6.4 DataFrame的基本操作

6.4.1 DSL语法风格

6.4.2 SQL语法风格

6.4.1 DSL语法风格



- DSL（Domain Specific Language）意为“领域专用语言”，DSL语法类似于RDD中的操作，允许开发者通过调用方法对DataFrame内部的数据进行分析。
- DataFrame创建好以后，可以执行一些常用的DataFrame操作，包括printSchema()、show()、select()、filter()、groupBy()、sort()、withColumn()和drop()等。这里首先从Spark自带的样例文件people.json创建一个名称为df的DataFrame：

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

6.4.1 DSL语法风格



- printSchema()

可以使用printSchema()操作，打印出DataFrame的模式（Schema）信息（见图6-6）。

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

df.printSchema()
df.show()
```

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

- show()

show()操作用于显示一个DataFrame的具体内容（如图6-7所示）。

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

6.4.1 DSL语法风格



- select()

select()操作的功能，是从DataFrame中选取部分列的数据。如图6-8所示，select()操作选取了name和age这两个列，并且把age这个列的值增加1。

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

```
+-------+---------+
|   name|(age + 1)|
+-------+---------+
|Michael|     null|
|   Andy|       31|
| Justin|       20|
+-------+---------+
```

select()操作还可以实现对列名称进行重命名的操作。如图6-9所示，name列名称被重命名为username。

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

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

6.4.1 DSL语法风格



- filter()

  filter()操作可以实现条件查询，找到满足条件要求的记录。如图6-10所示，df.filter(df["age"]>20)用于查询所有age字段大于20的记录。

- where()

  where()操作可以实现条件查询，找到满足条件要求的记录。如图6-11所示，df.where(df["age"]>20)用于查询所有age字段大于20的记录。或者，也可以写成“df.where("age>20").show()”。

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

```
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
```

```
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
```

6.4.1 DSL语法风格



- groupBy()

groupBy()操作用于对记录进行分组。如图6-12所示，可以根据age字段进行分组，并对每个分组中包含的记录数量进行统计。

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

```
+----+-----+
| age|count|
+----+-----+
|  19|    1|
|null|    1|
|  30|    1|
+----+-----+
```

- sort()

sort()操作用于对记录进行排序。如图6-13所示，df.sort(df["age"].desc)表示根据age字段进行降序排序。df.sort(df["age"].desc,df["name"].asc)表示根据age字段进行降序排序，当age字段的值相同时，再根据name字段进行升序排序。

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

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

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

6.4.1 DSL语法风格



- orderBy()

orderBy()操作用于对记录进行排序。
如图6-14所示，df.orderBy("age",ascending=False)表示根据age字段进行降序排序。
或者，也可以写成df.orderBy(df["age"],ascending=False)。

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

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


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

6.4.1 DSL语法风格



- withColumn()

withColumn()操作用于为一个DataFrame增加一个新的列。虽然一个DataFrame本身是不可变的，但是我们依然可以通过创建另一个DataFrame来实现增加一个列。如图6-15所示，这里新建了一个名称为“IfWithAge”的列，该列的取值取决于“age”这个列的值，如果“age”这个列有值，则“IfWithAge”的值为“YES”，如果“age”这个列的值是null，则“IfWithAge”的值为“NO”。expr()是org.apache.spark.sql.functions包提供的一个函数接口，它所接受的参数会被Spark解析为表达式，用于计算结果。

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

```
+----+-------+---------+
| age|   name|IFWithAge|
+----+-------+---------+
|null|Michael|       NO|
|  30|   Andy|      YES|
|  19| Justin|      YES|
+----+-------+---------+
```

6.4.1 DSL语法风格



- withColumnRenamed()

withColumnRenamed()操作用于为一个DataFrame的列起一个新的名字，如图6-16所示。

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

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

- drop()

要想删除DataFrame中的一个列，可以使用drop()方法。如图6-17所示，这里使用drop()方法把“IfWithAge”这个列删除了。

```
df3=df2.drop('IfWithAge')
df3.show()
```

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

6.4.1 DSL语法风格



- 其他常用操作

DataFrame API也提供了描述型的统计方法，比如min()、max()、sum()和avg()等，图6-18是一个具体实例。针对数据科学应用中其他常用的高级统计需求，可以参考官方文档，其中有stat()、describe()、correlation()、covariance()、sampleBy()、approxQuantile()、frequentItems()等函数的介绍。

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

```
+--------+--------+--------+--------+
|sum(age)|avg(age)|min(age)|max(age)|
+--------+--------+--------+--------+
|      49|    24.5|      19|      30|
+--------+--------+--------+--------+
```

6.4.2 SQL语法风格



1.使用SQL语句操作DataFrame

- 熟练使用SQL语法的开发者，可以直接使用SQL语句进行数据操作。相比于DSL语法风格，在执行SQL语句之前，需要通过DataFrame实例创建临时视图(temporary view)。
- 创建临时视图的方法是调用DataFrame实例的createTempView或createOrReplaceTempView方法，二者的区别是，后者会进行判断。
- 对于createOrReplaceTempView方法而言，如果在当前会话中存在相同名称的临时视图，则用新视图替换原来的临时视图，如果在当前会话中不存在相同名称的临时视图，则创建临时视图。
- 对于createTempView方法而言，如果在当前会话中存在相同名称的临时视图，则会直接报错。

6.4.2 SQL语法风格



```
# coding:utf8
from pyspark.sql import SparkSession

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

    filePath = "file:///usr/local/spark/examples/src/main/resources/people.json"
    df = spark.read.json(filePath)
    df.createTempView("people")
    spark.sql("SELECT * FROM people").show()
    spark.sql("SELECT name FROM people where age>20").show()
    spark.stop()
```

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


+----+
|name|
+----+
|Andy|
+----+
```

6.4.2 SQL语法风格



2.SQL函数

- 在分析数据的过程中编写的SQL语句有时会涉及SQL函数的调用。
- Spark SQL提供了丰富的函数供用户选择，一共200多个，基本涵盖了大部分的日常应用场景，
  - 包括转换函数、数学函数、字符串函数、二进制函数、日期时间函数、
  - 正则表达式函数、JSON函数、URL函数、聚合函数、
  - 窗口函数和集合函数等。
- 当Spark自带的这些系统函数无法满足用户需求时，用户还可以创建“用户自定义函数”。关于Spark自带的200多个系统函数，这里不做介绍，感兴趣的学生可以参考Spark官网资料。

- 下面给出一个实例来介绍SQL函数的用法。

6.4.2 SQL语法风格



假设在一张用户信息表中有name、age、create_time三列数据，这里要求使用Spark的系统函数from_unixtime，将时间戳类型的create_time格式化成时间字符串，然后，使用用户自定义函数将用户名转化为大写英文字母。具体实现代码如下：

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

    schema = StructType([StructField("name", StringType(), True), \
                         StructField("age", IntegerType(), True), \
                         StructField("create_time", LongType(), True)])
    data = []
    data.append(Row("Xiaomei", 21, 1580432800))
    data.append(Row("Xiaoming", 22, 1580436400))
    data.append(Row("Xiaoxue", 23, 1580438800))
    df=spark.createDataFrame(data,schema)
    df.show()
```

```
+--------+---+-----------+
|    name|age|create_time|
+--------+---+-----------+
| Xiaomei| 21| 1580432800|
|Xiaoming| 22| 1580436400|
| Xiaoxue| 23| 1580438800|
+--------+---+-----------+
```

6.4.2 SQL语法风格



```
data = []
data.append(Row("Xiaomei", 21, 1580432800))
data.append(Row("Xiaoming", 22, 1580436400))
data.append(Row("Xiaoxue", 23, 1580438800))
df=spark.createDataFrame(data,schema)
df.createTempView("user_info")
spark.sql("SELECT name,age,from_unixtime(create_time,'yyyy-MM-dd HH:mm:ss') \
    FROM user_info").show()
spark.udf.register("toUpperCaseUDF",lambda column:column.upper())
spark.sql("SELECT toUpperCaseUDF(name),age,from_unixtime(create_time,'yyyy-MM-dd HH:mm:ss')\
    FROM user_info").show()
spark.stop()
```

```
+--------+---+-------------------------------------------------+
|    name|age|from_unixtime(create_time, yyyy-MM-dd HH:mm:ss)|
+--------+---+-------------------------------------------------+
| Xiaomei| 21|                              2020-01-31 10:06:40|
|Xiaoming| 22|                              2020-01-31 11:06:40|
| Xiaoxue| 23|                              2020-01-31 11:46:40|
+--------+---+-------------------------------------------------+


+--------------------+---+-------------------------------------------------+
|toUpperCaseUDF(name)|age|from_unixtime(create_time, yyyy-MM-dd HH:mm:ss)|
+--------------------+---+-------------------------------------------------+
|             XIAOMEI| 21|                              2020-01-31 10:06:40|
|            XIAOMING| 22|                              2020-01-31 11:06:40|
|             XIAOXUE| 23|                              2020-01-31 11:46:40|
+--------------------+---+-------------------------------------------------+
```



```
from pyspark.sql import SparkSession
from pyspark.sql.functions import avg
if __name__ == '__main__':
    spark = SparkSession\
        .builder\
        .master("local[*]")\
        .appName("Simple Application")\
        .getOrCreate()

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

```
/home/hadoop/anaconda3/envs/nanyang/bin/python /home/hadoop/PycharmProjects/ch06SparkSQL/p142_Dat
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)

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