> 万物皆工具，只不过有的工具比较高级，有的工具处于底层，互相结合使用罢了

SparkSQL自定义UDF和UDAF函数

IDEA创建SparkSQL程序

（1）添加SparkSQL的依赖

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.11</artifactId>
    <version>2.1.1</version>
</dependency>
```

（2）编写SparkSQL代码

```
object SparkSQLDemo {
  def main(args: Array[String]): Unit = {
    // 创建SparkSession
    val spark: SparkSession = SparkSession.builder()
      .master("local[*]")
      .appName("SparkSQLDemo")
      .getOrCreate()

    //导入用到隐式转换

    //从json文件中创建DF
    val df = spark.read.json("file://" +
ClassLoader.getSystemResource("phone.json").getPath)

    // 查找年龄大于19岁的
    // df.filter($"price" >= 4500).show

    // 创建临时表
    df.createTempView("phone")
    spark.sql("select * from phone where price >= 4500").show

    //关闭连接
    spark.stop()

  }

}
```

> 从上面的代码可以得知我们能使用两种方式来处理DataFrame中的数据，一种是调用DataFrame自身的API，一种是通过SparkSQL来处理数据。对于简单的数据处理可以使用DataFrame自带的接口处理，如果是业务处理相对复杂，需要**关联其它数据，那就使用SparkSQL来处理。**

自定义UDF函数

需求：将name这列所有字母变为大写（这个函数很重要，可以用于数据清洗，数据提取）

1.从HDFS中加载数据到DataFrame中

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

scala> studentDF.show
+---+-----+
| id| name|
+---+-----+
|  1|  Tom|
|  2| Jack|
|  3|Marry|
|  4|Amili|
|  5|James|
+---+-----+
```

2.注册UDF函数，函数名为toUpper就是将所有名字变成大写

```
scala> spark.udf.register("toUpper",(s:String) => s.toUpperCase)
res1: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<function1>
,StringType,Some(List(StringType)))
```

3.创建临时视图，然后执行注册的函数

```
scala> studentDF.createOrReplaceTempView("student")

scala> spark.sql("select id,toUpper(name) from student").show
+---+-----------------+
| id|UDF:toUpper(name)|
+---+-----------------+
|  1|              TOM|
|  2|             JACK|
|  3|            MARRY|
|  4|            AMILI|
|  5|            JAMES|
+---+-----------------+
```

自定义聚合函数

> 强类型的Dataset和弱类型的DataFrame都提供了相关的聚合函数，如 count()，countDistinct()，avg()，max()，min()。除此之外，用户可以设定自己的自定义聚合函数。
>
> 通过继承UserDefinedAggregateFunction来实现聚合函数【有问题都可以私聊我WX：focusbigdata，或者关注我的公众号：FocusBigData，注意大小写】

需求：求price这一列的平均值

```
object UDFDemo {
  def main(args: Array[String]): Unit = {
    // 测试自定义的聚合函数
    val spark: SparkSession = SparkSession
      .builder()
      .master("local[*]")
      .appName("UDFDemo")
      .getOrCreate()

    // 注册自定义函数
    spark.udf.register("myAvg",  new MyAvg)

    val df = spark.read.json("file://" +
ClassLoader.getSystemResource("phone.json").getPath)
    df.createTempView("phone")
    spark.sql("select myAvg(price) price_avg from phone").show
```

```
  }
}

class MyAvg extends UserDefinedAggregateFunction {
  /**
    * 返回聚合函数输入参数的数据类型
    *
    * @return
    */
  override def inputSchema: StructType = {
    StructType(StructField("inputColumn", DoubleType) :: Nil)
  }

  /**
    * 聚合缓冲区中值的类型
    *
    * @return
    */
  override def bufferSchema: StructType = {
    StructType(StructField("sum", DoubleType) :: StructField("count", LongType)
:: Nil)
  }

  /**
    * 最终的返回值的类型
    *
    * @return
    */
  override def dataType: DataType = DoubleType

  /**
    * 确定性: 比如同样的输入是否返回同样的输出
    *
    * @return
    */
  override def deterministic: Boolean = true

  /**
    * 初始化
    *
    * @param buffer
    */
  override def initialize(buffer: MutableAggregationBuffer): Unit = {
    // 存数据的总和
    buffer(0) = 0d
    // 储存数据的个数
    buffer(1) = 0L
  }

  /**
    * 相同 Executor间的合并!!!
    *
    * @param buffer
    * @param input
    */
  override def update(buffer: MutableAggregationBuffer, input: Row): Unit = {
    if (!input.isNullAt(0)) {
      buffer(0) = buffer.getDouble(0) + input.getDouble(0)
```

```
      buffer(1) = buffer.getLong(1) + 1
    }
  }
  /**
    * 不同 Executor间的合并!!!
    *
    * @param buffer1
    * @param buffer2
    */
  override def merge(buffer1: MutableAggregationBuffer, buffer2: Row): Unit = {
    if (!buffer2.isNullAt(0)) {
      buffer1(0) = buffer1.getDouble(0) + buffer2.getDouble(0)
      buffer1(1) = buffer1.getLong(1) + buffer2.getLong(1)
    }
  }

  /**
    * 计算最终的结果.  因为是聚合函数,  所以最后只有一行了
    *
    * @param buffer
    * @return
    */
  override def evaluate(buffer: Row): Double = {
    println(buffer.getDouble(0), buffer.getLong(1))

    // 平均=总和/总数
    buffer.getDouble(0) / buffer.getLong(1)
  }
}
```