Traitement des données structurées avec *Spark*

Données structurées sur Spark: DataSet/DataFrame/Spark SQL

- Ce sont des RDDs structurées dans lesquelles les données sont sous format tabulaire
- Une collection immutable d'objets fortement typés qui sont mappés vers un schéma relationnel
  - Elles peuvent être assimilées à des tables relationnelles
- En interne:
  - ***Catalyst***: application des optimisations inspirées des technique d'évaluation de requêtes dans les systèmes relationnelles grâce à la connaissance du schéma et des types de données
  - ***Tungsten***: représentation en format binaire des données pour minimiser les opérations de dé-sérialisation
- 2 types:
  - ***DataSet***: quand le type des colonnes est connu
  - ***DataFrame***: un D*ataSet* avec des lignes dont le schéma n'est pas connu, i.e. de type générique appelé ***Row***

DataFrame & DataSet

Le moteur Spark SQL permet d'effectuer des requêtes similaires à SQL sur des données distribuées. Il est dédié aux traitements de données structurées et semi structurées via l'API **de haut niveau**: DataFrame/DataSet.

DataFrame

- Collection de donnée distribuée.
- Contrairement aux RDD, les données sont organisées par colonne
- Conceptuellement, un DataFrame est l'équivalent d'une table dans une base de données relationnelle.
- Une DataFrame est non typé: toutes les entrées sont de type générique appelé ***Row***
- Une DataFrame supporte des transformations basées sur des paradigmes fonctionnels ou relationnels.
- Elle peut être créée à partir:
  1. d'une RDD.
  2. d'une table (hive, SGBD, etc.)
  3. d'une ou plusieurs DataFrame via des opérations de transformation.
- Disponible en Java, Scala, Python et R

DataSet

- Une DataSet est une DataFrame fortement typé.
- Disponible seulement en Scala et Java.

API DataFrame (1)

```java
public class Main {
    public static void main(String[] args) {
        SparkSession spark = SparkSession.builder()
                .appName("Java Spark SQL basic example")
                .getOrCreate();
        // Read CVS file
        Dataset<Row> dfFromCSV = spark.read()
                .format("csv")
                .option("header", "true")
                .load("csvfile.csv");
        // Read Parquet file
        Dataset<Row> dfFromParquet = spark.read()
                .parquet("parquetfile.parquet");
        // Read ORC File
        Dataset<Row> dfFromORC = spark.read()
                .orc("orcfile.orc");
    }
}
```

Construction à partir de la lecture d'un fichier

```java
public class DFRDD {

    public static void main (String[] args) {
        SparkSession spark = SparkSession
                    .builder()
                    .appName("Java Spark SQL basic example")
                    .getOrCreate();
        JavaRDD<Person> peopleRDD = spark.read()
            .textFile("examples/src/main/resources/people.txt")
            .javaRDD()
            .map(new Function<String, Person>() {
                private static final long serialVersionUID = 1L;

                @Override
                public Person call(String line) throws Exception {
                    String[] parts = line.split(",");
                    Person person = new Person();
                    person.setName(parts[0]);
                    person.setAge(Integer.parseInt(parts[1].trim()));
                    return person;
                }
            });
        //Apply a schema to an RDD of JavaBeans to get a DataFrame
        Dataset<Row> peopleDF = spark.createDataFrame(peopleRDD,
                Person.class);
        peopleDF.show();
    }
}
```

Construction à partir d'un RDD

Manipulation de DataFrames

```java
Dataset<Row> selectedColRows = dfFromCSV.select(col("column_name"),col("age").plus(1));

Dataset<Row> filtredRowd = dfFromCSV.filter(col("column_name").geq(13));

Dataset<Row> results = dfFromCSV.groupBy("column_name").avg("column_name") ;

JavaRDD<Row> rowJavaRDD = dfFromCSV.toJavaRDD();
```

API DataFrame (2)

```java
import java.util.ArrayList;
import java.util.List;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.sql.*;
import org.apache.spark.sql.types.DataTypes;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;

public class RDDtoDFSchemaSpec {
    public static void main (String[] args)  {
        SparkSession spark = SparkSession.builder().appName("Java Spark SQL basic example").getOrCreate();
        // Creer un RDD
        JavaRDD<String> peopleRDD = spark.sparkContext().textFile("examples/src/main/resources/people.txt", 1).toJavaRDD();

        String schemaString = "name age";

        // Generation du schema
        List<StructField> fields = new ArrayList<StructField>();
        for (String fieldName : schemaString.split(" ")) {
            StructField field = DataTypes.createStructField(fieldName, DataTypes.StringType, true);
            fields.add(field);
        }
        StructType schema = DataTypes.createStructType(fields);

        // Convert records of the RDD (people) to Rows
        JavaRDD<Row> rowRDD = peopleRDD.map(new Function<String, Row>() {
            @Override
            public Row call(String record) throws Exception {
                String[] attributes = record.split(",");
                return RowFactory.create(attributes[0], attributes[1].trim());
            }
        });
        // Apply the schema to the RDD
         Dataset<Row> peopleDataFrame = spark.createDataFrame(rowRDD, schema);
    }
}
```

Définir programmatiquement le schéma d'un DataFrame

API DataSet

```java
import org.apache.spark.sql.*;
public class DataSETest {
    public static void main (String[] args){
        SparkSession spark = SparkSession
                .builder()
                .appName("Java Spark SQL basic example")
                .getOrCreate();

        //Read CVS file
        Encoder<Person> encoder = Encoders.bean(Person.class);
        Dataset<Person> dsFromCSV = spark.read().format("csv")
                                        .option("header", "true")
                                        .load("csvfile.csv").as(encoder);
        //Read Parquet file
        Dataset<Person> dfFromParquet = spark.read().parquet("parquetfile.parquet").as(encoder);

        //Read ORC File
        Dataset<Person> dfFromORC = spark.read().orc("orcfile.orc").as(encoder); ;
    }
}
```

Construction à partir de la lecture d'un fichier

API DataSet

Construction à partir d'un RDD

```java
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.sql.*;
import org.apache.spark.api.java.function.Function;
public class DFRDD {

    public static void main (String[] args) {
        SparkSession spark = SparkSession
                .builder()
                .appName("Java Spark SQL basic example").getOrCreate();
        JavaRDD<Person> peopleRDD = spark.read()
                .textFile("examples/src/main/resources/people.txt").javaRDD()
                .map(new Function<String, Person>() {
                    @Override
                    public Person call(String line) throws Exception {
                        String[] parts = line.split(",");
                        Person person = new Person();
                        person.setName(parts[0]);
                        person.setAge(Integer.parseInt(parts[1].trim()));
                        return person;
                    }
                });
     Encoder<Person> encoder = Encoders.bean(Person.class);
     Dataset<Row> peopleDF = spark.createDataFrame(peopleRDD, Person.class);
     // DataFrame --> DataSet
     Dataset<Person> datasetFromDF = peopleDF.as(encoder);
     // RDD --> DataSet
     Dataset<Person> datasetFromRDD = spark.createDataset(peopleRDD.rdd(), encoder);
    }
}
```

API DataSet

```
Dataset<Person> selectedColRows = datasetFromDF.select(col("column_name"),col("age")).as(encoder) ;
Dataset<Person> filteredDS = datasetFromDF.filter(new FilterFunction<Person>() {
    @Override
    public boolean call(Person person) throws Exception { return false; } });
Dataset<Integer> results =
datasetFromDF.groupBy("column_name").avg("column_name").as(Encoders.INT());

// DataSet --> DataFrame
Dataset<Row> dSToDF = selectedColRows.toDF();

// DataSet --> JavaRDD
JavaRDD<Person> dsToJavaRDD = selectedColRows.toJavaRDD();
```


Spark SQL

- Spark SQL est un module de Spark permettant de requêter et d'appliquer des transformations sur une collection de données en utilisant un langage dit SQL-Like
- Il peut être également vu comme un moteur SQL distribué
- Avec les trois APIs DataFrame, DataSet et SQL, les développeurs ont la capacité de choisir l'API qui répondrait le mieux à leurs besoins et switcher d'une API à une autre dans la même application
  - Une DataFrame ou un DataSet peut être exposé comme une table relationnelle qui peut être manipulée en utilisant un langage SQL-Like

```
Encoder<Person> encoder = Encoders.bean(Person.class);
Dataset<Row> dfFromRDD = spark.createDataFrame(peopleRDD,
Person.class);
// DataFrame --> DataSet
Dataset<Person> datasetFromDF = dfFromRDD.as(encoder);

// RDD --> DataFrame
Dataset<Person> datasetFromRDD = spark.createDataset(peopleRDD.rdd(),
encoder);
//Creer les tables relationnelles
try {
    dfFromRDD.createGlobalTempView("people_df");
    datasetFromRDD.createGlobalTempView("people_ds");
    datasetFromDF.createOrReplaceTempView("people_local_temp");
} catch (AnalysisException e) {
    e.printStackTrace();
}

// Requeter les tables crees
Dataset<Row> result_df = spark.sql("select * from people_df");
Dataset<Row> result_ds = spark.sql("select * from people_ds");
Dataset<Row> result_ds_tmp = spark.sql("select * from
people_local_temp");

// Supprimer les tables
spark.catalog().dropGlobalTempView("people_local_temp") ;
spark.catalog().dropGlobalTempView("people_df") ;
```

Comparaison RDD, DataFrame/DataSet

	RDD	Dataframe	Dataset
Représentation des données	• Ensemble d'objets JAVA	• Une collection organisée en colonnes nommées	• Une extension des DataFrame mais fortement typées
Interopérabilité	• Peut créer un DF à partir d'une RDD et vice-versa	• Peut être créée à partir d'un RDD • Pas de conversion DF vers RDD	• Peut être créé à partir d'un RDD ou DF • Peut être converti vers un RDD
Erreur de Syntaxe	• À la compilation	• À la compilation	• À la compilation
Erreur d'analyse	• À la compilation	• À l'exécution	• À la compilation
Langages	• Scala, Java, Python	• Scala, Java, python, R	• Scala, Java
Paradigme d'interrogation	• Fonctionnelle	• Fonctionnelle, DSL, SQL	• Fonctionnelle, DSL, SQL
Sérialisation	• Utilise la sérialisation de JAVA pour stocker et distribuer les objets (couteux)	• Les données sont transformées en format binaire et application des transformation en utilisant Tungsten • Génération dynamique de bytecode	• Utilise les encodeurs et stocke les données sous format tabulaire en utilisant Tungsten
Optimisation	• Pas d'optimisation	• Utilise Tungsten et Catalyst	• Utilise Tungsten et Catalyst
Garbage collection	• Oui	• Pas besoin	• Pas besoin de GC pour détruire les objets car la sérialisation est en offheap et géré par Tungsten
Utilisation de la mémoire	• Pas très efficace à cause de la sérialisation	• Efficace car utilise une sérialisation en off-heap	• Très efficace car les opérations sont directement faites sur les données sérialisées

RDD, DataFrame/DataSet: quoi utiliser quand ?

RDD

- **Type de données à traiter:**
  - Les données **ne sont pas structurées** (media, textes).
  - Pas besoin d'imposer un schéma ou une structure lors du traitement de données.
  - On veut utiliser la programmation fonctionnelle plutôt qu'un DSL
- **Optimisation ou contrôle bas niveau:**
  - Besoin d'avoir un contrôle niveau bas sur les données.
  - On peut renoncer à certains avantages en termes d'optimisation de performance disponibles sur l'API DataFrames/Dataset.

DataFrame/DataSet

- **Type de données à traiter:**
  - Les traitements exige à la fois des expressions du haut niveau (sql, aggregation, sum, etc) et des fonctions lambda sur des **données semi-structurées ou structurées.**
  - Besoin d'une sémantique riche, des abstractions de haut niveau et des API DSL.
  - Besoin d'utiliser des objets typés (**DataSet**).
- **Optimisation ou contrôle bas niveau:**
  - Représentation binaire des données
  - Tirez parti de l'optimisation Catalyst et de la génération de code efficace de Tungsten.

Spark SQL

https://towardsdatascience.com/sql-at-scale-with-apache-spark-sql-and-dataframes-concepts-architecture-and-examples-c567853a702f