Spark SQL

Spark SQL

- Spark SQL是Spark生态系统中重要的组件，其前身为Shark。

- Shark是Spark上的数据仓库，最初设计成与Hive兼容，但是该项目于2014年开始停止开发，转向Spark SQL。Spark SQL全面继承了Shark，并进行了优化。

Shark

- Shark 是Spark on Spark
- 为了能兼容Hive，Shark在HiveQL解析方面将MapReduce作业转换为Spark作业
- 面临的问题：
  - 执行计划完全依赖于Hive，不方便添加优化策略
  - Shark继承了大量Hive代码不方便维护
- Spark SQL在Shark原有的

Spark SQL

- Spark SQL中增加了SchemaRDD，使得用户可以在Spark中使用SQL，数据即可以是来自于RDD也可以是来自于Hive、HDFS或者外部数据
- Spark SQL目前提供Java、Scala和Python三种语言的API。和支持SQL-92的规范

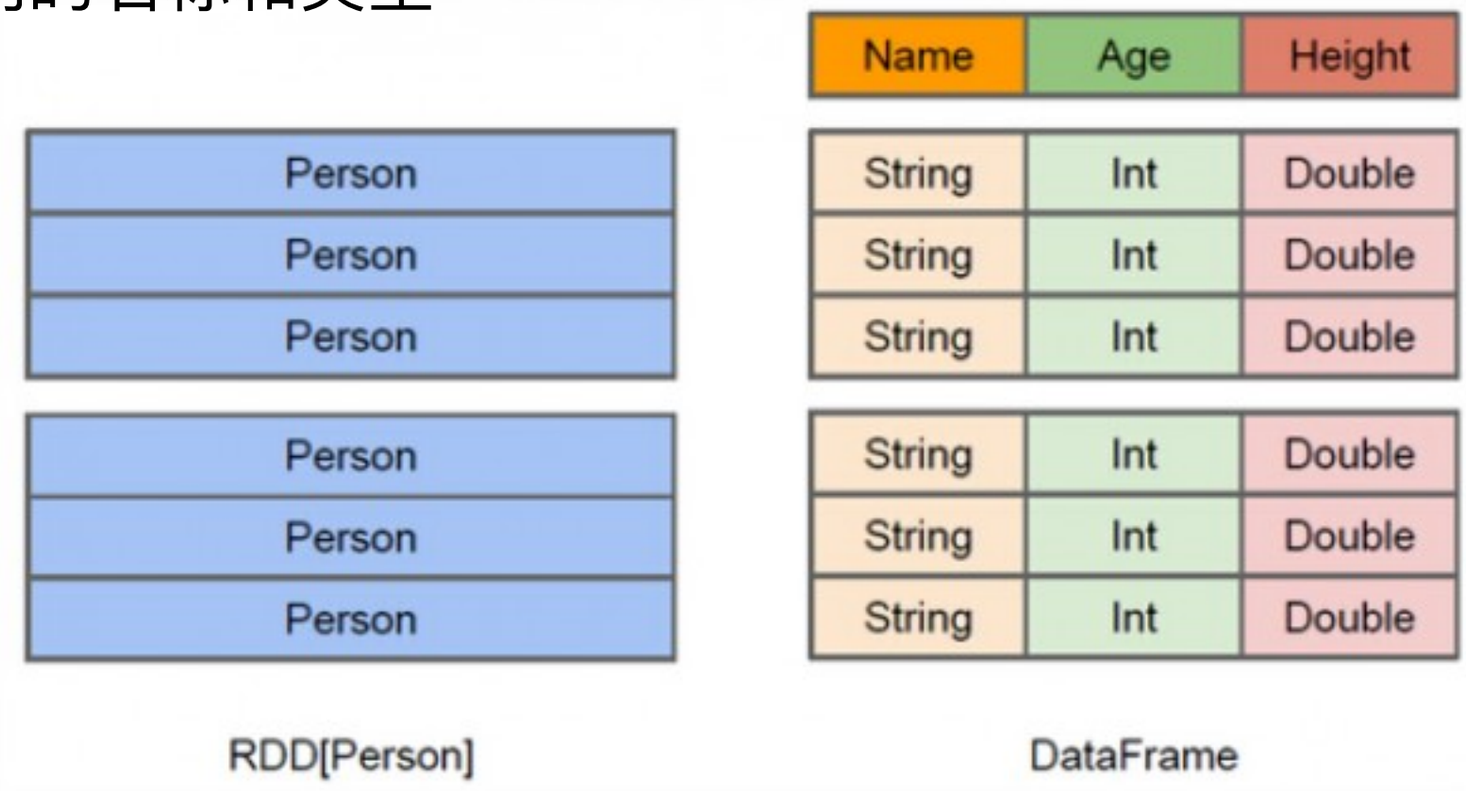
DataFrame

- DataFrame是具有结构化的RDD（SchemaRDD）
- DataFrame类似一张数据库中的二维表
- DataFrame仍然是惰性操作，只记录操作的过程（DAG图），只有到了执行步骤的时候才会真正的执行

DataFrame与RDD

- RDD中存储的是Person的实例，RDD并不知道它内部是什么

- DataFrame是基于分布式RDD对象的，但是指明了每一列存储的内容，每一列存储的Row对象。Spark 可以清楚的知道都存储了哪些列，列的名称和类型



DataFrame的创建-SparkSession

- Spark 1.X实际上有两个Context， SparkContext和SQLContext，它们负责不同的功能。 前者专注于对Spark的中心抽象进行更细粒度的控制，而后者则专注于Spark SQL等更高级别的API。
  - Spark2.0之前通过SQLContext进行SparkSQL操作

- 在Spark 2.X中，这两个API被集成到SparkSession中。 但是，这两个API仍然存在。我们尽可能通过SparkSession访问它们

DataFrame的创建-SparkSession

- SparkSession中的SparkContext对象可以告诉Spark程序如何连接访问Spark集群。 我们可以通过它在Spark中使用一些较低级API，例如创建一个RDD，累加器和广播变量等。在2.0之前我们是通过SparkContext进行操作的

- 在大多数情况下，我们不需要显式初始化SparkContext; 而尽量通过SparkSession来访问它。

- SparkSession支持从不同的数据源加载数据，并把数据转换成DataFrame，并且支持把DataFrame转换成SQLContext自身中的表，然后使用SQL语句来操作数据。

DataFrame的创建-SparkSession

创建SparkSession对象

[ ]:
```
# Creating a SparkSession in Python
from pyspark.sql import SparkSession
# getOrCreate()如果存在则获取，如果不存在则创建
spark = SparkSession.builder.master("local").appName("Spark SQL").getOrCreate()
```

DataFrame的创建-从json中读取

```
# 从JSON文件中创建DataFrame
df = spark.read.json("./data/people.json")
# show() 打印DataFrame中的所有内容
df.show()
```

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

DataFrame的创建-从csv中读取

[6]:
```
# 从CSV文件中创建
df = spark.read.csv('./data/people.csv', sep = ',', header = True)
```

[7]:
```
df.show()
```

```
+-----+---+---------+
| name|age|      job|
+-----+---+---------+
|Jorge| 30|Developer|
|  Bob| 32|Developer|
+-----+---+---------+
```

DataFrame的创建-从RDD中转换

使用编程方式创建DataFrame

]:
```
from pyspark.sql.types import Row
from pyspark.sql.types import StructType
from pyspark.sql.types import StructField
from pyspark.sql.types import StringType

# 读取txt生成RDD
rdd = spark.sparkContext.textFile('./data/people.txt')\
           .map(lambda line: line.split(','))


# 定义Schema中的字段
# StructField(fieldName, DataType(), nullable = True)
schema = StructType([
    StructField("name", StringType(), True),
    StructField("age", StringType(), True)
])


# Apply the schema to the RDD and Create DataFrame
df = spark.createDataFrame(rdd, schema)
df.show()
```

```
+-------+---+
|   name|age|
+-------+---+
|Michael| 29|
|   Andy| 30|
| Justin| 19|
+-------+---+
```

DataFrame的常用方法

[56]:
```
df.show()
```

```
+-------+---+
|   name|age|
+-------+---+
|Michael| 29|
|   Andy| 30|
| Justin| 19|
+-------+---+
```

[21]:
```
df_json.show()
```

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

[22]:
```
# DataFrame 常见操作
# 打印DataFrame的模式
df_json.printSchema()
```

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

DataFrame的常用方法

[24]:
```
# DataFrame列的选择，并且支持表达式
df_json.select(df_json.name, df_json.age + 1).show()
```

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

[26]:
```
# DataFrame 条件过滤
df_json.filter(df_json.age > 20 ).show()
```

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

[27]:
```
# 分组聚合
df_json.groupBy('age').count().show()
```

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

DataFrame的常用方法

28]:
```
# 排序
df_json.sort(df_json.age.desc()).show()
```

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

30]:
```
# 多列排序
df_json.sort(df_json.age.desc(), df_json.name.asc()).show()
```

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

35]:
```
# 对列进行重命名
df_json.select(df_json.name.alias("username"), df_json.age).show()
```

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

DataFrame的常用方法

[37]:
```
# 查看前两行
df_json.head(2)
```

[37]:
```
[Row(age=None, name='Michael'), Row(age=30, name='Andy')]
```

[38]:
```
# 查看列名
df_json.columns
```

[38]:
```
['age', 'name']
```

[39]:
```
# 查看统计信息
df_json.describe().show()  #查看内容---描述统计值（最大，最小，频数等）
```

```
+-------+------------------+-------+
|summary|               age|   name|
+-------+------------------+-------+
|  count|                 2|      3|
|   mean|              24.5|   null|
| stddev|7.7781745930520225|   null|
|    min|                19|   Andy|
|    max|                30|Michael|
+-------+------------------+-------+
```

[40]:
```
df_json.describe('age').show() #查看其中一列
```

```
+-------+------------------+
|summary|               age|
+-------+------------------+
|  count|                 2|
|   mean|              24.5|
| stddev|7.7781745930520225|
|    min|                19|
|    max|                30|
+-------+------------------+
```

DataFrame的常用方法

[41]:
```
# 筛选特定样本
# 选择name是'Michael'的值
df_json.filter(df_json['name'] == 'Michael').show()
```

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
+----+-------+
```

[42]:
```
# 只显示name那一列
df_json.filter(df_json['name'] == 'Michael').select('name').show()
```

```
+-------+
|   name|
+-------+
|Michael|
+-------+
```

[44]:
```
# 去除重复
df_json.select('name').distinct().show()
# 计算不重复值有多少
df_json.select('name').distinct().count()
```

```
+-------+
|   name|
+-------+
|Michael|
|   Andy|
| Justin|
+-------+
```

DataFrame的常用方法

[45]:
```
# 处理缺失值
# 缺失值用0代替
df_json.fillna(0).show()
```

```
+---+-------+
|age|   name|
+---+-------+
|  0|Michael|
| 30|   Andy|
| 19| Justin|
+---+-------+
```

[46]:
```
# 直接删除缺失值
df_json.dropna().show()
```

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

[49]:
```
# 转换类型
from pyspark.sql.types import DoubleType
df_json = df_json.withColumn('age', df_json['age'].cast(DoubleType()))
df_json.printSchema()
df_json.show()
```

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

+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|30.0|   Andy|
|19.0| Justin|
```

DataFrame的常用方法-使用SQL

```
2]: df_json.show()
```

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|30.0|   Andy|
|19.0| Justin|
+----+-------+
```

```
3]: # 注册DataFrame为临时表，  可以进行SQL的查询
df_json.createOrReplaceTempView("people")
spark.sql("select * from people").show()
```

```
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|30.0|   Andy|
|19.0| Justin|
+----+-------+
```

```
]: spark.sql("select * from people where name = 'Andy'").show()
```

```
+----+----+
| age|name|
+----+----+
|30.0|Andy|
+----+----+
```

```
]: res_df = spark.sql("select * from people where name = 'Andy'")
res_df.rdd.map( lambda attributes : (1 + attributes[0] , attributes[1] + 'a')).collect()
```

```
]: [(31.0, 'Andya')]
```