SparkSQL

Spark SQL允许使用SQL或熟悉的DataFrame API查询Spark程序内的**结构化数据**。DataFrames和SQL提供了访问各种数据源的常用方式，包括**Hive，Avro，Parquet，ORC，JSON和JDBC**。server模式为BI工具提供行业标准的JDBC和ODBC连接。

参考

Datasets and DataFrames

RDD、DataSet、DataFrame的区别

RDD

优点:

1. 编译时类型安全：编译时就能检查出类型错误
2. 面向对象的编程风格：直接通过类名点的方式来操作数据

缺点:

1. 序列化和反序列化的性能开销：无论是集群间的通信, 还是IO操作都需要对对象的结构和数据进行序列化和反序列化.
2. GC的性能开销：频繁的创建和销毁对象, 势必会增加GC

DataFrame

DataFrame引入了schema和off-heap，解决了RDD的缺点, 但是却丢了RDD的优点。DataFrame不是类型安全的，API也不是面向对象风格的。

- schema : RDD每一行的数据，结构都是一样的。这个结构就存储在schema中。Spark通过schame就能够读懂数据，因此在通信和IO时就只需要序列化和反序列化数据，而结构的部分就可以省略了。
- off-heap : 意味着JVM堆以外的内存，这些内存直接受操作系统管理（而不是JVM）。Spark能够以二进制的形式序列化数据(不包括结构)到off-heap中，当要操作数据时，就直接操作off-heap内存。由于Spark理解schema，所以知道该如何操作。

DataSet

DataSet结合了RDD和DataFrame的优点, 并带来的一个新的概念Encoder

当序列化数据时, Encoder产生字节码与off-heap进行交互, 能够达到按需访问数据的效果, 而不用反序列化整个对象。

DataSet是Spark2的新特性，从接口和1.6中的DataFrame完全兼容。

Quick Start

SparkSession

SparkSession是DataSet和DataFrame的API切入点，等同于SparkContext之于RDD以及SparkStreamingContext之于流处理。

```
import
```

```
val spark = SparkSession
  .builder()
  .appName("Spark SQL basic example")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()

// For implicit conversions like converting RDDs to DataFrames
import spark.implicits._
```

Creating DataFrames

可以从RDD、Hive table、或者Spark data sources中创建DataFrames。

```
### spark2-shell --driver-class-path /usr/share/java/mysql-connector-java.
```

```
val jdbcDF = spark.read.format("jdbc")
.option("url", "jdbc:mysql://172.24.33.20:63306/scm")
.option("driver", "com.mysql.jdbc.Driver")
.option("user", "scm")
.option("password", "scm123qaz")
.option("dbtable", "hosts")
.load()

jdbcDF.show()
jdbcDF.printSchema()
jdbcDF.select("IP_ADDRESS").show()

### 读HDFS上的Parquet文件
val parquetFile = spark.read.parquet("file:///opt/users.parquet")
parquetFile.show()

```

使用DataSet

SparkSQL支持将存在的RDD装换为Datasets

```
case class Person(name: String, age: Long)

// Encoders are created for case classes
val caseClassDS = Seq(Person("Andy", 32)).toDS()
caseClassDS.show()
// +----+---+
// |name|age|
// +----+---+
// |Andy| 32|
// +----+---+

// Encoders for most common types are automatically provided by importing
val primitiveDS = Seq(1, 2, 3).toDS()
primitiveDS.map(_ + 1).collect() // Returns: Array(2, 3, 4)

// DataFrames can be converted to a Dataset by providing a class. Mapping
val path = "examples/src/main/resources/people.json"
```

```
val peopleDS = spark.read.json(path).as[Person]
peopleDS.show()
// +----+-------+
// | age| name|
// +----+-------+
// |null|Michael|
// | 30| Andy|
// | 19| Justin|
// +----+-------+
```