₩
map(lambda line: line.split(",")).₩
map(lambda p: Row(name=p[0], age=int(p[1])))
```

这里有个重要的概念——我们使用了链式操作（chaining operations）。第一个map把每行按逗号分割，第二个map创建Row对象。

接下来，我们创建DataFrame：

```
schemaPeople = spark.createDataFrame(people)
```

大家注意这里！我们必须注册为临时表才能进行SQL查询：

```
schemaPeople.createOrReplaceTempView("people")
```

最后，我们就可以使用SQL语句查询了：

```
personsDF = spark.sql("select name,age from people where age > 20")
```

运行结果显示，Michael 29岁，Andy 30岁，都大于20岁，所以被选出来了。

슬라이드 4　6.5.1 利用反射机制推断RDD模式

这张图很重要，我们来仔细看看数据转换的过程。

最开始，我们有三个字符串：

- "Michael,29"
- "Andy,30"
- "Justin,19"

第一次map操作，使用line.split(",")，把字符串分割成列表：

- ["Michael","29"]
- ["Andy","30"]
- ["Justin","19"]

第二次map操作，创建Row对象：

- Row("Michael",29)
- Row("Andy",30)
- Row("Justin",19)

这个转换过程体现了函数式编程（functional programming）的思想，每一步都是数据的变换，非常清晰。大家要理解这个思路，这是Spark编程的核心思想。

```
from pyspark.sql import SparkSession, Row

if __name__ == '__main__':
    spark = SparkSession ₩
        .builder ₩
        .master("local[*]") ₩
        .appName("Reflection mechanism Application") ₩
        .getOrCreate()

    spark.sparkContext.setLogLevel('ERROR')



    people=spark.sparkContext.₩
```

```
        textFile("file:///usr/local/spark/examples/src/main/resources/people.txt").₩
        map(lambda line:line.split(",")).₩
        map(lambda p:Row(name=p[0], age=int(p[1])))

print('='*50)
print(people.collect())
print('=' * 50)

# Reflection inference로 DataFrame 생성(通过反射机制推断创建DataFrame)
# 내부적으로 데이터 샘플링과 타입 추론이 발생(内部进行数据采样和类型推断)
schemaPeople=spark.createDataFrame(people)
print('=' * 50)
print(schemaPeople.show())
print('=' * 50)

#必须注册为临时表， 才能供后面的查讯使用
schemaPeople.createOrReplaceTempView("people")

querySQL="select name, age from people where age>20"
personsDF=spark.sql(querySQL)

#DataFrame中的每个元素都是一行记录，包含name和 age 两个字段， 分别用 p.name 和 p.age 来获取值
personsRDD=personsDF.rdd.map(lambda p:"名字: "+p.name+","+"年龄: "+str(p.age))
personsRDD.foreach(print)

#资源清理
spark.stop()
```

슬라이드 5　6.5.2 使用编程方式定义RDD模式

现在我们来学习第二种方法——使用编程方式定义RDD模式。

什么时候需要用这种方法呢？当我们无法提前获知数据结构的时候。比如说，数据来源不确定，或者字段类型比较复杂的时候。

这种方法的思路是这样的：我们手动构建表的结构，就像搭积木一样。

第一步：制作"表头"（header），定义字段名称和类型 第二步：制作"表中的记录"（records），就是实际的数据行 第三步：把"表头"和"表中的记录"拼装在一起

你们看这个图，最终我们要得到一个完整的表格，有name和age两个字段，包含三条学生记录。

这种方法虽然代码稍微复杂一点，但是控制力更强，适合处理复杂的数据结构。

슬라이드 6　6.5.2 使用编程方式定义RDD模式

现在我们来看具体的代码实现。

首先，我们导入必要的类型定义：

```
from pyspark.sql.types import *
from pyspark.sql import Row
```

然后，我们来制作"表头"。注意看这几行代码：

```
schemaString = "name age"
fields = [StructField(field_name, StringType(), True) for field_name in schemaString.split(" ")]
schema = StructType(fields)
```

这里我们使用了StructField（StructField）和StructType（StructType）来定义模式。每个字段都是StringType（字符串类型），True表示允许空值。

接下来，我们处理数据，生成"表中的记录"：

```
lines = spark.sparkContext.textFile("file:///usr/local/spark/examples/src/main/resources/people.txt")
parts = lines.map(lambda x: x.split(","))
people = parts.map(lambda p: Row(p[0], p[1].strip()))
```

注意这里我们使用了strip()方法去除空格，这是数据清洗（data cleaning）的一个小技巧。

最后，我们把模式和数据组合起来：

```
schemaPeople = spark.createDataFrame(people, schema)
```

这种方法的好处是，我们可以精确控制每个字段的类型和属性。

슬라이드 7 6.5.2 使用编程方式定义RDD模式

这张图展示了第二种方法的数据转换过程，和第一种方法类似，但稍有不同。

我们从原始字符串开始：

- "Michael,29"
- "Andy,30"
- "Justin,19"

通过split操作得到列表：

- ["Michael","29"]
- ["Andy","30"]
- ["Justin","19"]

然后创建Row对象，但注意这里的写法稍有不同：

- Row("Michael","29")
- Row("Andy","30")
- Row("Justin","19")

第二种方法中，我们没有指定字段名，而是通过预先定义的schema来指定结构。这就是编程方式的特点——更加灵活，但需要更多的代码。

슬라이드 8

最后，我们来验证一下结果。

我们同样需要注册临时表：

```
schemaPeople.createOrReplaceTempView("people")
```

然后执行查询：

```
results = spark.sql("SELECT name,age FROM people")

results.show()
```

结果显示：

```
+-------+---+

|     name|age|

+-------+---+

|Michael|  29|

|     Andy|  30|

|  Justin|  19|

+-------+---+
```

大家看，两种方法得到的结果是一样的。第一种方法（反射机制）更简单，适合数据结构简单的情况；第二种方法（编程方式）更灵活，适合复杂的数据处理场景。

在实际工作中，我建议大家根据具体情况选择合适的方法。

```
from pyspark.sql import SparkSession, Row
from pyspark.sql.types import *
if __name__ == '__main__':
    spark = SparkSession ₩
        .builder ₩
        .master("local[*]") ₩
        .appName("Programming Method") ₩
        .getOrCreate()
```

```
# 下面生成"表头"
schemaString="name age"
fields=[StructField(field_name, StringType(), True) for field_name in schemaString.split(" ")]
schema=StructType(fields)

# 下面生成"表中的记录"
lines=spark.sparkContext.₩
      textFile("file:///usr/local/spark/examples/src/main/resources/people.txt")
parts=lines.map(lambda   x:x.split(","))
people=parts.map(lambda p:Row(p[0],p[1].strip()))

# 下面把"表头"和"表中的记录"拼装在一起
schemaPeople=spark.createDataFrame(people,schema)

# 注册一个临时表供下面查询使用
schemaPeople.createOrReplaceTempView("people")

results=spark.sql("SELECT name, age FROM people")
results.show()

spark.stop()
```