Тема 20. Обработка данных в памяти с помощью функций PySpark.

Цель занятия:

Изучить различные подходы к интеграции MongoDB с приложениями.

Учебные вопросы:

1. Основные компоненты PySpark
2. Работа с текстовыми файлами
3. Работа с CSV файлами
4. Работа с JSON файлами

1. Основные компоненты PySpark

1. SparkContext

Это главный объект, который позволяет взаимодействовать с кластером Spark. Он используется для инициализации Spark и получения доступа к ресурсам кластера.

Основные функции:

- Создание RDD (Resilient Distributed Dataset).
- Настройка конфигурации кластера.
- Установка соединения с кластером.

SparkContext — это основной объект для взаимодействия с кластером Spark. Он позволяет создавать RDD и управлять ресурсами.

```
from pyspark import SparkConf, SparkContext

# Настройка конфигурации и создание SparkContext
conf = SparkConf().setAppName("MyApp").setMaster("local")
sc = SparkContext(conf=conf)

# Пример создания RDD
data = [1, 2, 3, 4, 5]
rdd = sc.parallelize(data)

# Печать содержимого RDD
print(rdd.collect())  # Вывод: [1, 2, 3, 4, 5]

# Остановка SparkContext
sc.stop()
```

2. SparkSession

Это более высокоуровневый объект, который объединяет функциональность SparkContext, SQLContext и HiveContext. Он является единой точкой входа для работы с DataFrame и SQL.

Основные функции:

- Создание и работа с DataFrame.
- Чтение и запись данных в различных форматах (CSV, JSON, Parquet и др.).
- Выполнение SQL-запросов.
- Управление конфигурацией Spark.

SparkSession является единой точкой входа для работы с DataFrame и SQL-запросами.

```
from pyspark.sql import SparkSession

# Создание SparkSession
spark = SparkSession.builder \
    .appName("MyApp") \
    .getOrCreate()

# Чтение данных из CSV файла в DataFrame
df = spark.read.csv("path/to/file.csv", header=True, inferSchema=True)

# Печать схемы DataFrame
df.printSchema()

# Остановка SparkSession
spark.stop()
```

3. DataFrame

Это распределенная коллекция данных, организованная в виде таблицы, где данные представлены в виде строк и столбцов. DataFrame предоставляет возможности для работы с структурированными и полуструктурированными данными.

Основные функции:

- Выполнение SQL-подобных операций (фильтрация, агрегация, сортировка).
- Поддержка различных источников данных (SQL-базы данных, NoSQL, файлы и т. д.).
- Использование функций для трансформации данных.

DataFrame — это распределенная коллекция данных, организованная в виде таблицы, где данные представлены в виде строк и столбцов.

```
from pyspark.sql import SparkSession

# Создание SparkSession
spark = SparkSession.builder \
    .appName("MyApp") \
    .getOrCreate()

# Создание DataFrame из списка
data = [("Alice", 1), ("Bob", 2), ("Cathy", 3)]
columns = ["Name", "Id"]
df = spark.createDataFrame(data, schema=columns)

# Выполнение SQL-подобных операций
df.filter(df.Id > 1).show()  # Вывод: Bob и Cathy

# Остановка SparkSession
spark.stop()
```

4. RDD (Resilient Distributed Dataset)

Это основная абстракция данных в Spark. RDD представляет собой распределенный набор данных, который можно параллельно обрабатывать на кластере.

Основные функции:

- Поддержка ленивых вычислений (вычисления происходят только по мере необходимости).
- Возможность восстановления после сбоя (за счет хранения информации о трансформациях).
- Поддержка различных операций, таких как map, filter, reduce, и т. д.

RDD — это основная абстракция данных в Spark, представляющая собой распределенный набор данных, который можно параллельно обрабатывать.

```
from pyspark import SparkConf, SparkContext

# Настройка конфигурации и создание SparkContext
conf = SparkConf().setAppName("MyApp").setMaster("local")
sc = SparkContext(conf=conf)

# Пример создания RDD
data = ["apple", "banana", "cherry", "date"]
rdd = sc.parallelize(data)

# Применение трансформации map
rdd_upper = rdd.map(lambda x: x.upper())

# Печать результатов
print(rdd_upper.collect())  # Вывод: ['APPLE', 'BANANA', 'CHERRY', 'DATE']

# Остановка SparkContext
sc.stop()
```

Взаимосвязь компонентов

- **SparkContext** создает SparkSession, который затем используется для работы с DataFrame и RDD.
- **DataFrame** более высокоуровневый, чем RDD, и предлагает оптимизации, которые могут улучшить производительность операций.
- **RDD** является более гибким инструментом для работы с неструктурированными данными, тогда как DataFrame подходит для структурированных данных.

2. Работа с текстовыми файлами

Работа с текстовыми файлами в PySpark позволяет обрабатывать большие объемы данных.

Вот основные шаги для работы с текстовыми файлами:

Инициализация SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
      .appName("TextFileProcessing") \
      .getOrCreate()
```

2. Чтение текстового файла

```
# Чтение текстового файла в RDD
rdd = spark.sparkContext.textFile("path/to/textfile.txt")

# Чтение текстового файла в DataFrame
df = spark.read.text("path/to/textfile.txt")
```

3. Преобразования и действия

```
# Преобразование: разбить строки на слова
words_rdd = rdd.flatMap(lambda line: line.split(" "))

# Преобразование: сопоставить каждое слово с числом 1
word_pairs = words_rdd.map(lambda word: (word, 1))

# Сложение пар с одинаковыми ключами
word_counts = word_pairs.reduceByKey(lambda a, b: a + b)

# Действие: собрать результат
result = word_counts.collect()

for word, count in result:
    print(f"{word}: {count}")
```

Фильтрация

```
# Фильтрация строк, содержащих определенное слово
filtered_rdd = rdd.filter(lambda line: "keyword" in line)
```

Разделение строк

```
# Разделение строк на слова
words_rdd = rdd.flatMap(lambda line: line.split(" "))
```

Действия

```
# Сбор всех результатов
result = filtered_rdd.collect()

for line in result:
    print(line)
```

4. Запись результата

```
# Запись результата в файл
word_counts.saveAsTextFile("path/to/output")
```

5. Закрытие SparkSession

```
spark.stop()
```

3. Работа с CSV файлами

Работа с CSV файлами в PySpark позволяет легко обрабатывать табличные данные.

```
Date,Country,Units,Revenue
2019-01-08,USA,343,15461.36
2019-01-04,Panama,93,4681.26
2019-01-07,Panama,42,2220.36
2019-01-16,Brazil,103,1853.78
2019-01-17,USA,28,286.3
2019-01-24,Canada,372,24826.98
2019-01-26,Canada,61,1592.42
2019-01-28,Canada,264,3228.11
2019-01-13,Canada,27,257.97
2019-01-28,Brazil,323,3024.25
```

1. Инициализация SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("CSVFileProcessing") \
    .getOrCreate()
```

2. Чтение CSV файлов

```
# Чтение CSV файла
df = spark.read.csv(
    "path/to/file.csv",
    sep=",",  # Задайте разделитель
    header=True,  # Укажите, есть ли заголовки
    inferSchema=True  # Автоматически определить типы данных
)
```

3. Преобразование и очистка данных

```
# Выборка конкретных столбцов
selected_df = df.select("column1", "column2")

# Фильтрация данных
filtered_df = selected_df.filter(selected_df.column1 > 100)

# Добавление нового столбца
from pyspark.sql.functions import col, lit

transformed_df = filtered_df.withColumn("new_column", col("column2") * lit(10))
```

Замена значений в столбце

```
# Замена значений в столбце "status"
df_replaced = df.withColumn(
    "status",
    when(df["status"] == "old_value", "new_value").otherwise(df["status"])
)
```

Удаление строк с пропусками

```
# Удаление строк с любыми пропусками
df_dropped = df.dropna()
```

Заполнение пропусков

```
# Заполнение пропусков в конкретном столбце
df_filled = df.fillna({"column_name": "default_value"})
```

```
# Показать результат
df_filled.show()
```

4. Запись данных в CSV

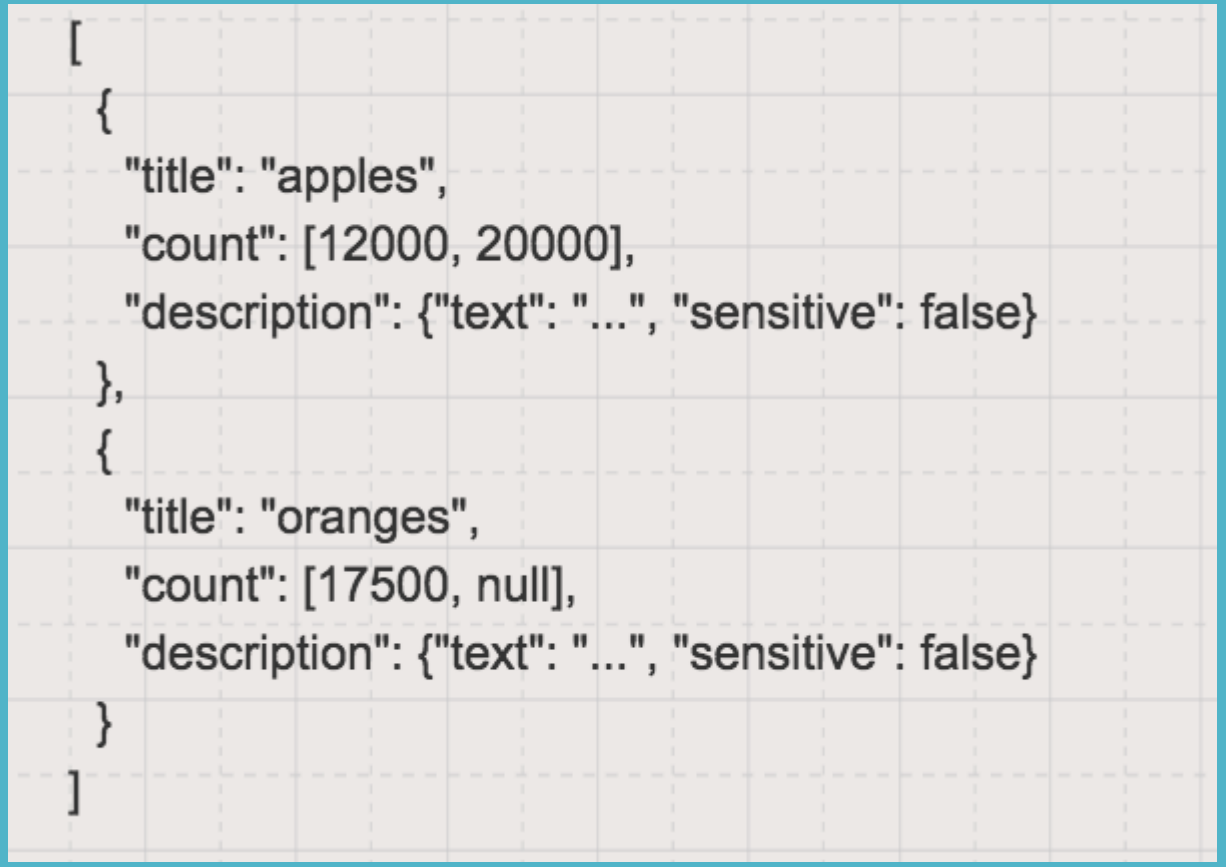
```
# Запись DataFrame в CSV файл
transformed_df.write.csv(
    "path/to/output.csv",
    sep=",",  # Задайте разделитель
    header=True,  # Добавьте заголовки
    mode="overwrite"  # Режим записи
)
```

Закрытие SparkSession

```
spark.stop()
```

4. Работа с JSON файлами

Работа с JSON файлами в PySpark позволяет обрабатывать сложные структуры данных.



1. Инициализация SparkSession

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
      .appName("JSONFileProcessing") \
      .getOrCreate()
```

2. Чтение JSON файлов

```
# Чтение JSON файла
df = spark.read.json("path/to/file.json")
```

3. Обработка вложенных данных

```
# Предположим, у вас есть столбец "info" с вложенными данными
df.select("info.name", "info.age").show()
```

Работа с массивами

```
from pyspark.sql.functions import explode

# Предположим, у вас есть столбец "items", который является массивом
df.select("id", explode("items").alias("item")).show()
```

4. Запись данных в JSON

```
# Запись DataFrame в JSON файл
df.write.json("path/to/output.json", mode="overwrite")
```

Закрытие SparkSession

```
spark.stop()
```

Домашнее задание:

1. Повторить материал лекции.

Список литературы:

1. В. Ю. Кара-ушанов SQL — язык реляционных баз данных
2. А. Б. ГРАДУСОВ. Введение в технологию баз данных
3. А.Мотеев. Уроки MySQL

Материалы лекций:

https://github.com/ShViktor72/Education

Обратная связь:

colledge20education23@gmail.com