6.4 Spark DataFrame的基本操作

슬라이드 1: 강의 개요

今天我们要学习的是的内容—DataFrame（数据框）的基本操作。

大家看到屏幕上的内容，今天我们主要讲两个部分：

第一个是DSL语法风格（DSL Syntax Style），

第二个是SQL语法风格（SQL Syntax Style）。

这两种风格其实都是操作DataFrame（数据框）的方法，但是使用的语法不太一样。

DSL更像是函数式编程（Functional Programming），而SQL大家应该都比较熟悉，

就是我们平时写的那种查询语句。

好，我们先从DSL语法风格开始讲起。

6.4.1 슬라이드 2: DSL 语法风格介绍

DSL语法风格。

什么是DSL呢？DSL的全称是Domain Specific Language，中文叫"领域专用语言"。在Spark里面，这种DSL语法其实跟我们之前学过的RDD（弹性分布式数据集）操作很相似。

大家注意，DataFrame（数据框）创建好以后，我们可以调用很多方法来分析里面的数据。

比如说 printSchema()、show()、select()、filter()、groupBy()、sort()等等，这些都是常用的操作。

现在我们来创建一个DataFrame（数据框）。这里我用的是Spark自带的一个样例文件，

叫people.json。我们来看看这个代码：

```
filePath = "file:///usr/local/spark/examples/src/main/resources/people.json"
df = spark.read.json(filePath)
```

这样我们就创建了一个名为df的DataFrame（数据框）。

6.4.1 슬라이드 3: printSchema()和show()操作

好，现在我们来学习几个最基本的操作。

首先是printSchema()方法。大家看这个方法的名字就知道，它是用来打印Schema（模式）信息的。什么是Schema（模式）呢？就是告诉我们这个DataFrame（数据框）里面有哪些列，每一列是什么数据类型。

我们运行df.printSchema()，看到输出结果：

root

```
|-- age: long (nullable = true)
|-- name: string (nullable = true)
```

这告诉我们这个DataFrame（数据框）有两列：

一列叫age，类型是long（长整型），

另一列叫name，类型是string（字符串）。

nullable = true表示这一列可以为空值。

接下来是show()方法。这个方法是用来显示DataFrame（数据框）里面的具体数据的。

我们看一下输出：

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19|  Justin|
+----+-------+
```

大家可以看到，我们有三行数据：第一行是Michael，年龄为null（空值）；第二行是Andy，30岁；第三行是Justin，19岁。

6.4.1 슬라이드 4: select()操作

现在我们来学习select()操作。

select（选择）这个词大家应该很熟悉，在SQL（结构化查询语言）里面也有。

select()操作的作用是什么呢？

就是从DataFrame（数据框）中选取部分列的数据。

我们来看第一个例子：

```
df.select(df["name"], df["age"]+1).show()
```

这个操作做了什么呢？

它选择了name列和age列，但是把age列的值都加了1。

大家看输出结果，原来Andy是30岁，现在变成31了；Justin原来19岁，现在变成20了。

注意Michael那一行，因为原来age是null（空值），null加1还是null。

select()操作还有一个很有用的功能，就是可以给列重命名。

我们来看第二个例子：

```
df.select(df["name"].alias("username"), df["age"]).show()
```

这里我们用了alias()方法，把name列重命名为username。

大家看输出结果，列名确实变成username了。

6.4.1 슬라이드 5: filter()和where()操作

现在我们学习条件查询，也就是filter()和where()操作。

filter()操作是用来筛选数据的。

比如我们想找出所有年龄大于20岁的人：

```
df.filter(df["age"] > 20).show()
```

结果只返回了Andy这一行，因为只有他的年龄是30，大于20。

where()操作跟filter()操作功能是一样的，只是名字不同。

我们来看：

```
df.where(df["age"] > 20).show()
```

输出结果是一样的。

where()操作还有另一种写法，可以直接写字符串条件：

```
df.where("age > 20").show()
```

这种写法对熟悉SQL（结构化查询语言）的同学来说可能更亲切一些。

6.4.1 슬라이드 6: groupBy()和sort()操作

现在我们学习分组和排序操作。

groupBy()操作是用来分组的。比如我们想按照年龄分组，看看每个年龄有多少人：

```
df.groupBy("age").count().show()
```

结果显示：19岁有1个人，null（空值）有1个人，30岁有1个人。

sort()操作是用来排序的。我们来看两个例子：

第一个例子，按年龄降序排列：

```
df.sort(df["age"].desc()).show()
```

结果是30岁的Andy排在最前面，然后是19岁的Justin，最后是null（空值）的Michael。

第二个例子更复杂一点，先按年龄降序，如果年龄相同再按姓名升序：

```
df.sort(df["age"].desc(), df["name"].asc()).show()
```

当然，我们这个例子里面没有年龄相同的情况，所以结果看起来一样。

6.4.1 슬라이드 7: orderBy()操作

orderBy()操作跟sort()操作功能是一样的，都是用来排序的。

我们来看两种写法：

```
df.orderBy("age", ascending=False).show()
```

这里ascending=False表示降序排列。

我们也可以这样写：

```
df.orderBy(df["age"], ascending=False).show()
```

两种写法的结果是一样的。大家可以根据自己的习惯选择使用哪种。

6.4.1 슬라이드 8: withColumn()操作

现在我们学习一个很重要的操作，叫withColumn()。

withColumn()是用来给DataFrame（数据框）添加新列的。

大家要记住一个重要概念：DataFrame（数据框）本身是不可变的（immutable），也就是说我们不能直接修改原来的DataFrame（数据框）。但是我们可以创建一个新的DataFrame（数据框），在新的里面加上我们想要的列。

我们来看这个例子。首先需要导入一个函数：

```
from pyspark.sql.functions import expr
```

然后创建新的DataFrame（数据框）：

```
df2 = df.withColumn(
    "IfWithAge",
    expr("CASE WHEN age is null THEN 'NO' ELSE 'YES' END")
)
```

这里我们添加了一个新列叫"IfWithAge"。这个列的值是根据age列来决定的：如果age是null（空值），那么"IfWithAge"就是"NO"；否则就是"YES"。

expr()函数很重要，它可以把字符串解析成Spark（斯帕克）能理解的表达式。这里我们用的是CASE WHEN语句，这是SQL（结构化查询语言）里面的条件判断语法。

6.4.1 슬라이드 9: withColumnRenamed()和drop()操作

现在我们学习列的重命名和删除操作。

withColumnRenamed()操作是用来给列重命名的：

```
df.withColumnRenamed("name", "firstname").show()
```

这样就把name列重命名为firstname了。

drop()操作是用来删除列的。比如我们想删除刚才添加的"IfWithAge"列：

```
df3 = df2.drop("IfWithAge")

df3.show()
```

这样"IfWithAge"列就被删除了，DataFrame（数据框）又回到原来的样子。

6.4.1 슬라이드 10: 其他常用操作

DataFrame API（应用程序接口）还提供了很多统计方法，比如min()、max()、sum()、avg()等等。

我们来看一个例子：

```
import pyspark.sql.functions as F

df.select(F.sum("age"), F.avg("age"), F.min("age"), F.max("age")).show()
```

这里我们计算了age列的总和、平均值、最小值和最大值。注意Michael的年龄是null（空值），在计算的时候会被忽略。

除了这些基本的统计函数，Spark（斯帕克）还提供了很多高级的统计函数，比如stat()、describe()、correlation()（相关性）、covariance()（协方差）等等。感兴趣的同学可以查看官方文档。

```
from pyspark.sql import SparkSession

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

    filePath="file:///usr/local/spark/examples/src/main/resources/people.json"
    df=spark.read.json(filePath)

    print('='*50)
    print('DSL : printSchema() 结果 : ')
    df.printSchema()
    print('=' * 50)

    print('=' * 50)
    print('DSL : show() 结果 : ')
    df.show()
    print('=' * 50)

    print('=' * 50)
    print("select(df['name'],df['age']+1).show() 结果")
    df.select(df['name'],df['age']+1).show()
    print('=' * 50)

    print('=' * 50)
    print("select(df['name'].alias('username'),df['age']).show() 结果")
    df.select(df['name'].alias('username'),df['age']).show()
    print('=' * 50)

    print('=' * 50)
    print("filter(df['age']>20) 结果")
    df.filter(df['age']>20).show()
    print('=' * 50)

    print('=' * 50)
```

```
print("where(df['age']>20) 结果")
df.where(df['age']>20).show()
df.where('age>20').show()
print('=' * 50)

print('=' * 50)
print("groupBy('age').count() 结果")
df.groupBy('age').count().show()
print('=' * 50)

print('=' * 50)
print("sort(df['age'].desc()) 结果")
df.sort(df['age'].desc()).show()
df.sort(df['age'].desc(), df['name'].asc()).show()
print('=' * 50)

print('=' * 50)
print("orderBy('age',ascending=False) 结果")
df.orderBy('age',ascending=False).show()
df.orderBy(df['age'],ascending=False).show()
print('=' * 50)


from pyspark.sql.functions import expr
df2=df.withColumn(₩
      'IFWithAge',₩
      expr("CASE WHEN age is null THEN 'NO' ELSE 'YES' END")₩
      )
df2.show()


df.withColumnRenamed('name','firstname').show()
df3=df2.drop('IfWithAge')
df3.show()

import pyspark.sql.functions as F
df.select(F.sum('age'),F.avg('age'), F.min('age'),F.max('age')).show()
spark.stop()
```

6.4.2 슬라이드 11: SQL语法风格介绍

好，现在我们来学习第二种操作DataFrame（数据框）的方法——SQL语法风格（SQL Syntax Style）。

对于熟悉SQL（结构化查询语言）的同学来说，这种方法可能更亲切一些。我们可以直接写SQL（结构化查询语言）语句来操作DataFrame（数据框）。

但是有一个前提条件：在执行SQL（结构化查询语言）语句之前，我们需要给DataFrame（数据框）创建一个临时视图（Temporary View）。

创建临时视图有两个方法：createTempView()和createOrReplaceTempView()。

这两个方法有什么区别呢？

- createTempView()：如果当前会话中已经存在同名的临时视图，会直接报错。
- createOrReplaceTempView()：如果当前会话中已经存在同名的临时视图，会用新的视图替换原来的；如果不存在，就创建新的。

一般来说，我们更常用createTempView()，因为它更安全。

6.4.2 슬라이드 12: 创建DataFrame并显示数据

我们还是用刚才的数据，重新创建一个DataFrame（数据框）：

```
filePath = "file:///usr/local/spark/examples/src/main/resources/people.json"
df = spark.read.json(filePath)
df.show()
```

大家看到输出结果还是那三行数据：Michael（年龄null）、Andy（30岁）、Justin（19岁）。

```
df.createTempView("people")
```

这样我们就创建了一个名为"people"的临时视图。

现在我们可以用SQL（结构化查询语言）语句来查询了：

```
spark.sql("SELECT * FROM people").show()
```

这个查询选择了所有列和所有行，结果跟df.show()是一样的。

我们再来看一个带条件的查询：

```
spark.sql("SELECT name FROM people where age > 20").show()
```

这个查询选择了年龄大于20的人的姓名，结果只有Andy。

大家看到了吗？用SQL（结构化查询语言）语句来写查询，对于熟悉数据库的同学来说是不是更简单直观？

슬라이드 13: SQL函数介绍

现在我们来讲SQL函数（SQL Functions）。

在分析数据的时候，我们经常需要用到各种函数。Spark SQL（斯帕克SQL）提供了非常丰富的函数库，一共有200多个函数！

这些函数涵盖了各种应用场景：

- 转换函数（Conversion Functions）
- 数学函数（Math Functions）
- 字符串函数（String Functions）
- 二进制函数（Binary Functions）
- 日期时间函数（Date/Time Functions）
- 正则表达式函数（Regex Functions）
- JSON函数（JSON Functions）
- URL函数（URL Functions）
- 聚合函数（Aggregate Functions）
- 窗口函数（Window Functions）
- 集合函数（Collection Functions）

如果这200多个系统函数还不够用，我们甚至可以创建自己的用户定义函数（User Defined Functions，UDF）。

现在我们来看一个具体的例子。

슬라이드 14: 准备数据

我们来创建一个用户信息表，包含姓名（name）、年龄（age）和创建时间（create_time）三列：

```
from pyspark.sql import SparkSession, Row
from pyspark.sql.functions import from_unixtime
from pyspark.sql.types import IntegerType, StringType, LongType, StructField, StructType
```

首先定义Schema（模式）：

```
schema = StructType([
    StructField("name", StringType(), True),
    StructField("age", IntegerType(), True),
    StructField("create_time", LongType(), True)
])
```

然后准备数据：

```
data = []
data.append(Row("Xiaomei", 21, 1580432800))
data.append(Row("Xiaoming", 22, 1580436400))
data.append(Row("Xiaoxue", 23, 1580438800))
```

创建DataFrame（数据框）：

```
df = spark.createDataFrame(data, schema)
df.show()
```

大家看到，create_time列存储的是时间戳（timestamp），这些数字对人来说不太好理解。

슬라이드 15: 使用系统函数和用户定义函数

现在我们要做两件事：

1. 把时间戳转换成可读的时间格式
2. 把用户名转换成大写

首先创建临时视图：

```
df.createTempView("user_info")
```

使用系统函数from_unixtime()转换时间戳：

```
spark.sql("SELECT name,age,from_unixtime(create_time,'yyyy-MM-dd HH:mm:ss') FROM user_info ").show()
```

大家看到，时间戳被转换成了可读的日期时间格式。

现在我们创建一个用户定义函数（UDF）来把用户名转换成大写：

```
spark.udf.register("toUpperCaseUDF", lambda column: column.upper())
```

然后在SQL（结构化查询语言）中使用这个函数：

```
spark.sql("SELECT toUpperCaseUDF(name),age,from_unixtime(create_time,'yyyy-MM-dd HH:mm:ss') FROM user_info").show()
```

大家看到，用户名都变成大写了：XIAOMEI、XIAOMING、XIAOXUE。

```
# 146 코드를 수정함. 자료를 2개를 사용해서 비교가 가능하게 작성함. (Deppsekk 이용)
from pyspark.sql import SparkSession

if __name__ == '__main__':
    # 创建SparkSession
    spark = SparkSession ₩
        .builder ₩
        .master("local[*]") ₩
        .appName("SQL: createTempView vs createOrReplaceTempView") ₩
        .getOrCreate()

    # 创建示例数据
    data1 = [("Alice", 25), ("Bob", 30), ("Charlie", 35)]
    data2 = [("David", 28), ("Eve", 32), ("Frank", 40)]

    columns = ["name", "age"]
    df1 = spark.createDataFrame(data1, columns)
    df2 = spark.createDataFrame(data2, columns)

    print("=== DataFrame1 ===")
    df1.show()
    print("=== DataFrame2 ===")
    df2.show()

    print("₩n" + "=" * 50)
    print("1. createOrReplaceTempView() 使用示例")
    print("=" * 50)

    # createOrReplaceTempView - 可以使用相同名称重新创建（会替换）
    df1.createOrReplaceTempView("people_view")
    print("第一个 createOrReplaceTempView('people_view') 成功")
    spark.sql("SELECT * FROM people_view").show()

    # 使用相同名称重新创建（会替换现有视图）
    df2.createOrReplaceTempView("people_view")
    print("₩n第二个 createOrReplaceTempView('people_view') 成功（替换现有视图）")
    spark.sql("SELECT * FROM people_view").show()
    print("→ createOrReplaceTempView: 可以使用相同名称重新创建")
```

```
print("₩n" + "=" * 50)
print("2. createTempView() 使用示例")
print("=" * 50)

# createTempView - 尝试使用相同名称重新创建（会出错）
df1.createTempView("people_temp")
print("第一个 createTempView('people_temp') 成功")
spark.sql("SELECT * FROM people_temp").show()

# 尝试使用相同名称重新创建（会抛出TempTableAlreadyExistsException）
try:
    df2.createTempView("people_temp")
    print("第二个 createTempView('people_temp') 成功")
except Exception as e:
    print(f"₩ncreateTempView 错误: {type(e).__name__}")
    print(f"错误信息: {str(e)}")
    print("→ createTempView: 不能使用相同名称重新创建")

print("₩n" + "=" * 50)
print("3. 其他比较示例")
print("=" * 50)

# 检查当前会话中的临时视图
print("当前已注册的临时视图:")
for table in spark.catalog.listTables():
    if table.isTemporary:
        print(f"  - {table.name}")

# 使用不同名称创建视图
print("₩n使用不同名称创建视图测试:")
df1.createTempView("view1")
df1.createOrReplaceTempView("view2")
print("createTempView('view1') 成功")
print("createOrReplaceTempView('view2') 成功")

# 删除视图后重新创建
print("₩n视图删除测试:")
```

```
spark.catalog.dropTempView("people_temp")
print("dropTempView('people_temp') 成功")

# 删除后重新创建
df2.createTempView("people_temp")
print("删除后重新创建 createTempView('people_temp') 成功")
spark.sql("SELECT * FROM people_temp").show()

spark.stop()
```

总结

好，今天我们学习了DataFrame（数据框）的两种操作方式：

1. **DSL语法风格（DSL Syntax Style）**：使用方法链调用，比如df.select().filter().show()
2. **SQL语法风格（SQL Syntax Style）**：创建临时视图后使用SQL（结构化查询语言）语句

两种方式各有优点：

- DSL语法更加面向对象，代码补全更好
- SQL语法对数据库背景的同学更友好，表达更直观

在实际工作中，大家可以根据具体情况和个人偏好来选择使用哪种方式。很多时候，我们甚至会在同一个项目中混合使用这两种方式。

今天的内容就到这里，大家有什么问题吗？