在 Apache Spark 中使用 UDF

用户定义函数（User-defined functions, UDFs）是大多数 SQL
环境的关键特性，用于扩展系统的内置功能。
UDF允许开发人员通过抽象其低级语言实现来在更高级语言（如SQL）中启用新功能。 Apache Spark 也不例外，并且提供了用于将 UDF 与 Spark SQL工作流集成的各种选项。

在这篇博文中，我们将回顾 Python，Java和 Scala 中的 Apache Spark UDF和UDAF（user-defined aggregate function, 用户定义的聚合函数）实现的简单示例。我们还将讨论重要的 UDF API 功能和集成点。最后，我们将介绍一些在实际应用程序中利用 UDF
需要考虑的重要性能问题。

如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

Spark SQL UDFs

UDF对表中的单行进行转换，以便为每行生成单个对应的输出值。例如，大多数 SQL 环境提供 UPPER 函数返回作为输入提供的字符串的大写版本。

用户自定义函数可以在 Spark SQL 中定义和注册为
UDF，并且可以关联别名，这个别名可以在后面的 SQL
查询中使用。作为一个简单的示例，我们将定义一个 UDF 来将以下 JSON
数据中的温度从摄氏度（degrees Celsius）转换为华氏度（degrees Fahrenheit）：

```
{"city":"St. John's","avgHigh":8.7,"avgLow":0.6}
{"city":"Charlottetown","avgHigh":9.7,"avgLow":0.9}
```

{"city":"Halifax","avgHigh":11.0,"avgLow":1.6}
{"city":"Fredericton","avgHigh":11.2,"avgLow":-0.5}
{"city":"Quebec","avgHigh":9.0,"avgLow":-1.0}
{"city":"Montreal","avgHigh":11.1,"avgLow":1.4}
...

文本文件可以到 这里 下载。

以下示例代码使用 SQL 别名为 CTOF 来注册我们的转换 UDF，然后在 SQL 查询使用它来转换每个城市的温度。为简洁起见，省略了 SQLContext 对象和其他代码的创建，每段代码下面都提供了完整的代码链接。

Python

```
df = sqlContext.read.json("temperatures.json")

df.registerTempTable("citytemps")
# Register the UDF with our SQLContext

sqlContext.registerFunction("CTOF", lambda degreesCelsius: ((degreesCelsius * 9.0 / 5.0) + 32.0))
sqlContext.sql("SELECT city, CTOF(avgLow) AS avgLowF, CTOF(avgHigh) AS avgHighF FROM citytemps").show()
```

完整的代码片段看这里

Scala

```
val df = sqlContext.read.json("temperatures.json")

df.registerTempTable("citytemps")
// Register the UDF with our SQLContext

sqlContext.udf.register("CTOF", (degreesCelcius: Double) => ((degreesCelcius * 9.0 / 5.0) + 32.0))
sqlContext.sql("SELECT city, CTOF(avgLow) AS avgLowF, CTOF(avgHigh) AS avgHighF FROM citytemps").show()
```

完整的代码片段看这里

Java

```
DataFrame df = sqlContext.read().json("temperatures.json");
df.registerTempTable("citytemps");

// Register the UDF with our SQLContext
sqlContext.udf().register("CTOF", new UDF1<Double, Double>() {
  @Override
  public Double call(Double degreesCelcius) {
    return ((degreesCelcius * 9.0 / 5.0) + 32.0);
  }
}, DataTypes.DoubleType);
sqlContext.sql("SELECT city, CTOF(avgLow) AS avgLowF, CTOF(avgHigh) AS avgHighF FROM cityt
emps").show();
```

完整的代码片段看这里

注意，Spark SQL定义了 UDF1 到 UDF22 共22个类，UDF
最多支持22个输入参数。上面的例子中使用 UDF1
来处理我们单个温度值作为输入。如果我们不想修改 Apache Spark 的源代码，对于需要超过22个输出参数的应用程序我们可以使用数组或结构作为参数来解决这个问题，如果你发现自己用了UDF6或者更高UDF类你可以考虑这样操作。

Spark SQL UDAF functions

用户定义的聚合函数（User-defined aggregate functions,
UDAF）同时处理多行，并且返回一个结果，通常结合使用 GROUP BY 语句（例如 COUNT 或 SUM）。为了简单起见，我们将实现一个叫 SUMPRODUCT 的 UDAF
来计算以库存来分组的所有车辆零售价值，具体的数据如下：

```
{"Make":"Honda","Model":"Pilot","RetailValue":32145.0,"Stock":4}
{"Make":"Honda","Model":"Civic","RetailValue":19575.0,"Stock":11}
{"Make":"Honda","Model":"Ridgeline","RetailValue":42870.0,"Stock":2}
{"Make":"Jeep","Model":"Cherokee","RetailValue":23595.0,"Stock":13}
{"Make":"Jeep","Model":"Wrangler","RetailValue":27895.0,"Stock":4}
{"Make":"Volkswagen","Model":"Passat","RetailValue":22440.0,"Stock":2}
```

文本文件可以到 这里 下载。
Apache Spark UDAF 目前只支持在 Scala 和 Java 中通过扩展 UserDefinedAggregateFunction 类使用。下面例子中我们定义了一个名为 SumProductAggregateFunction 的类，并且为它取了一个名为 SUMPRODUCT 的别名，现在我们可以在 SQL 查询中初始化并注册它，和上面的 CTOF UDF 的操作步骤很类似，如下：

```
object ScalaUDAFExample {

 // Define the SparkSQL UDAF logic
 private class SumProductAggregateFunction extends UserDefinedAggregateFunction {
  // Define the UDAF input and result schema's
  def inputSchema: StructType =     // Input  = (Double price, Long quantity)
   new StructType().add("price", DoubleType).add("quantity", LongType)
  def bufferSchema: StructType =    // Output = (Double total)
   new StructType().add("total", DoubleType)
  def dataType: DataType = DoubleType
  def deterministic: Boolean = true // true: our UDAF's output given an input is deterministic

  def initialize(buffer: MutableAggregationBuffer): Unit = {
   buffer.update(0, 0.0)          // Initialize the result to 0.0
  }

  def update(buffer: MutableAggregationBuffer, input: Row): Unit = {
   val sum   = buffer.getDouble(0) // Intermediate result to be updated
   val price = input.getDouble(0)  // First input parameter
   val qty   = input.getLong(1)    // Second input parameter
   buffer.update(0, sum + (price * qty))   // Update the intermediate result
  }
  // Merge intermediate result sums by adding them
  def merge(buffer1: MutableAggregationBuffer, buffer2: Row): Unit = {
   buffer1.update(0, buffer1.getDouble(0) + buffer2.getDouble(0))
  }
  // THe final result will be contained in 'buffer'
  def evaluate(buffer: Row): Any = {
   buffer.getDouble(0)
  }
 }

 def main (args: Array[String]) {
  val conf       = new SparkConf().setAppName("Scala UDAF Example")
  val sc         = new SparkContext(conf)
  val sqlContext = new SQLContext(sc)

  val testDF = sqlContext.read.json("inventory.json")
  testDF.registerTempTable("inventory")
  // Register the UDAF with our SQLContext
```

```
    sqlContext.udf.register("SUMPRODUCT", new SumProductAggregateFunction)

    sqlContext.sql("SELECT Make, SUMPRODUCT(RetailValue,Stock) AS InventoryValuePerMake F
ROM inventory GROUP BY Make").show()
  }
}
```

完整的代码片段看这里

Apache Spark中的其他UDF支持

Spark SQL 支持集成现有 Hive 中的 UDF，UDAF 和 UDTF 的（Java或Scala）实现。UDTFs（user-defined table functions, 用户定义的表函数）可以返回多列和多行 - 它们超出了本文的讨论范围，我们可能会在以后进行说明。集成现有的 Hive UDF是非常有意义的，我们不需要向上面一样重新实现和注册他们。Hive 定义好的函数可以通过 HiveContext 来使用，不过我们需要通过 spark-submit 的 –jars 选项来指定包含 HIVE UDF 实现的 jar 包，然后通过 CREATE TEMPORARY FUNCTION 语句来定义函数，如下：

Hive UDF definition in Java

```
package com.cloudera.fce.curtis.sparkudfexamples.hiveudf;

import org.apache.hadoop.hive.ql.exec.UDF;

public class CTOF extends UDF {
  public Double evaluate(Double degreesCelsius) {
    return ((degreesCelsius * 9.0 / 5.0) + 32.0);
  }
}
```

完整的代码片段看这里

在 Python 中使用 Hive UDF

```
df = sqlContext.read.json("temperatures.json")
df.registerTempTable("citytemps")

# Register our Hive UDF
sqlContext.sql("CREATE TEMPORARY FUNCTION CTOF AS 'com.cloudera.fce.curtis.sparkudfexa
```

```
mples.hiveudf.CTOF'")

sqlContext.sql("SELECT city, CTOF(avgLow) AS avgLowF, CTOF(avgHigh) AS avgHighF FROM cityt
emps").show()
```

完整的代码片段看这里

注意，Hive UDF只能使用 Apache Spark 的 SQL 查询语言来调用 - 换句话说，它们不能与 Dataframe API的领域特定语言（domain-specific-language, DSL）一起使用。

另外，通过包含实现 jar 文件（在 spark-submit 中使用 -jars 选项）的方式 PySpark 可以调用 Scala 或 Java 编写的 UDF（through the SparkContext object's private reference to the executor JVM and underlying Scala or Java UDF implementations that are loaded from the jar file）。下面的示例演示了如何使用先前 Scala 中定义的 SUMPRODUCT UDAF：

Scala UDAF definition

```
object ScalaUDAFFromPythonExample {
  // ... UDAF as defined in our example earlier ...
}

  // This function is called from PySpark to register our UDAF
  def registerUdf(sqlCtx: SQLContext) {
    sqlCtx.udf.register("SUMPRODUCT", new SumProductAggregateFunction)
  }
}
```

完整的代码片段看这里

Scala UDAF from PySpark

```
df = sqlContext.read.json("inventory.json")
df.registerTempTable("inventory")

scala_sql_context  =  sqlContext._ssql_ctx
scala_spark_context = sqlContext._sc
scala_spark_context._jvm.com.cloudera.fce.curtis.sparkudfexamples.scalaudaffrompython.Scal
aUDAFFromPythonExample.registerUdf(scala_sql_context)
```

```
sqlContext.sql("SELECT Make, SUMPRODUCT(RetailValue,Stock) AS InventoryValuePerMake FROM inventory GROUP BY Make").show()
```

完整的代码片段看这里

每个版本的 Apache Spark 都在不断地添加与 UDF 相关的功能，比如在 2.0 中 R 增加了对 UDF 的支持。作为参考，下面的表格总结了本博客中讨论特性版本：

Apache Spark Version	Spark SQL UDF (Python, Java, Scala)	Spark SQL UDAF (Java, Scala)	Spark SQL UDF (R)	Hive UDF, UDAF, UDTF
1.1-1.4	✓			✓
1.5	✓	experimental		✓
1.6	✓	✓		✓
2.0	✓	✓	✓	✓

如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

性能相关

了解 Apache Spark UDF 功能的性能影响很重要。例如，Python UDF（比如上面的 CTOF 函数）会导致数据在执行器的 JVM 和运行 UDF 逻辑的 Python 解释器之间进行序列化操作；与 Java 或 Scala 中的 UDF 实现相比，大大降低了性能。缓解这种序列化瓶颈的解决方案如下：

1、从 PySpark 访问 Hive UDF。 Java UDF实现可以由执行器JVM直接访问。
2、在 PySpark 中访问在 Java 或 Scala 中实现的 UDF 的方法。正如上面的 Scala UDAF 实例。

本文翻译自：Working with UDFs in Apache Spark