> 概况能力很重要，需要锻炼

SparkSql之DataFrame和DataSet

SparkSQL概述

> Spark SQL是Spark用来处理结构化数据的一个模块，
> 它提供了2个编程抽象：DataFrame和DataSet，并且作为分布式SQL查询引擎的作用。
>
> Hive是将Hive SQL转换成MapReduce然后提交到集群上执行，大大简化了编写MapReduc的程序的复杂性，由于MapReduce这种计算模型执行效率比较慢。所有Spark SQL的应运而生，它是将Spark SQL转换成RDD，然后提交到集群执行，执行效率非常快！

SparkSQL特点

Integrated

无缝整合了SQL查询和Spark编程。

Uniform Data Access

使用相同的方式连接不同的数据源

Hive Integration

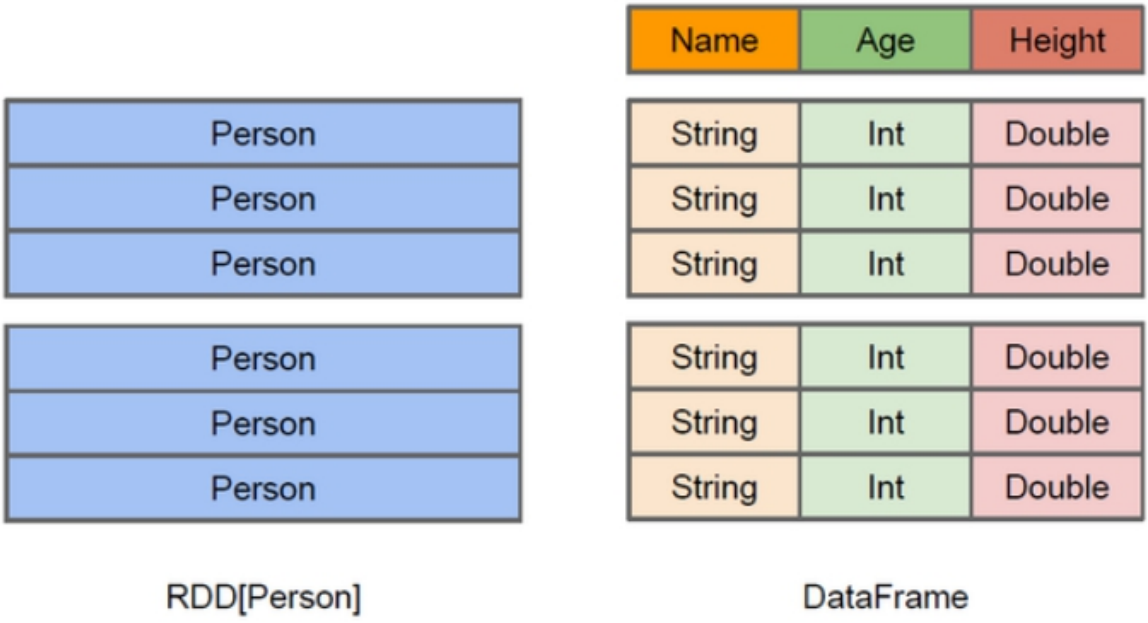
在已有的仓库上直接运行SQL或者HiveQL

Standard Connectivity

通过JDBC或者ODBC来进行连接

DataFrame

> 在Spark中，DataFrame是一种以RDD为基础的**分布式数据集**，类似于传统数据库中的二维表格。DataFrame与RDD的主要区别在于，**前者带有schema元信息**，即DataFrame所表示的二维表数据集的每一列都带有名称和类型。这使得Spark SQL得以洞察更多的结构信息，从而对藏于DataFrame背后的数据源以及作用于DataFrame之上的变换进行了针对性的优化，最终达到大幅提升运行时效率的目标。反观RDD，由于无从得知所存数据元素的具体内部结构，Spark Core只能在stage层面进行简单、通用的流水线优化。



DataFrame也是懒执行的，但性能上比RDD要高，主要原因：

优化的执行计划，即查询计划通过Spark catalyst optimiser进行优化。比如下面一个例子:

```
users.join(events, users("id") === events("uid"))
     .filter(events("date") > "2015-01-01")
```

DataSet

> DataSet是分布式数据集合。DataSet是Spark 1.6中添加的一个新抽象，是DataFrame的一个扩展。它提供了RDD的优势（**强类型，使用强大的lambda函数的能力**）以及Spark SQL优化执行引擎的优点。DataSet也可以使用功能性的转换（操作map，flatMap，filter等等）。

1）是DataFrame API的一个扩展，是SparkSQL最新的数据抽象；

2）用户友好的API风格，既具有类型安全检查也具有DataFrame的查询优化特性；

3）**用样例类来对DataSet中定义数据的结构信息**，样例类中每个属性的名称直接映射到DataSet中的字段名称；

4）**DataSet是强类型的。**比如可以有DataSet[Car]，DataSet[Person]。

> 后面几乎都使用Dataset

SparkSession

> 在老的版本中，SparkSQL提供两种SQL查询起始点：

> 一个叫SQLContext，用于Spark自己提供的SQL查询；
>
> 一个叫HiveContext，用于连接Hive的查询。
>
> SparkSession是Spark最新的SQL查询起始点，实质上是**SQLContext和HiveContext的组合**，所以在SQLContex和HiveContext上可用的API在SparkSession上同样是可以使用的。SparkSession内部封装了sparkContext，所以**计算实际上是由sparkContext完成的。**

DataFrame创建

（一）通过Spark的数据源进行创

Spark支持的数据源:

```
scala> spark.read.
csv   format   jdbc   json   load   option   options   orc   parquet   schema   table   text   textFile
```

将Json文件数据加载到DF中进行展示：

注意这里的文件路径是Hdfs上的路径，而不是Linux本地路径

```
scala> val phoneDf = spark.read.json("/data/phone.json")
phoneDf: org.apache.spark.sql.DataFrame = [name: string, price: bigint]

scala> phoneDf.show
+------+-----+
|  name|price|
+------+-----+
|Huawei| 6000|
|Xiaomi| 3000|
|  Oppo| 4000|
|  Vivo| 4500|
+------+-----+
```

将Csv文件数据加载到DF中进行展示：

```
scala> val agesDf = spark.read.csv("/data/ages.csv")
agesDf: org.apache.spark.sql.DataFrame = [_c0: string, _c1: string]

scala> agesDf.show
+-------+---+
|    _c0|_c1|
+-------+---+
|    Joe| 20|
|    Tom| 30|
|Hyukjin| 25|
+-------+---+
```

> 通过上面两个示例演示，我们知道了SparkSql提供一些非常好用的读取数据的接口，只需要将数据放到Hdfs上，然后将路径信息传入到read.xxx(“文件路径”)即可把数据封装到DF中，然后就可以使用DF里面的方法对数据进一步处理了。

（二）从一个存在的RDD进行转换

从RDD到DataFrame

> 涉及到RDD，DataFrame，DataSet之间的操作时，需要导入:import spark.implicits._ 这里的spark不是包名，而是表示SparkSession 的那个对象. 所以必须先创建SparkSession对象再导入. implicits是一个内部object。

1.手动转换

创建 personRDD，并查看数据

```
scala> val personRDD = sc.textFile("/data/person.txt")
personRDD: org.apache.spark.rdd.RDD[String] = /data/person.txt MapPartitionsRDD[21] at textFi

scala> personRDD.collect
res4: Array[String] = Array(Jack,man,20, Marry,women, 26, Tom,man,16)
```

切割 RDD，形成人员分组

```
scala> val personRDD2 = personRDD.map(line=>{ val params = line.split(",");(params(0),params(1),params(2))})
personRDD2: org.apache.spark.rdd.RDD[(String, String, String)] = MapPartitionsRDD[22] at map at <console>:26

scala> personRDD2.collect
res5: Array[(String, String, String)] = Array((Jack,man,20), (Marry,women," 26"), (Tom,man,16))
```

转换为 DF，并展示结果

```
scala> val personDF = personRDD2.toDF("name","sex","age")
personDF: org.apache.spark.sql.DataFrame = [name: string, sex: string ... 1 more field]

scala> personDF.show
+-----+-----+---+
| name|  sex|age|
+-----+-----+---+
| Jack|  man| 20|
|Marry|women| 26|
|  Tom|  man| 16|
+-----+-----+---+
```

2.样例类转换（推荐）

创建样例类 Person

```
scala> case class Person(name:String,sex:String,age:String)
defined class Person
```

使用样例类封返回的参数，并查看结果

```
scala> val personRDD2 = personRDD.map(line=>{ val params = line.split(",");Person(params(0),params(1),params(2))})
personRDD2: org.apache.spark.rdd.RDD[Person] = MapPartitionsRDD[42] at map at <console>:31

scala> personRDD2.collect
res12: Array[Person] = Array(Person(Jack,man,20), Person(Marry,women, 26), Person(Tom,man,16))
```

转换为 DF，并查看结果

```
scala> val personDF = personRDD2.toDF
personDF: org.apache.spark.sql.DataFrame = [name: string, sex: string ... 1 more field]

scala> personDF.show
+-----+-----+---+
| name|  sex|age|
+-----+-----+---+
| Jack|  man| 20|
|Marry|women| 26|
|  Tom|  man| 16|
+-----+-----+---+
```

3.通过API方式转换

```
import org.apache.spark.SparkContext
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.types.{IntegerType, StringType, StructField,
StructType}
import org.apache.spark.sql.{DataFrame, Dataset, Row, SparkSession}
```

```
object RddToDataFrame {
    def main(args: Array[String]): Unit = {
        val spark: SparkSession = SparkSession.builder()
            .master("local[*]")
            .appName("RddToDataFrame ")
            .getOrCreate()
        val sc: SparkContext = spark.sparkContext
        val rdd: RDD[(String，Int)] = sc.parallelize(Array(("tom"，10)，("jack"，
20)，("marry"，40)))

        // DataFrame其实就是 DataSet[Row]
        val rowRdd: RDD[Row] = rdd.map(x => Row(x._1，x._2))

        // 创建 StructType 类型
        val types = StructType(Array(StructField("name"，StringType)，
StructField("age"，IntegerType)))
        val df: DataFrame = spark.createDataFrame(rowRdd，types)
        df.show

    }
}
```

(三）从Hive Table进行查询得到

> ApacheHive是Hadoop上的SQL引擎，SparkSQL编译时可以包含Hive支持，也可以不包含。包含Hive支持的SparkSQL可以支持Hive表访问、UDF(用户自定义函数)以及Hive查询语言(HiveQL/HQL)等。需要强调的一点是，如果要在SparkSQL中包含Hive的库，**并不需要事先安装Hive。**一般来说，最好还是在编译SparkSQL时引入Hive支持，这样就可以使用这些特性了。如果你下载的是二进制版本的Spark，它应该已经在编译时添加了Hive支持。

> 若要把SparkSQL连接到一个部署好的Hive上，你必须**把hive-site。xml复制到Spark的配置文件目录中($SPARK_HOME/conf)。**即使没有部署好Hive，SparkSQL也可以运行。需要注意的是，如果你没有部署好Hive，SparkSQL会在当前的工作目录中创建出自己的Hive元数据仓库，叫作**metastore_db。**此外，如果你尝试使用HiveQL中的CREATETABLE(并非CREATEEXTERNALTABLE)语句来创建表，这些表会被放在你默认的文件系统中的/user/hive/warehouse目录中(如果你的classpath中有配好的**hdfs-site.xml**，默认的文件系统就是HDFS，否则就是本地文件系统)。

1.使用内嵌的Hive

> 如果使用Spark内部的Hive则什么不用做直接使用spark.sql()即可。Hive的元数据存储在derby中，仓库地址:$SPARK_HOME/spark-warehouse

```
scala> val sql = spark.sql("show tables")
sql: org.apache.spark.sql.DataFrame = [database: string, tableName: string ... 1 more field]

scala> sql.show
+--------+---------+-----------+
|database|tableName|isTemporary|
+--------+---------+-----------+
+--------+---------+-----------+
```

从上图我们知道调用spark.sql()去执行sql语句，可以返回一个DataFrame，这样就可以通过使用DataFrame中的API对数据进行处理。现来来创建一张表：

```
scala> spark.sql("create table person(name string)")
19/11/07 09:25:52 WARN metastore.HiveMetaStore: Location: file:/root/spark-warehouse/person s
rson
res20: org.apache.spark.sql.DataFrame = []

scala> spark.sql("show tables").show
+--------+---------+-----------+
|database|tableName|isTemporary|
+--------+---------+-----------+
| default|   person|      false|
+--------+---------+-----------+
```

往表中添加数据：

```
scala> spark.sql("load data local inpath '/opt/data/person.txt' into table person")
res2: org.apache.spark.sql.DataFrame = []
```

```
scala> spark.sql("select * from person")
res3: org.apache.spark.sql.DataFrame = [name: string]

scala> spark.sql("select * from person").show
+-----+
| name|
+-----+
| Jack|
|Marry|
|  Tom|
+-----+
```

但是在实际使用中，几乎没有任何人会使用内置的Hive

2.使用外置的Hive

（1）Spark要接管Hive需要把hive-site.xml复制到conf/目录下。

（2）把Mysql的驱动copy到jars/目录下。

（3）如果访问不到hdfs，则需要把core-site.xml和hdfs-site.xml拷贝到conf/目录下。

打开spark-shell，查看student表的数据

```
scala> spark.sql("show tables").show
+--------+---------+-----------+
|database|tableName|isTemporary|
+--------+---------+-----------+
| default|  student|      false|
+--------+---------+-----------+


scala> spark.sql("select * from student").show
+----+---------+
|  id|     name|
+----+---------+
|1001|zhangshan|
|1002|    lishi|
|1003|  zhaoliu|
|1001|zhangshan|
|1002|    lishi|
|1003|  zhaoliu|
+----+---------+
```

DataSet创建

（一）根据样例类创建

1）创建一个样例类

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

2）创建DataSet

```
scala> val caseClassDS = Seq(Person("Andy", 32)).toDS(
caseClassDS: org.apache.spark.sql.Dataset[Person] = [name: string, age: bigint]
```

（二）从一个存在的RDD进行转换

1）创建一个RDD

```
scala> val peopleRDD = sc.textFile("examples/src/main/resources/people.txt")
peopleRDD: org.apache.spark.rdd.RDD[String] =
examples/src/main/resources/people.txt MapPartitionsRDD[3] at textFile at
<console>:27
```

2）创建一个样例类

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

3）将RDD转化为DataSet

```
scala> peopleRDD.map(line => {val para = line.split(",");
Person(para(0),para(1).trim.toInt)}).toDS
res8: org.apache.spark.sql.Dataset[Person] = [name: string, age: bigint]
```

RDD和DataFrame和DataSet之间的相互转换

RDD->DataFrame

```
val personDF = personRDD.toDF("name","sex","age")
```

RDD->DataSet

```
peopleRDD.map(
    line => {val para = line.split(",");
    Person(para(0),para(1).trim.toInt)}).toDS
```

DataFrame->RDD

```
personDF.rdd
```

DataFrame->DataSet

```
personDS = personDF.as[Person]
```

DataSet->RDD

```
personDS.rdd
```

DataSet->DataFrame

```
val personDF = personDS.toDF
```