

DataFrames

Лекция 3

Узнаем про базовую структуру данных Spark



Алексей Ёлгин

Ведущий Дата Инженер, Альфа-Банк

- Управляю инфраструктурой кластера и всеми ETL процессами расчёта витрин
- Отслеживаю и оптимизирую работу запускаемых spark приложений на кластере
- Внедряю spark ETL-процессы, формирующие логику обновления данных в витринах



Проекты:

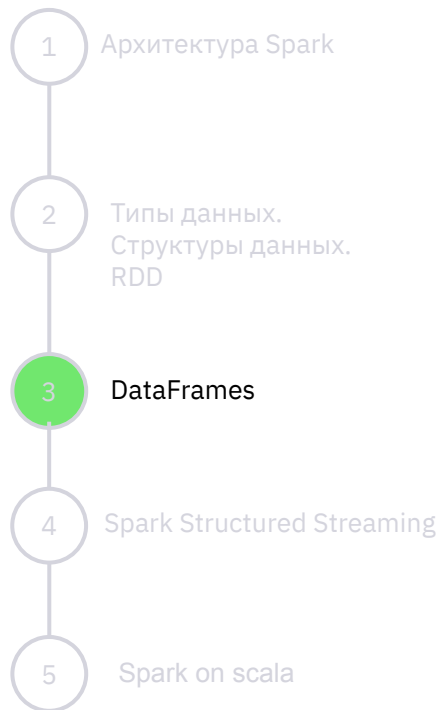
- Разработал гибкую систему оповещений по отслеживанию запускаемых ETL потоков кластера
- Модифицировал логику формирования автоматической отчётности по контролю качества поступающих данных
- Выстраивал ETL процессы для ML моделей отдела



Опыт работы: СберБанк, Альфа-Банк










План курса (вертикальный)





Что будет на уроке сегодня

-  узнаем о DataFrames
-  Кратко обсудим форматы данных
-  поговорим о физическом плане чуть подробнее
-  узнаем про Catalyst
-  рассмотрим создание DataFrame
-  глянем на функции DataFrames
-  узнаем как сохранять данные



DataFrame

DataFrame- это структура данных, представленная библиотекой Spark SQL. Она представляет собой таблицу данных с именованными столбцами. В отличие от RDD, датафреймы позволяют осуществлять более удобную и эффективную работу с данными, благодаря более высокому уровню абстракции.



Форматы данных

- это способ организации структуры файлов данных в компьютере



**строковый формат данных
(линейный)**

AVRO, Sequence



**колоночный формат данных
(столбчатый)**

Orc, Parquet



Пример

id	name	age
1	Иван	30
2	Лена	25
3	Петр	50

В линейных форматах данные будут храниться следующим образом

1	Иван	30	2	Лена	25	3	Петр	50
---	------	----	---	------	----	---	------	----

В колоночных форматах

1	2	3	Иван	Лена	Петр	30	25	50
---	---	---	------	------	------	----	----	----



Catalyst

Catalyst - это оптимизатор выполнения запросов, используемый в Apache Spark. Он представляет собой модуль, который анализирует запросы, представленные в виде логического плана, и оптимизирует их для выполнения на кластере Spark. Catalyst используется для оптимизации выполнения запросов над датафреймами и наборами данных Spark.



Catalyst

Catalyst работает в несколько этапов.

1. Catalyst составляет неразрешенный логический план. На этом этапе Spark проверяет “правильность” Spark кода
2. Затем Catalyst составляет разрешенный логический план. На этом этапе Spark проверяет правильность типов столбцов
3. Далее Catalyst составляет оптимизированный логический план. На этом этапе Spark выполняет различные оптимизации. Например, если вы в своем коде сначала делаете группировку, а затем фильтрацию, то Catalyst в оптимизированном плане запроса “передвинет” фильтрацию вперед группировки, чтобы облегчить расчеты, но при условии, что это возможно.
4. Затем уже составляются несколько физических планов из которых Catalyst выберет наиболее оптимизированный



Вопрос

расставьте в правильном порядке этапы составления
физического плана запроса

- физический план
- оптимизированный физический план
- разрешенный логический план
- неразрешенный логический план





Ответ

- 1) неразрешенный логический план
- 2) разрешенный логический план
- 3) оптимизированный физический план
- 4) физический план



Spark SQL

Spark SQL - это модуль Apache Spark, который позволяет работать с данными в табличной форме, используя SQL-запросы. Spark SQL использует датафреймы в качестве основной структуры данных, что позволяет проводить анализ данных с использованием знакомых SQL-запросов. Spark SQL также поддерживает работу с данными из различных источников, включая файлы, базы данных, Hive и другие.



Как создать DataFrame

Есть несколько способов для создания DataFrame.

- из python list
- прочитать из файла
- прочитать таблицу из Hive
- прочитать таблицу по jdbc
- range

рассмотрим кратко каждый из способов:



Из python list:

Здесь мы инициализируем spark сессию

составляем data - это лист из кортежей из элементов таблицы, так как у нас два столбца, то и элементов в кортежах два

составляем список с наименованиями столбцов - columns

создаем датафрейм с помощью метода spark createDataFrame, в качестве параметров передаем data и columns

```
3 spark = (SparkSession.builder.master("local[*]").getOrCreate())
4
5 data = [("Ivan", 10),
6         ("Petr", 20),
7         ("Elena", 30)]
8
9
10 columns = ['name', 'age']
11
12 df = spark.createDataFrame(data=data, schema=columns)
```



Из файла:

допустим у нас есть файл data.csv

Здесь мы активируем spark сессию

Затем используем метод `read`, чтобы прочесть данные, также указываем `option` - это метод для добавления различных опций в формате ключ — значение. Мы указали ключ `"header"` и значение `true`, что означает что первая строка csv файла является наименованием столбцов. И затем используем метод `csv` для прочтения csv файла, в качестве параметра передаем путь до нашего файла.

1	name, age
2	Ivan, 30
3	Petr, 50
4	Elena, 25

```
1 from pyspark.sql import SparkSession
2
3 spark = (SparkSession.builder.master("local[*]").getOrCreate())
4
5 df = spark.read.option(key="header", value="true").csv("data.csv")
```



Чтобы прочитать данные из таблицы hive:

В крупных компаниях часто используют СУБД Apache Hive, коннект с которой доступен в spark сразу.

Инициализируем spark сессию, но с настройкой `enableHiveSupport()`. Эта настройка позволяет нам считывать и записывать таблицы, используя только название таблицы, т.е вам не нужно прописывать путь хранения данных.

В 5 строке мы используем метод `table()`, параметр которого это название таблицы. Этот метод вернет нам `DataFrame` с данными из таблицы Hive

```
1 from pyspark.sql import SparkSession
2
3 spark = (SparkSession.builder.enableHiveSupport().getOrCreate())
4
5 df = spark.table("table_name")
```




Прочитать таблицу по JDBC:

JDBC (Java Database Connector) - специальная библиотека, написанная на java, позволяющая подключаться к базе данных (Postgres, Oracle, GreenPlum и тд) и читать или записывать туда данные.

Здесь мы инициализируем спарк сессию, в переменной creds у нас словарь, в котором указаны логин и пароль для подключения к базе данных

Затем мы читаем данные с помощью метода jdbc, в нем указываем url базы данных, нужную нам таблицу (которую мы хотим прочитать), properties - это наш словарь creds, в котором указаны наши логин и пароль.

```
1 from pyspark.sql import SparkSession
2
3 spark = (SparkSession.builder.master("local[*]").getOrCreate())
4
5 creds = {"user": "name", "password": "12345"}
6
7 df = spark.read.jdbc(url="url", table="table", properties=creds)
```



Range

Самый простой способ создать датафрейм, в котором будут числа, а столбец будет называться id

инициализируем спарк сессию, затем используем метод range и указываем количество чисел в датафрейме

```
1 from pyspark.sql import SparkSession
2
3 spark = (SparkSession.builder.master("local[*]").getOrCreate())
4
5 df = spark.range(100)
```



Функции

DataFrames в Apache Spark предоставляют множество функций для работы с данными. Некоторые из них:

- **select**: выбор столбцов из датафрейма
- **filter**: фильтрация строк датафрейма на основе условия
- **withColumn**: создание столбца в датафрейме
- **withColumnRenamed**: переименование столбца в датафрейме
- **groupBy**: группировка строк датафрейма по значениям в столбцах
- **agg**: агрегация данных в датафрейме с использованием агрегатных функций, таких как сумма, среднее и т.д.
- **join**: объединение двух датафреймов на основе заданных условий
- **orderBy**: сортировка строк датафрейма по значениям в столбцах

Кроме того, в DataFrames есть множество других функций, таких как **distinct**, **dropDuplicates**, **count**, **mean**, **sum**, **max**, **min** и т.д., которые позволяют проводить анализ данных и получать информацию о данных.



Функции

Перед тем, как мы разберем эти функции, рассмотрим специальную функцию в Spark - `col("column_name")`. Эта функция "указывает" Spark, что мы обращаемся к колонке датафрейма с названием `column_name`.

`select`

```
df.select(col("id")) # вернет DataFrame с одной колонкой id
```

`filter`

```
df.filter(col("id") < 50) # вернет DataFrame с числами меньше 50
```

`withColumn`

```
df.withColumn(colName: "new", col("id") % 2)
# первый параметр - имя нового столбца
# второй параметр - как создать новый столбец (мы берем остаток от деления на 2)
# , т.е в столбце new будут значения 0 и 1
```



Функции

withColumnRenamed

```
df.withColumnRenamed( existing: "id", new: "id_rename")  
# переименовываем столбец id в id_rename
```

groupby

```
df.withColumn( colName: "new", col("id") % 2).groupBy(col("new"))  
# создадим новый датафрейм, и сделаем группировку по нему  
# но дальше нам нужно сделать агрегацию!
```

agg

```
(df.withColumn( colName: "new", col("id") % 2)  
  .groupBy(col("new"))  
  .agg(max(col("id"))))  
# здесь мы делаем агрегацию - находим максимальное число в id при группировке по new
```



Функции

Join

```
df2 = spark.range(10).withColumnRenamed(existing: "id", new: "id2")
df.join(df2, col("id") == col("id2"), how: "inner")
# делаем джойн df и df2 по столбцам id и id2
# тип джойна - inner, но можно использовать также и left, right, full, leftanti, leftsemi
```

orderBy

```
df.orderBy(col("id").desc)
df.orderBy(col("id").asc)
# метод orderBy позволяет сортировать датафрейм по столбцу
# asc - по возрастанию
# desc - по убыванию
```

distinct

```
df.distinct()
# делаем distinct, т.е убираем дубли из датафрейма
```



Группировки и агрегации

name	age
Лена	20
Петр	30
Вася	15
Петр	25
Лена	50

после группировки

Лена	20, 50
Петр	30, 25
Вася	15

после агрегации

name	mean_age
Лена	35
Петр	27.5
Вася	15



Группировки и агрегации

исходные данные

1	name, age
2	Lena, 25
3	Ivan, 50
4	Petr, 25
5	Lena, 80
6	Katya, 10
7	Petr, 1
8	Ivan, 50
9	Katya, 35
10	Lena, 45
11	Ivan, 60

```
spark = SparkSession.builder.master("local[*]").getOrCreate()

df = spark.read.option(key: "header", value: "true").csv("data.csv")
```

```
df.groupby(col("name")) \
    .agg(mean(col("age")).alias("count_age")) \
    .show()
```

```
+-----+-----+
| name|      count_age|
+-----+-----+
| Lena|          50.0|
| Petr|          13.0|
| Ivan|53.33333333333336|
| Katya|          22.5|
```




Группировки и агрегации

исходные данные

1	name,age
2	Lena,25
3	Ivan,50
4	Petr,25
5	Lena,80
6	Katya,10
7	Petr,1
8	Ivan,50
9	Katya,35
10	Lena,45
11	Ivan,60

```
spark = SparkSession.builder.master("local[*]").getOrCreate()

df = spark.read.option(key: "header", value: "true").csv("data.csv")
```

```
df.groupby(col("name")) \
    .agg(collect_list(col("age")).alias("count_age")) \
    .show()
```

```
+-----+-----+
| name|    count_age|
+-----+-----+
| Lena|[25, 80, 45]|
| Petr|    [25, 1]|
| Ivan|[50, 50, 60]|
| Katya|   [10, 35]|
+-----+-----+
```



Оконные функции

Оконная функция — это функция, которая выполняет агрегирующую функцию на определенном наборе (окне, партии) строк и результат записывает в новый столбец в таблице.

name	age
Иван	15
Иван	30
Лена	20
Лена	40
Лена	60



Оконные функции

Пример агрегирующей функции max

name	age	max(age)
Иван	15	30
Иван	30	30
Лена	20	60
Лена	40	60
Лена	60	60



Оконные функции

Пример ранжирующей функции row_number

name	age	row_number(age)
Иван	15	2
Иван	30	1
Лена	20	3
Лена	40	2
Лена	60	1



Оконные функции

Row_number

Col_Value	RowID
A	1
A	2
A	3
B	4
B	5
C	6
C	7

rank

Col_Value	RowID
A	1
A	1
A	1
B	4
B	4
C	6
C	6

dense_rank

Col_Value	RowID
A	1
A	1
A	1
B	2
B	2
C	3
C	3



Оконные функции

Пример функций смещения lag, lead

name	quartal	subject	grade	previous_grade	next_grade
Петя	1 четверть	физика	4	[NULL]	3
Петя	2 четверть	физика	3	4	4
Петя	3 четверть	физика	4	3	5
Петя	4 четверть	физика	5	4	[NULL]



Оконные функции

Пример кода

```
from pyspark.sql.window import Window
```

```
from pyspark.sql.functions import rank
df.withColumn( colName: "rank_col", rank() \
               .over(Window.partitionBy("partition_col") \
                     .orderBy("orderby_col"))) )
```



When n otherwise

```
from pyspark.sql.functions import when, col

spark = SparkSession.builder.getOrCreate()

df = spark.range(100)

df.withColumn(colName="test_col", when(col("id") > 50, value="value_more_50").otherwise("value_less_50"))
```




Pivot

Pivot - позволяет развернуть сгруппированные значения строк в столбцы

name	city	salary
Ivan	Moscow	150000
Ivan	Moscow	20000
Lena	Moscow	100000
Petr	Sochi	50000
Roma	Sochi	60000
Poma	Sochi	10000



Pivot

name	sity	mean(salary)
Ivan	Moscow	85000
Lena	Moscow	100000
Petr	Sochi	50000
Roma	Sochi	35000

city	Ivan	Lena	Petr	Roma
Moscow	85000	100000	null	null
Sochi	null	null	60000	35000



Pivot

Посмотрим на код

так выглядит наш датафрейм

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, mean

spark = SparkSession.builder.master("local[*]").getOrCreate()

data = [("Ivan", "Moscow", 150000),
        ("Lena", "Moscow", 100000),
        ("Ivan", "Moscow", 20000),
        ("Petr", "Sochi", 60000),
        ("Roma", "Sochi", 50000),
        ("Roma", "Sochi", 10000)]

df = spark.createDataFrame(data=data).toDF(*cols: "name", "city", "salary")
```

```
+---+-----+-----+
|name|  city|salary|
+---+-----+-----+
|Ivan|Moscow|150000|
|Lena|Moscow|100000|
|Ivan|Moscow| 20000|
|Petr| Sochi| 60000|
|Roma| Sochi| 50000|
|Roma| Sochi| 10000|
+---+-----+-----+
```



Pivot

для примера сделаем группировку

```
df.groupby(col("city"), col("name")).agg(mean(col("salary")))
```

```
+-----+-----+-----+
|  city|name|avg(salary)|
+-----+-----+-----+
|Moscow|Ivan|      85000.0|
|Moscow|Lena|     100000.0|
| Sochi|Petr|      60000.0|
| Sochi|Roma|      30000.0|
+-----+-----+-----+
```



Pivot

Теперь сделаем pivot

```
df.groupby("city").pivot("name").agg(mean('salary'))
```

```
+-----+-----+-----+-----+-----+
|  city|   Ivan|    Lena|   Petr|   Roma|
+-----+-----+-----+-----+-----+
|Moscow|85000.0|100000.0|   null|   null|
| Sochi|   null|    null|60000.0|30000.0|
+-----+-----+-----+-----+-----+
```



Cast

Функция Cast - меняет тип данных

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, mean

spark = SparkSession.builder.master("local[*]").getOrCreate()

df = spark.range(100)

df.printSchema()
```

```
root
 |-- id: long (nullable = false)
```



Cast

```
new_df = df.select(col("id").cast("string"))

new_df.printSchema()

# ИЛИ
from pyspark.sql.types import StringType
new_df2 = df.select(col("id").cast(StringType))
```

```
root
 |-- id: string (nullable = false)
```



Cast

Другой способ

```
new_df = df.withColumn(colName: "id", col("id").cast("string"))  
new_df.printSchema()
```

```
root  
|-- id: string (nullable = false)
```




UDF

UDF - user defined function - это функции, определяемые пользователем.

Эти функции используются, когда вы реализовать более сложную логику.

```
spark = SparkSession.builder.master("local[*]").getOrCreate()

data = [("Iphone", "Iphone14", 150000),
        ("Huawei", "Mate14", 100000),
        ("Samsung", "A14", 20000),
        ("Iphone", "Iphone13", 70000),
        ("Huawei", "Mate12", 50000),
        ("Samsung", "A11", 5000),
        ]

df = spark.createDataFrame(data=data).toDF(*cols: "company", "model", "price")
```

```
+-----+-----+-----+
|company|  model| price|
+-----+-----+-----+
| Iphone|Iphone14|150000|
| Huawei|  Mate14|100000|
| Samsung|    A14| 20000|
| Iphone|Iphone13| 70000|
| Huawei|  Mate12| 50000|
| Samsung|    A11|  5000|
+-----+-----+-----+
```



UDF

```
def recognize_model(model: str) -> str:
    if "14" in model:
        return "new model"
    else:
        return "old model"

recognize_model_udf = udf(lambda x: recognize_model(x))

new_df = df.withColumn(colName="version_model", recognize_model_udf("model"))
```

company	model	price	version_model
Iphone	Iphone14	150000	new model
Huawei	Mate14	100000	new model
Samsung	A14	20000	new model
Iphone	Iphone13	70000	old model
Huawei	Mate12	50000	old model
Samsung	A11	5000	old model



Сохранение данных

Для сохранения данных из DataFrames в Apache Spark можно использовать метод `write` объекта `DataFrame`, который позволяет записывать данные в различные источники, такие как файлы, базы данных, S3 и другие.

Например, чтобы сохранить данные в CSV файл, можно использовать следующий код:

```
df.write.csv("path/to/output/folder", header=True)
```

В данном примере мы записываем данные в CSV файл, указывая путь к выходной папке и опцию `header=True`, которая указывает на запись заголовка в файл.



Подведем итоги лекции



Узнали о такой структуре данных как DataFrame



Обсудили форматы данных и какой формат для каких задач использовать



Разобрали Catalyst



Посмотрели, как создать DataFrame руками или прочитав данные



Рассмотрели основные операции в Spark



Узнали как сохранять данные



Спасибо за внимание

