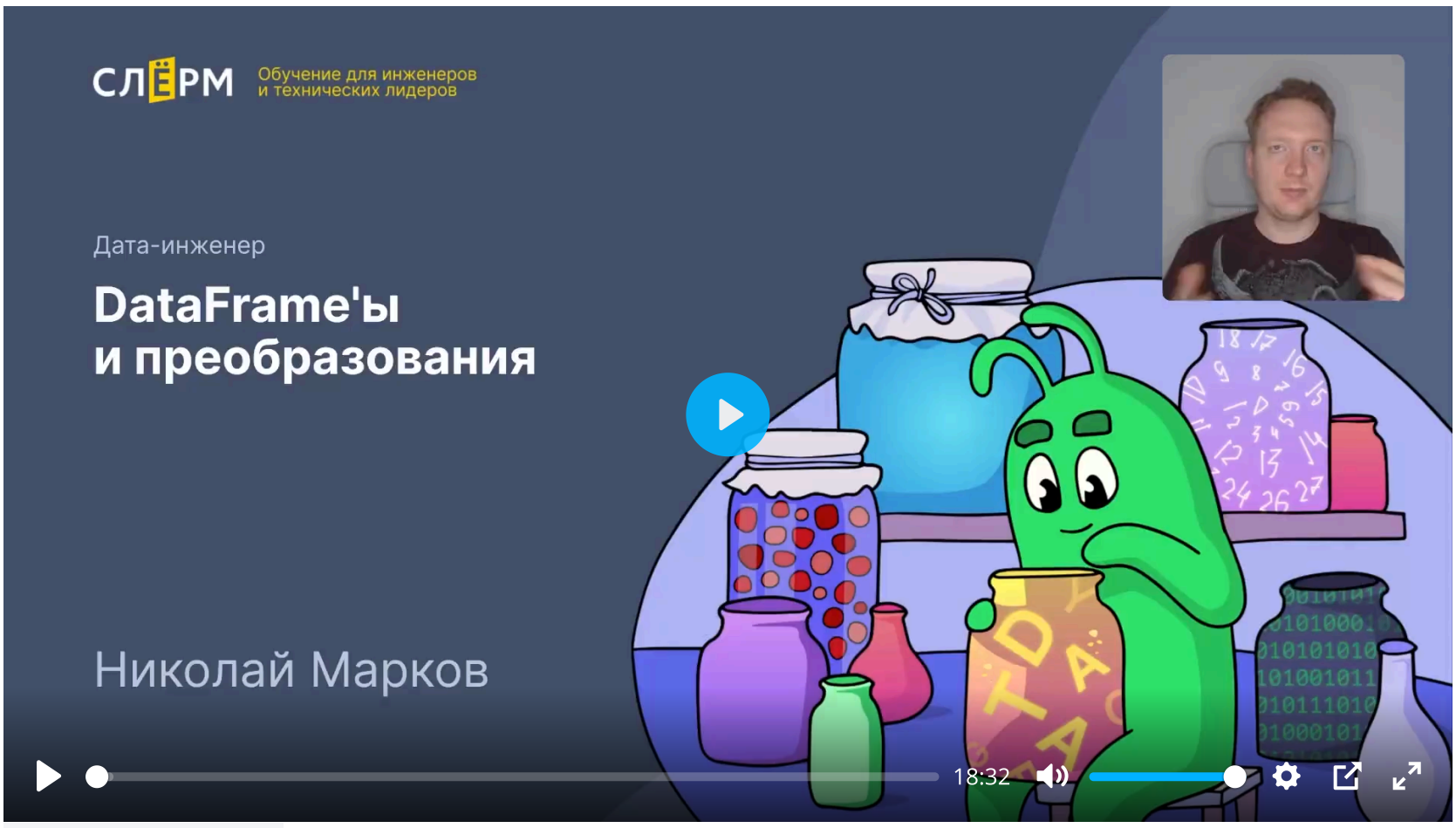
4.3 Dataframe'ы и преобразования



Презентация к уроку 4.3

Текстовая расшифровка видео:

DATAFRAME'Ы И ПРЕОБРАЗОВАНИЯ

План:

- Базовая разница;
- Глазами Scala/Java;
- Глазами Python;
- Конвертация между JVM и нативным Python;
- Создание сессий.

Базовая разница

RDD – это распределенная коллекция данных любого типа.

Однако, получая такую гибкость, мы теряем возможные оптимизации. Например, четко зная, что данные в нашей структуре представляют собой определенный тип, мы можем более эффективно оптимизировать разные запросы и разные механизмы работы с ними, получив от этого профит.

DataFrame – это таблица со своей схемой, состоящая из колонок конкретных типов (**org.apache.spark.sql.types**).

С DataFrame мы встречались в Python, когда обсуждали Pandas. **Pandas** – это библиотека в Python для работы с DataFrame.

Идея проста: у нас есть колонки и ряды. В контексте DataFrame типы колонок не ограничены конкретным набором. Мы можем использовать только те типы, которые поднимают подлежащий под фреймворком движок (как в Pandas, так и в Spark). В языках, на которых написан Spark (Scala и Java) используются в первую очередь обертки Scala и Java.

- RDD использует Java-сериализацию, что позволяет работать с любыми типами.
- DataFrame использует механизмы RDD, но лишь отчасти. DataFrame более структурирован.
- Для понимания основной «магии» существуют три ключевых проекта:

1. Project Tungsten – проект, позволяющий максимально эффективно использовать железо без дополнительных оверхедов.
2. Whole-Stage Java Code Generation – проект, созданный для ускорения работы. Идея проекта: исходя их знаний о том, какое у нас железо, нода и ограничения, Spark использует кодогенерацию на Java и создает целые классы в рамках исполнения графа задач для ускорения.
3. Catalyst Optimizer – проект, разработанный в мире Spark. Основная идея: когда мы работаем с DataFrame, чаще всего работа с ним происходит в виде SQL. Данный оптимизатор – cost based-оптимизатор, который в процессе выполнения графа задач рассчитывает разные планы выполнения и выбирает оптимальный на основе метода ветвей и границ.

- RDD позволяет использовать почти любые функции как в составе трансформаций, так и в составе действий.
- DataFrame позволяет ограниченный набор операций: SQL-выражения, org.apache.spark.sql.types, пользовательские функции (User-Defined Functions, UDF).
- В RDD данные из источника необходимо достать вручную, в DF (DataFrame) часто есть обобщенные коннекторы.
- RDD DStreams устарел, DF Structured Streaming – наше все.

Глазами Python

Как в случае с RDD, у нас есть возможность создать DataFrame из структуры, а также бесшовно сконвертировать Pandas DataFrame, Spark DataFrame и наоборот.

Рассмотрим пример:

```
1. import numpy as np
2. import pandas as pd
3.
4. # Включить конвертацию типов через PyArrow
5. spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")
6.
7. pandas_df = pd.DataFrame(np.random.rand(100, 3))
8.
9. # Создание Spark DataFrame из Pandas DataFrame
10. df = spark.createDataFrame(pandas_df)
11.
12. # Сконвертировать Spark DataFrame обратно в Pandas DataFrame
13. result_pdf = df.select("*").toPandas()
14.
15. print("Pandas DataFrame result statistics:\n%s\n" % str(result_pdf.describe()))
```

Это пример кода, где мы создаем DataFrame. С помощью метода «Spark» и «DataFrame» конвертируем его в Spark. Мы можем сделать процедуры преобразования с DataFrame в обратном направлении (отфильтровать, извлечь и т.д.) и вызвать метод «toPandas», который его превратит обратно.

Нюанс: у нас есть система типов Java и система типов Python. Раньше нужно было прилагать много усилий для конвертации одного в другое, несмотря на их схожесть. Данную ситуацию пытается решить проект, который называется Apache Arrow.

Конвертация между JVM и нативным Python

Дедлайн 07 июля, 23:59 Мск

Apache Arrow – очень низкоуровневая платформа, которая описывает стандарт хранения данных в памяти.

Основная идея: это еще один стандарт хранения памяти. Apache Arrow пытается стать lingua franca для разных фреймворков, баз данных и компонентов инфраструктуры.

В примере, который мы рассмотрели ранее, можно заметить пятую строчку, где явно включается конвертация через PyArrow, то есть конвертация из Pandas DataFrame в Spark с преобразованием типов может осуществляться через PyArrow:

```
1. import numpy as np
2. import pandas as pd
3.
4. # Включить конвертацию типов через PyArrow
5. spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")
6.
7. pandas_df = pd.DataFrame(np.random.rand(100, 3))
8.
9. # Создание Spark DataFrame из Pandas DataFrame
10. df = spark.createDataFrame(pandas_df)
11.
12. # Сконвертировать Spark DataFrame обратно в Pandas DataFrame
13. result_pdf = df.select("*").toPandas()
14.
15. print("Pandas DataFrame result statistics:\n%s\n" % str(result_pdf.describe()))
```

Arrow для работы с DataFrame необязателен, но крайне желателен.

Создание сессий

Ранее мы говорили о существовании двух объектов, выдающихся драйвером для подключения к Spark:

- **SparkSession;**
- **SparkContext.**

Мы рассматривали именно SparkContext, так как он чаще всего используется в работе с RDD. В случае с DataFrame чаще всего используется SparkSession. Данный пример показывает то, как создать SparkSession в коде Python:

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession

# из существующего контекста
conf = SparkConf().setAppName("slurm_de").setMaster("local[8]")
sc = SparkContext(conf=conf)
spark = SparkSession(sc)
# напрямую
spark = SparkSession \
    .builder \
    .master("local[8]") \
    .appName("slurm_de") \
    .getOrCreate()
```

Это можно делать на основе SparkContext, где из первых 5-6 строчек импортировать все, что нужно. Сначала описываем через SparkConf конфигурацию кластера, на котором мы хотим все запускать. Поскольку запуск производится локально, добавляем локальный мастер. Далее создаем SparkContext, после чего создаем SparkSession, скормив ему SparkContext, как аргумент. Таким образом появятся нужные объекты.

Дедлайн 07 июля, 23:59 Мск

Шаги:

- Создаем SparkSession;
- Вызываем .builder;
- Прописываем через опции набор параметров, с которыми мы хотим создавать подключение;
- Вызываем getOrCreate.

Как вам урок?

Изучил, далее

Слёрм ©

+7 (495) 248-05-80

Лицензия №ДЛ-1368 от 22.08.2019

Политика конфиденциальности

Публичная оферта

Дедлайн 07 июля, 23:59 Мск