pyspark.sql module

Module Context

Important classes of Spark SQL and DataFrames:

[pyspark.sql.SparkSession](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.SparkSession) :DataFrame和SQL功能的主要入口点。

[pyspark.sql.DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) :将分组的数据集合分组到命名列中。

[pyspark.sql.Column](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.Column) :DataFrame中的列表达式。

[pyspark.sql.Row](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.Row) :DataFrame中的一行数据。

[pyspark.sql.GroupedData](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.GroupedData):聚合方法，返回 [DataFrame.groupBy()](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame.groupBy).

[pyspark.sql.DataFrameNaFunctions](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrameNaFunctions):处理丢失数据的方法（空值）。

[pyspark.sql.DataFrameStatFunctions](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrameStatFunctions): 统计功能的方法。

[pyspark.sql.functions](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#module-pyspark.sql.functions) :DataFrame可用的内置函数列表。

[pyspark.sql.types](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#module-pyspark.sql.types) :可用数据类型列表。

[pyspark.sql.Window](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.Window) : 使用窗口功能。

# **SparkSession**

**class pyspark.sql.SparkSession(sparkContext, jsparkSession=None)**

使用Dataset和DataFrame API编程Spark的切入点。

可以使用SparkSession创建DataFrame，将DataFrame注册为表，在表上执行SQL，缓存表和读取拼贴文件。 要创建SparkSession，请使用以下构建器模式：

**>>> spark = SparkSession.builder \**

**... .master("local") \**

**... .appName("Word Count") \**

**... .config("spark.some.config.option", "some-value") \**

**... .getOrCreate()**

class Builder

Builder for [SparkSession](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.SparkSession).

appName(name)

设置应用程序的名称，这将在Spark Web UI中显示。

**如果没有设置应用程序名称，将使用随机生成的名称。**

|  |  |
| --- | --- |
| Parameters: | name – 应用名称 |

New in version 2.0.

config(key=None, value=None, conf=None)

设置一个配置选项。 使用此**方法设置的选项将自动传播到SparkConf和SparkSession自己的配置。**

对于现有的SparkConf，请使用conf参数。

>>> from pyspark.conf import SparkConf

>>> SparkSession.builder.config(conf=SparkConf())

<pyspark.sql.session…

对于（键，值）对，可以省略参数名称。

>>> SparkSession.builder.config("spark.some.config.option", "some-value")

<pyspark.sql.session...

|  |  |
| --- | --- |
| Parameters: | key – 配置属性的键名字符串  value – 配置属性的值  conf – SparkConf的一个实例 |

New in version 2.0.

**enableHiveSupport()**

启用Hive支持，包括连接到持续的Hive转移，支持Hive serdes和Hive用户定义的功能。

2.0版新功能

**getOrCreate() （推荐使用）**

**获取现有的SparkSession，如果没有现有的SparkSession，则根据此构建器中设置的选项创建一个新的SparkSession。**

此方法首先检查是否存在有效的全局默认SparkSession，如果是，则返回一个。 如果没有有效的全局默认SparkSession存在，该方法将创建一个新的SparkSession并将新创建的SparkSession分配为全局默认值。

>>> s1 = SparkSession.builder.config("k1", "v1").getOrCreate()

>>> s1.conf.get("k1") == s1.sparkContext.getConf().get("k1") == "v1"

True

如果返回现有的SparkSession，则此构建器中指定的配置选项将应用于现有的SparkSession。

>>> s2 = SparkSession.builder.config("k2", "v2").getOrCreate()

>>> s1.conf.get("k1") == s2.conf.get("k1")

True

>>> s1.conf.get("k2") == s2.conf.get("k2")

True

New in version 2.0.

**master(master)**

设置要连接的Spark主URL，例如“本地”在本地运行，“local [4]”以4个核心或“spark：// master：7077”本地运行以在Spark独立群集上运行。

|  |  |
| --- | --- |
| Parameters: | master – a url for spark master |

New in version 2.0.

**SparkSession.builder**

= <pyspark.sql.session.Builder object at 0x7fdfde29fbd0>

**SparkSession.catalog**

用户可以创建，删除，更改或查询底层数据库，表格，功能等的界面。

2.0版新功能

**SparkSession.conf**

Spark的运行时配置界面。

这是用户可以通过该接口获取和设置与Spark SQL相关的所有Spark和Hadoop配置的界面。 获取配置值时，这将默认为底层SparkContext中设置的值（如果有）。

2.0版新功能

**SparkSession.createDataFrame(data, schema=None, samplingRatio=None, verifySchema=True)**

**从RDD，列表或pandas.DataFrame创建DataFrame。当模式是列名列表时，每个列的类型将从数据中推断出来。**

当schema为None时，它将尝试从数据中推断出模式（列名和类型），它们应该是Row的RDD，或者是namedtuple或dict。

当schema是pyspark.sql.types.DataType或数据类型字符串时，它必须与实际数据匹配，否则将在运行时抛出异常。 如果给定的模式不是pyspark.sql.types.StructType，它将被包装到一个pyspark.sql.types.StructType作为其唯一的字段，并且字段名称将是“值”，每个记录也将被包装成 元组，可以稍后转换成行。

如果需要模式推理，则使用samplingRatio来确定用于模式推理的行的比率。 如果samplingRatio为None，则会使用第一行。

|  |  |
| --- | --- |
| Parameters: | data – 任何类型的SQL数据表示（例如row，tuple，int，boolean等）的RDD或者pandas.DataFrame。  schema – pyspark.sql.types.DataType或数据类型字符串或列名称列表，默认值为无。 数据类型字符串格式等于pyspark.sql.types.DataType.simpleString，除了顶级结构类型可以省略struct <>和原子类型使用typeName（）作为其格式，例如。 对于pyspark.sql.types.ByteType使用字节而不是tinyint。 我们也可以使用int作为IntegerType的简称。  samplingRatio – 用于推测的行的采样比率  verifySchema – 根据模式验证每行的数据类型。 |
| Returns: | [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) |

在版本2.1中更改：添加了verifySchema。

>>> l = [('Alice', 1)]

>>> spark.createDataFrame(l).collect()

[Row(\_1=u'Alice', \_2=1)]

>>> spark.createDataFrame(l, ['name', 'age']).collect()

[Row(name=u'Alice', age=1)]

>>> d = [{'name': 'Alice', 'age': 1}]

>>> spark.createDataFrame(d).collect()

[Row(age=1, name=u'Alice')]

>>> rdd = sc.parallelize(l)

>>> spark.createDataFrame(rdd).collect()

[Row(\_1=u'Alice', \_2=1)]

>>> df = spark.createDataFrame(rdd, ['name', 'age'])

>>> df.collect()

[Row(name=u'Alice', age=1)]

>>> **from pyspark.sql import Row**

>>> Person = Row('name', 'age')

>>> **person = rdd.map(lambda r: Person(\*r))**

>>> **df2 = spark.createDataFrame(person)**

**>>> df2.collect()**

[Row(name=u'Alice', age=1)]

>>> **from pyspark.sql.types import \***

**>>> schema = StructType([**

**... StructField("name", StringType(), True),**

**... StructField("age", IntegerType(), True)])**

**>>> df3 = spark.createDataFrame(rdd, schema)**

**>>> df3.collect()**

[Row(name=u'Alice', age=1)]

>>> spark.createDataFrame(df.toPandas()).collect()

[Row(name=u'Alice', age=1)]

>>> spark.createDataFrame(pandas.DataFrame([[1, 2]])).collect()

[Row(0=1, 1=2)]

>>> spark.createDataFrame(rdd, "a: string, b: int").collect()

[Row(a=u'Alice', b=1)]

>>> rdd = rdd.map(lambda row: row[1])

>>> spark.createDataFrame(rdd, "int").collect()

[Row(value=1)]

>>> spark.createDataFrame(rdd, "boolean").collect()

Traceback (most recent call last):

...

Py4JJavaError: ...

New in version 2.0.

**SparkSession.newSession()**

返回一个新的SparkSession作为新会话，它具有单独的SQLConf，注册的临时视图和UDF，但共享SparkContext和表缓存。

2.0版新功能

**SparkSession.range(start, end=None, step=1, numPartitions=None)**

使用单个pyspark.sql.types.LongType列命名为id创建一个DataFrame，其中包含从起始到结束（独占）的范围内的元素，具有步长值步长。

|  |  |
| --- | --- |
| Parameters: | start – 起始值  end – 结束值（独占）  step – 增量步骤（默认值：1）  numPartitions – DataFrame的分区数 |
| Returns: | [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) |

>>> spark.range(1, 7, 2).collect()

[Row(id=1), Row(id=3), Row(id=5)]

如果只有一个参数被指定，它将被用作结束值。

>>> spark.range(3).collect()

[Row(id=0), Row(id=1), Row(id=2)]

New in version 2.0.

**SparkSession.read**

返回一个DataFrameReader，可用于以DataFrame的形式读取数据。

返回：

DataFrameReader

2.0版新功能

**SparkSession.readStream**

返回一个DataStreamReader，可以用来读取数据流作为流数据帧。

注意

实验。

返回：DataStreamReader

2.0版新功能

**SparkSession.sparkContext**

返回底层的SparkContext。

2.0版新功能

**SparkSession.sql(sqlQuery)**

返回表示给定查询结果的DataFrame。

|  |  |
| --- | --- |
| Returns: | [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) |

>>> df.**createOrReplaceTempView**("table1")

**>>> df2 = spark.sql("SELECT field1 AS f1, field2 as f2 from table1")**

**>>> df2.collect()**

[Row(f1=1, f2=u'row1'), Row(f1=2, f2=u'row2'), Row(f1=3, f2=u'row3')]

New in version 2.0.

**SparkSession.stop()**

**停止底层SparkContext。**

2.0版新功能

**SparkSession.streams**

**返回一个StreamingQueryManager，允许管理在此上下文中活动的所有StreamingQuery StreamingQueries。**

注意

实验。

返回：

StreamingQueryManager

2.0版新功能

**SparkSession.table(tableName)**

作为DataFrame返回指定的表。

|  |  |
| --- | --- |
| Returns: | [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) |

>>> df.createOrReplaceTempView("table1")

>>> df2 = spark.table("table1")

>>> sorted(df.collect()) == sorted(df2.collect())

True

New in version 2.0.

**SparkSession.udf**

返回用于UDF注册的UDF注册。

返回：UDFRegistration

2.0版新功能

SparkSession.version

此应用程序正在运行的Spark版本。

2.0版新功能

# **SQLContext**

**class pyspark.sql.SQLContext**(**sparkContext**, **sparkSession=None**, **jsqlContext=None**)

.在Spark 1.x中使用Spark中的结构化数据（行和列）的入口点。

**从Spark 2.0开始，这被SparkSession所取代。 但是，我们正在为此向后兼容。**

可以使用SQLContext创建DataFrame，将DataFrame注册为表，在表上执行SQL，缓存表和读取拼贴文件

|  |  |
| --- | --- |
| Parameters: | sparkContext – SparkContext支持此SQLContext。  sparkSession – 这个SQLContext所围绕的SparkSession。  jsqlContext –可选的JVM Scala SQLContext。 如果设置，我们不会在JVM中实例化一个新的SQLContext，而是对该对象进行所有调用。 |

**cacheTable(tableName)**

将指定的表缓存在内存中。

新版本1.0。

**clearCache()**

**从内存中缓存中删除所有缓存的表。**

1.3版新功能

**createDataFrame(data, schema=None, samplingRatio=None, verifySchema=True)**

**从RDD，列表或pandas.DataFrame创建DataFrame。**

当模式是列名列表时，每个列的类型将从数据中推断出来。

当schema为None时，它将尝试从数据中推断出模式（列名和类型），它们应该是Row的RDD，或者是namedtuple或dict。

**当模式是pyspark.sql.types.DataType或数据类型字符串时，它必须与实际数据匹配，否则将在运行时抛出异常**。 如果给定的模式不是pyspark.sql.types.StructType，它将被包装到一个pyspark.sql.types.StructType作为其唯一的字段，并且字段名称将是“值”，每个记录也将被包装成 元组，可以稍后转换成行。

如果需要模式推理，则使用samplingRatio来确定用于模式推理的行的比率。 如果samplingRatio为None，则会使用第一行。

|  |  |
| --- | --- |
| Parameters: | data – 任何类型的SQL数据表示（例如Row，tuple，int，boolean等）或列表或pandas.DataFrame的RDD。  schema – pyspark.sql.types.DataType或数据类型字符串或列名称列表，默认值为无。 数据类型字符串格式等于pyspark.sql.types.DataType.simpleString，除了顶级结构类型可以省略struct <>和原子类型使用typeName（）作为其格式，例如。 对于pyspark.sql.types.ByteType使用字节而不是tinyint。 我们也可以使用int作为pyspark.sql.types.IntegerType的简称。  samplingRatio – 用于推测的行的采样比率  verifySchema – 根据模式验证每行的数据类型。 |
| Returns: | [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) |

版本2.0中更改：**schema参数可以是pyspark.sql.types.DataType**或2.0之后的数据类型字符串。 如果它不是一**个pyspark.sql.types.StructType**，它将被包装到一个pyspark.sql.types.StructType中，每个记录也将被包装成一个元组。

在版本2.1中更改：添加了**verifySchema**。

>>> l = [('Alice', 1)]

>>> sqlContext.createDataFrame(l).collect()

[Row(\_1=u'Alice', \_2=1)]

>>> sqlContext.createDataFrame(l, ['name', 'age']).collect()

[Row(name=u'Alice', age=1)]

>>> d = [{'name': 'Alice', 'age': 1}]

>>> sqlContext.createDataFrame(d).collect()

[Row(age=1, name=u'Alice')]

>>> rdd = sc.parallelize(l)

>>> sqlContext.createDataFrame(rdd).collect()

[Row(\_1=u'Alice', \_2=1)]

>>> df = sqlContext.createDataFrame(rdd, ['name', 'age'])

>>> df.collect()

[Row(name=u'Alice', age=1)]

>>> from pyspark.sql import Row

>>> Person = Row('name', 'age')

>>> person = rdd.map(lambda r: Person(\*r))

>>> df2 = sqlContext.createDataFrame(person)

>>> df2.collect()

[Row(name=u'Alice', age=1)]

>>> from pyspark.sql.types import \*

>>> schema = StructType([

... StructField("name", StringType(), True),

... StructField("age", IntegerType(), True)])

>>> df3 = sqlContext.createDataFrame(rdd, schema)

>>> df3.collect()

[Row(name=u'Alice', age=1)]

>>> sqlContext.createDataFrame(df.toPandas()).collect()

[Row(name=u'Alice', age=1)]

>>> sqlContext.createDataFrame(pandas.DataFrame([[1, 2]])).collect()

[Row(0=1, 1=2)]

>>> sqlContext.createDataFrame(rdd, "a: string, b: int").collect()

[Row(a=u'Alice', b=1)]

>>> rdd = rdd.map(lambda row: row[1])

>>> sqlContext.createDataFrame(rdd, "int").collect()

[Row(value=1)]

>>> sqlContext.createDataFrame(rdd, "boolean").collect()

Traceback (most recent call last):

...

Py4JJavaError: ...

New in version 1.3.

**createExternalTable(tableName, path=None, source=None, schema=None, \*\*options)**

根据数据源中的数据集创建一个外部表。

它返回与外部表关联的DataFrame。

数据源由源和一组选项指定。 如果未指定源，则将使用由spark.sql.sources.default配置的默认数据源。

可选地，可以将模式提供为返回的DataFrame和创建的外部表的模式。

返回：

DataFrame

1.3版新功能

**dropTempTable(tableName)**

从目录中删除临时表。

>>> sqlContext.registerDataFrameAsTable(df, "table1")

>>> sqlContext.dropTempTable("table1")

New in version 1.6.

**getConf(key, defaultValue=None)** **：返回给定键的Spark SQL配置属性的值。**

如果未设置键，并且defaultValue不为None，则返回defaultValue。 如果未设置密钥，并且defaultValue为None，则返回系统默认值。

>>> sqlContext.getConf("spark.sql.shuffle.partitions")

u'200'

>>> sqlContext.getConf("spark.sql.shuffle.partitions", u"10")

u'10'

>>> sqlContext.setConf("spark.sql.shuffle.partitions", u"50")

>>> sqlContext.getConf("spark.sql.shuffle.partitions", u"10")

u'50'

New in version 1.3.

**classmethod getOrCreate(sc)**

**获取现有的SQLContext或使用给定的SparkContext创建一个新的。**

参数：

sc - SparkContext

新版本1.6。

**newSession()**

返回一个新的SQLContext作为新会话，具有单独的SQLConf，注册的临时视图和UDF，但共享SparkContext和表缓存。

新版本1.6。

**range(start, end=None, step=1, numPartitions=None)**

使用单个pyspark.sql.types.LongType列命名为id创建一个DataFrame，其中包含从起始到结束（独占）的范围内的元素，具有步长值步长。

|  |  |
| --- | --- |
| Parameters: | start – the start value  end – the end value (exclusive)  step – the incremental step (default: 1)  numPartitions – the number of partitions of the DataFrame |
| Returns: | [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) |

>>> sqlContext.range(1, 7, 2).collect()

[Row(id=1), Row(id=3), Row(id=5)]

**如果只有一个参数被指定，它将被用作结束值。**

>>> sqlContext.range(3).collect()

[Row(id=0), Row(id=1), Row(id=2)]

New in version 1.4.

**read**

**返回一个DataFrameReader，可用于以DataFrame的形式读取数据。**

返回：

DataFrameReader

版本1.4中的新功能。

**readStream**

**返回一个DataStreamReader，可以用来读取数据流作为流数据帧。**

注意

实验。

|  |  |
| --- | --- |
| Returns: | DataStreamReader |

**>>> text\_sdf = sqlContext.readStream.text(tempfile.mkdtemp())**

>>> text\_sdf.isStreaming

True

New in version 2.0.

**registerDataFrameAsTable(df, tableName)**

**将给定的数据帧注册为目录中的临时表。**

**临时表只存在于此SQLContext实例的生命周期内。**

>>> sqlContext.registerDataFrameAsTable(df, "table1")

New in version 1.3.

**registerFunction(name, f, returnType=StringType) （自定义函数作为SQL语句使用）**

**注册一个python函数（包括lambda函数）作为UDF，因此可以在SQL语句中使用。**

除了名称和函数本身之外，还可以指定返回类型。 当没有给出返回类型时，默认为字符串，转换将自动完成。 对于任何其他返回类型，生成的对象必须与指定的类型匹配。

|  |  |
| --- | --- |
| Parameters: | name – name of the UDF  f – python function  returnType – a [pyspark.sql.types.DataType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.DataType) object |

>>> sqlContext.registerFunction("stringLengthString", lambda x: len(x))

>>> sqlContext.sql("SELECT stringLengthString('test')").collect()

[Row(stringLengthString(test)=u'4')]

>>> from pyspark.sql.types import IntegerType

>>> sqlContext.registerFunction("stringLengthInt", lambda x: len(x), IntegerType())

>>> sqlContext.sql("SELECT stringLengthInt('test')").collect()

[Row(stringLengthInt(test)=4)]

>>> from pyspark.sql.types import IntegerType

>>> sqlContext.udf.register("stringLengthInt", lambda x: len(x), IntegerType())

>>> sqlContext.sql("SELECT stringLengthInt('test')").collect()

[Row(stringLengthInt(test)=4)]

New in version 1.2.

registerJavaFunction(name, javaClassName, returnType=None)

**注册一个java UDF，这样就可以在SQL语句中使用。**

除了名称和函数本身之外，还可以指定返回类型。 当没有指定返回类型时，我们将通过反射推断它。 ：param name：UDF的名称：param javaClassName：java类的完全限定名：param returnType：一个pyspark.sql.types.DataType对象

>>> sqlContext.registerJavaFunction("javaStringLength",

... "test.org.apache.spark.sql.JavaStringLength", IntegerType())

>>> sqlContext.sql("SELECT javaStringLength('test')").collect()

[Row(UDF(test)=4)]

>>> sqlContext.registerJavaFunction("javaStringLength2",

... "test.org.apache.spark.sql.JavaStringLength")

>>> sqlContext.sql("SELECT javaStringLength2('test')").collect()

[Row(UDF(test)=4)]

New in version 2.1.

setConf(key, value)

设置给定的Spark SQL配置属性。

1.3版新功能

**sql(sqlQuery)**

返回表示给定查询结果的DataFrame。

|  |  |
| --- | --- |
| Returns: | [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) |

>>> s**qlContext.registerDataFrameAsTable**(df, "table1")

>>> df2 = sqlContext.sql("SELECT field1 AS f1, field2 as f2 from table1")

>>> df2.collect()

[Row(f1=1, f2=u'row1'), Row(f1=2, f2=u'row2'), Row(f1=3, f2=u'row3')]

New in version 1.0.

**streams**

返回一个**StreamingQueryManager**，允许管理在此上下文中活动的所有StreamingQuery StreamingQueries。

注意

实验。

2.0版新功能

**table(tableName)**

作为DataFrame返回指定的表。

|  |  |
| --- | --- |
| Returns: | [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) |

>>> sqlContext.registerDataFrameAsTable(df, "table1")

>>> df2 = sqlContext.table("table1")

>>> sorted(df.collect()) == sorted(df2.collect())

True

New in version 1.0.

**tableNames(dbName=None)**

返回数据库dbName中表的名称列表。

|  |  |
| --- | --- |
| Parameters: | dbName – 字符串，要使用的数据库的名称。 默认为当前数据库。 |
| Returns: | 表名列表，字符串 |

>>> sqlContext.registerDataFrameAsTable(df, "table1")

>>> "table1" in sqlContext.tableNames()

True

>>> "table1" in sqlContext.tableNames("default")

True

New in version 1.3.

**tables(dbName=None)**

返回包含给定数据库中表的名称的DataFrame。

如果未指定dbName，将使用当前数据库。

返回的DataFrame有两列：tableName和isTemporary（一个BooleanType的列，表示表是否为临时表）。

|  |  |
| --- | --- |
| Parameters: | dbName – string, name of the database to use. |
| Returns: | [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) |

>>> sqlContext.**registerDataFrameAsTable**(df, "table1")

>>> df2 = sqlContext.tables()

>>> df2.filter("tableName = 'table1'").first()

Row(database=u'', tableName=u'table1', isTemporary=True)

New in version 1.3.

**udf**

**返回用于UDF注册的UDF注册。**

返回：

**UDFRegistration**

1.3.1版新功能。

**uncacheTable(tableName)**

从内存中缓存中删除指定的表。

新版本1.0。

# **HiveContext**

**class pyspark.sql.HiveContext(sparkContext, jhiveContext=None)**

Spark SQL的一个变体，与存储在Hive中的数据集成在一起。

Hive的配置从类路径中的hive-site.xml中读取。 它支持运行SQL和HiveQL命令。

|  |  |
| --- | --- |
| Parameters: | sparkContext –.要包装的SparkContext  jhiveContext – 可选的JVM Scala HiveContext。 如果设置，我们不会在JVM中实例化一个新的HiveContext，而是对该对象进行所有调用。 |

Note

在2.0.0中弃用。 使用SparkSession.builder.**enableHiveSupport**（）.getOrCreate（）。

**refreshTable(tableName)**

无效并刷新给定表的所有缓存元数据。 **出于性能原因，Spark SQL或其使用的外部数据源库可能会缓存有关表的某些元数据，例如块的位置。** 当这些更改在Spark SQL之外时，用户应该调用此函数来使缓存无效。

# UDFRegistration

**class pyspark.sql.UDFRegistration(sqlContext)**

用户自定义功能注册包装器。

**register（name，f，returnType = StringType）**

注册一个python函数（包括lambda函数）作为UDF，因此可以在SQL语句中使用。

除了名称和函数本身之外，还可以指定返回类型。 当没有给出返回类型时，默认为字符串，转换将自动完成。 对于任何其他返回类型，生成的对象必须与指定的类型匹配。

|  |  |
| --- | --- |
| Parameters: | name – UDF的名称  f – python函数  returnType –一个pyspark.sql.types.DataType对象 |

>>> sqlContext.registerFunction("stringLengthString", lambda x: len(x))

>>> sqlContext.sql("SELECT stringLengthString('test')").collect()

[Row(stringLengthString(test)=u'4')]

>>> from pyspark.sql.types import IntegerType

>>> sqlContext.registerFunction("stringLengthInt", lambda x: len(x), IntegerType())

>>> sqlContext.sql("SELECT stringLengthInt('test')").collect()

[Row(stringLengthInt(test)=4)]

>>> from pyspark.sql.types import IntegerType

>>> sqlContext.udf.register("stringLengthInt", lambda x: len(x), IntegerType())

>>> sqlContext.sql("SELECT stringLengthInt('test')").collect()

[Row(stringLengthInt(test)=4)]

New in version 1.2.

# DataFrame

**class pyspark.sql.DataFrame(jdf, sql\_ctx)**

将分组的数据集合分组到命名列中。

DataFrame等价于Spark SQL中的关系表，可以使用SQLContext中的各种函数创建：

**people = sqlContext.read.parquet("...")**

创建后，可以使用以下定义的各种特定于语言（DSL）的功能进行操作：DataFrame，Column。

To select a column from the data frame, use the apply method:

ageCol = people.age

A more concrete example:

# To create DataFrame using SQLContext

people = sqlContext.read.parquet("...")

department = sqlContext.read.parquet("...")

**people.filter(people.age > 30).join(department, people.deptId == department.id) \**

**.groupBy(department.name, "gender").agg({"salary": "avg", "age": "max"})**

New in version 1.3.

**agg(\*exprs)**

Aggregate on the entire [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) without groups (shorthand for df.groupBy.agg()).

>>> df.agg({"age": "max"}).collect()

[Row(max(age)=5)]

>>> from pyspark.sql import functions as F

>>> df.agg(F.min(df.age)).collect()

[Row(min(age)=2)]

New in version 1.3.

**alias(alias)**

Returns a new [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) with an alias set.

>>> from pyspark.sql.functions import \*

>>> df\_as1 = df.alias("df\_as1")

>>> df\_as2 = df.alias("df\_as2")

>>> joined\_df = df\_as1.join(df\_as2, col("df\_as1.name") == col("df\_as2.name"), 'inner')

>>> joined\_df.select("df\_as1.name", "df\_as2.name", "df\_as2.age").collect()

[Row(name=u'Bob', name=u'Bob', age=5), Row(name=u'Alice', name=u'Alice', age=2)]

New in version 1.3.

**approxQuantile(col, probabilities, relativeError)**

计算DataFrame的数值列的近似分位数。

该算法的结果具有以下确定性界限：如果DataFrame具有N个元素，并且如果我们以概率p请求分数位错误err，则该算法将从DataFrame返回一个样本x，使得x的确切等级为 接近（p \* N）。 更确切地说，

（（p - err）\* N）<= rank（x）<= ceil（（p + err）\* N）。

该方法实现了Greenwald-Khanna算法的变体（具有一些速度优化）。 该算法首先出现在Greenwald和Khanna的[[http://dx.doi.org/10.1145/375663.375670空间效率在线计算分位数概要]中）。

|  |  |
| --- | --- |
| Parameters: | col – 数字列的名称  probabilities – 分位数概率列表每个数字必须属于[0,1]。 例如0是最小值，0.5是中位数，1是最大值。  relativeError – 相对目标精度达到（> = 0）。 如果设置为零，则计算确切的分位数，这可能非常昂贵。 请注意，大于1的值被接受，但结果与1相同。 |
| Returns: | 给定概率的近似分位数 |

New in version 2.0.

**cache()**

使用默认存储级别（MEMORY\_AND\_DISK）保留DataFrame。

注意

默认存储级别已更改为MEMORY\_AND\_DISK以匹配2.0中的Scala。

1.3版新功能

**checkpoint(eager=True)**

返回此数据集的检查点版本。 Checkpointing可用于截断此DataFrame的逻辑计划，这在计划可能呈指数增长的迭代算法中特别有用。 它将被保存到使用SparkContext.setCheckpointDir（）设置的检查点目录内的文件中。

参数：

渴望 - 是否立即检查该DataFrame

注意

试验

2.1版新功能

**coalesce(numPartitions)**

**返回一个新的具有numPartition分区的DataFrame。**

类似于在RDD上定义的聚结，该操作导致窄依赖性，例如。 如果从1000个分区转到100个分区，则不会有随机播放，而是100个新分区中的每一个将分配10个当前分区。 如果要求更大数量的分区，它将保留在当前数量的分区。

但是，如果你正在做一个激烈的聚结，例如 到numPartitions = 1，这可能导致您的计算发生在比您喜欢的更少的节点上（例如，在numPartitions = 1的情况下，一个节点）。 为了避免这种情况，可以调用repartition（）。 这将添加一个随机播放步骤，但是意味着当前的上游分区将被并行执行（无论当前的分区是什么）。

>>> df.coalesce(1).rdd.getNumPartitions()

1

New in version 1.4.

**collect()**

以Row的列表形式返回所有记录。

>>> df.collect()

[Row(age=2, name=u'Alice'), Row(age=5, name=u'Bob')]

New in version 1.3.

**columns**

将列名作为列表返回。

>>> df.columns

['age', 'name']

New in version 1.3.

**corr(col1, col2, method=None)**

将DataFrame的两列的相关性计算为双精度值。 目前**只支持Pearson相关系数**。 DataFrame.corr（）和DataFrameStatFunctions.corr（）是彼此的别名。

|  |  |
| --- | --- |
| Parameters: | col1 – 第一列的名称  col2 – 第二列的名称  method – 相关方法。 目前只支持“梨子” |

New in version 1.4.

**count()**

**返回此DataFrame中的行数。**

>>> df.count()

2

New in version 1.3.

**cov(col1, col2) （协方差）**

**计算由其名称指定的给定列的样本协方差为双精度值。 DataFrame.cov（）和DataFrameStatFunctions.cov（）是别名。**

|  |  |
| --- | --- |
| Parameters: | col1 – 第一列的名称  col2 – 第二列的名称 |

New in version 1.4.

**createGlobalTempView(name)**

**使用此DataFrame创建全局临时视图。**

**此临时视图的生命周期与此Spark应用程序相关。 抛出TempTableAlreadyExistsException，如果视图名称已经存在于目录中。**

>>> df.createGlobalTempView("people")

>>> df2 = spark.sql("select \* from global\_temp.people")

>>> sorted(df.collect()) == sorted(df2.collect())

True

>>> df.createGlobalTempView("people")

Traceback (most recent call last):

...

AnalysisException: u"Temporary table 'people' already exists;"

>>> spark.catalog.dropGlobalTempView("people")

New in version 2.1.

**createOrReplaceTempView(name)**

使用此DataFrame创建或替换本地临时视图。

此临时表的生命周期与用于创建此DataFrame的SparkSession相关联。

>>> df.createOrReplaceTempView("people")

>>> df2 = df.filter(df.age > 3)

>>> df2.createOrReplaceTempView("people")

>>> df3 = spark.sql("select \* from people")

>>> sorted(df3.collect()) == sorted(df2.collect())

True

>>> spark.catalog.dropTempView("people")

New in version 2.0.

createTempView(name)

使用此DataFrame创建本地临时视图。

此临时表的生命周期与用于创建此DataFrame的SparkSession相关联。 抛出TempTableAlreadyExistsException，如果视图名称已经存在于目录中。

>>> df.createTempView("people")

>>> df2 = spark.sql("select \* from people")

>>> sorted(df.collect()) == sorted(df2.collect())

True

>>> df.createTempView("people")

Traceback (most recent call last):

...

AnalysisException: u"Temporary table 'people' already exists;"

>>> spark.catalog.dropTempView("people")

New in version 2.0.

**crossJoin(other)**

用另一个DataFrame返回笛卡尔乘积。

|  |  |
| --- | --- |
| Parameters: | other – 笛卡儿产品的右侧。 |

>>> df.select("age", "name").collect()

[Row(age=2, name=u'Alice'), Row(age=5, name=u'Bob')]

>>> df2.select("name", "height").collect()

[Row(name=u'Tom', height=80), Row(name=u'Bob', height=85)]

>>> df.crossJoin(df2.select("height")).select("age", "name", "height").collect()

[Row(age=2, name=u'Alice', height=80), Row(age=2, name=u'Alice', height=85),

Row(age=5, name=u'Bob', height=80), Row(age=5, name=u'Bob', height=85)]

New in version 2.1.

**crosstab(col1, col2)**

计算给定列的成对频率表。 也称为应急表。 每列的不同值的数量应小于1e4。 最多将返回1e6非零对频率。 每行的第一列将是col1的不同值，列名称将是col2的不同值。 第一列的名称将为$ col1\_ $ col2。 没有事件的对将为零。 DataFrame.crosstab（）和DataFrameStatFunctions.crosstab（）是别名。

|  |  |
| --- | --- |
| Parameters: | col1 – 第一列的名称。 不同的项目将使每行的第一个项目。  col2 – 第二列的名称。 不同的项目将使DataFrame的列名称。 |

New in version 1.4.

**cube(\*cols)**

使用指定的列为当前DataFrame创建一个多维多维数据集，因此我们可以在其上运行聚合。

>>> df.cube("name", df.age).count().orderBy("name", "age").show()

+-----+----+-----+

| name| age|count|

+-----+----+-----+

| null|null| 2|

| null| 2| 1|

| null| 5| 1|

|Alice|null| 1|

|Alice| 2| 1|

| Bob|null| 1|

| Bob| 5| 1|

+-----+----+-----+

New in version 1.4.

**describe(\*cols)**

计算数字和字符串列的统计信息。

这包括count，mean，stddev，min和max。 如果没有列，则此函数计算所有数字或字符串列的统计信息。

注意

此功能用于探索性数据分析，因为我们不保证生成的DataFrame的模式的向后兼容性。

>>> df.describe(['age']).show()

+-------+------------------+

|summary| age|

+-------+------------------+

| count| 2|

| mean| 3.5|

| stddev|2.1213203435596424|

| min| 2|

| max| 5|

+-------+------------------+

>>> df.describe().show()

+-------+------------------+-----+

|summary| age| name|

+-------+------------------+-----+

| count| 2| 2|

| mean| 3.5| null|

| stddev|2.1213203435596424| null|

| min| 2|Alice|

| max| 5| Bob|

+-------+------------------+-----+

New in version 1.3.1.

**distinct()**

返回包含此DataFrame中不同行的新DataFrame。

>>> df.distinct().count()

2

New in version 1.3.

**drop(\*cols)**

返回一个新的DataFrame，删除指定的列。 如果模式不包含给定的列名，这是一个无操作的。

|  |  |
| --- | --- |
| Parameters: | cols – 要删除的列的字符串名称或要删除的列，或要删除的列的字符串名称列表。 |

>>> df.drop('age').collect()

[Row(name=u'Alice'), Row(name=u'Bob')]

>>> df.drop(df.age).collect()

[Row(name=u'Alice'), Row(name=u'Bob')]

>>> df.join(df2, df.name == df2.name, 'inner').drop(df.name).collect()

[Row(age=5, height=85, name=u'Bob')]

>>> df.join(df2, df.name == df2.name, 'inner').drop(df2.name).collect()

[Row(age=5, name=u'Bob', height=85)]

>>> df.join(df2, 'name', 'inner').drop('age', 'height').collect()

[Row(name=u'Bob')]

New in version 1.4.

dropDuplicates(subset=None)

返回一个新的DataFrame，删除重复的行，可选地只考虑某些列。

[**drop\_duplicates()**](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame.drop_duplicates) **is an alias for** [**dropDuplicates()**](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame.dropDuplicates)**.**

>>> from pyspark.sql import Row

>>> df = sc.parallelize([ \

... Row(name='Alice', age=5, height=80), \

... Row(name='Alice', age=5, height=80), \

... Row(name='Alice', age=10, height=80)]).toDF()

>>> df.dropDuplicates().show()

+---+------+-----+

|age|height| name|

+---+------+-----+

| 5| 80|Alice|

| 10| 80|Alice|

+---+------+-----+

>>> df.dropDuplicates(['name', 'height']).show()

+---+------+-----+

|age|height| name|

+---+------+-----+

| 5| 80|Alice|

+---+------+-----+

New in version 1.4.

drop\_duplicates(subset=None)

drop\_duplicates（）是dropDuplicates（）的别名。

版本1.4中的新功能。

**dropna(how='any', thresh=None, subset=None)**

返回一个新的DataFrame，省略具有空值的行。 DataFrame.dropna（）和DataFrameNaFunctions.drop（）是彼此的别名。

|  |  |
| --- | --- |
| Parameters: | how –“any”或“all”。 如果“any”，如果包含任何空值，则删除一行。 如果“all”，只有当所有值都为空时，才能放置一行。  thresh – int，default None如果指定，则删除具有小于thresh非空值的行。 这将覆盖如何参数。  subset –需要考虑的列名称的可选列表。 |

>>> df4.na.drop().show()

+---+------+-----+

|age|height| name|

+---+------+-----+

| 10| 80|Alice|

+---+------+-----+

New in version 1.3.1.

**dtypes**

将所有列名称及其数据类型作为列表返回。

>>> df.dtypes

[('age', 'int'), ('name', 'string')]

New in version 1.3.

**explain(extended=False)**

将（逻辑和物理）计划打印到控制台进行调试。

|  |  |
| --- | --- |
| Parameters: | extended –布尔值，默认为False。 如果为假，只打印物理计划。 |

>>> df.explain()

== Physical Plan ==

Scan ExistingRDD[age#0,name#1]

>>> df.explain(True)

== Parsed Logical Plan ==

...

== Analyzed Logical Plan ==

...

== Optimized Logical Plan ==

...

== Physical Plan ==

...

New in version 1.3.

**fillna(value, subset=None)**

**替换空值，na.fill（）的别名。 DataFrame.fillna（）和DataFrameNaFunctions.fill（）是彼此的别名。**

|  |  |
| --- | --- |
| Parameters: | value – int，long，float，string或dict。 用以替换空值的值。 如果该值为dict，则忽略该子集，并且值必须是从列名（字符串）到替换值的映射。 替换值必须是int，long，float或string。  subset – 需要考虑的列名称的可选列表。 子集中指定的不具有匹配数据类型的列将被忽略。 例如，如果value是字符串，并且子集包含非字符串列，则非字符串列将被简单地忽略。 |

>>> df4.na.fill(50).show()

+---+------+-----+

|age|height| name|

+---+------+-----+

| 10| 80|Alice|

| 5| 50| Bob|

| 50| 50| Tom|

| 50| 50| null|

+---+------+-----+

>>> df4.na.fill({'age': 50, 'name': 'unknown'}).show()

+---+------+-------+

|age|height| name|

+---+------+-------+

| 10| 80| Alice|

| 5| null| Bob|

| 50| null| Tom|

| 50| null|unknown|

+---+------+-------+

New in version 1.3.1.

**filter(condition)**

使用给定条件过滤行。

where（）是filter（）的别名。

|  |  |
| --- | --- |
| Parameters: | condition – 一个列的类型.BooleanType或一个字符串的SQL表达式。 |

>>> df.filter(df.age > 3).collect()

[Row(age=5, name=u'Bob')]

>>> df.where(df.age == 2).collect()

[Row(age=2, name=u'Alice')]

>>> df.filter("age > 3").collect()

[Row(age=5, name=u'Bob')]

>>> df.where("age = 2").collect()

[Row(age=2, name=u'Alice')]

New in version 1.3.

**first()**

以行的形式返回第一行。

>>> df.first()

Row(age=2, name=u'Alice')

New in version 1.3.

**foreach(f) （针对函数使用，类似apply（lambda x :function()）**

**将f函数应用于此DataFrame的所有行。**

This is a shorthand for df.rdd.foreach().

>>> def f(person):

... print(person.name)

>>> df.foreach(f)

New in version 1.3.

**foreachPartition(f) (针对分区)**

将f函数应用于此DataFrame的每个分区。

这是df.rdd.foreachPartition（）的缩写。

>>> def f(people):

... for person in people:

... print(person.name)

>>> df.foreachPartition(f)

New in version 1.3.

**freqItems(cols, support=None)**

**找到频繁的项目列，可能有误报。** 使用由Karp，Schenker和Papadimitriou提出的“http://dx.doi.org/10.1145/762471.762473”中描述的频繁元素计数算法。 DataFrame.freqItems（）和DataFrameStatFunctions.freqItems（）是别名。

注意

此功能用于探索性数据分析，因为我们不保证生成的DataFrame的模式的向后兼容性。

|  |  |
| --- | --- |
| Parameters: | cols – 用于计算频繁项目作为列表或字符串元组的列的名称。  support – 考虑一个项目“频繁”的频率。 默认值为1％。 支撑必须大于1e-4。 |

New in version 1.4.

**groupBy(\*cols)**

使用指定的列对DataFrame进行分组，因此我们可以在其上运行聚合。 有关所有可用的聚合函数，请参阅GroupedData。

**groupby（）是groupBy（）的别名。**

|  |  |
| --- | --- |
| Parameters: | cols –要分组的列的列表。 每个元素应该是一个列名（string）或一个表达式（Column）。 |

>>> df.groupBy().avg().collect()

[Row(avg(age)=3.5)]

>>> sorted(df.groupBy('name').agg({'age': 'mean'}).collect())

[Row(name=u'Alice', avg(age)=2.0), Row(name=u'Bob', avg(age)=5.0)]

>>> sorted(df.groupBy(df.name).avg().collect())

[Row(name=u'Alice', avg(age)=2.0), Row(name=u'Bob', avg(age)=5.0)]

>>> sorted(df.groupBy(['name', df.age]).count().collect())

[Row(name=u'Alice', age=2, count=1), Row(name=u'Bob', age=5, count=1)]

New in version 1.3.

**groupby(\*cols) （根据列进行的分组）**

groupby（）是groupBy（）的别名。

版本1.4中的新功能。

**head(n=None)**

**返回前n行。**

Note

只有在生成的数组预计为小的时候才应该使用这个方法，因为所有的数据被加载到驱动程序的内存中。

|  |  |
| --- | --- |
| Parameters: | n – int，default 1.要返回的行数。 |
| Returns: | 如果n大于1，返回一行列表。 如果n为1，则返回一行。 |

>>> df.head()

Row(age=2, name=u'Alice')

>>> df.head(1)

[Row(age=2, name=u'Alice')]

New in version 1.3.

**intersect(other)**

**在这个Frame和另一个Frame中返回一个包含行的新DataFrame。**

这相当于SQL中的INTERSECT。

1.3版新功能

**isLocal() （返回bool类型）**

**如果collect（）和take（）方法可以在本地运行（没有任何Spark执行程序），则返回True。**

1.3版新功能

**isStreaming**

如果此数据集包含一个或多个连续返回数据的源，则返回true。 从数据流源读取数据的数据集必须使用DataStreamWriter中的start（）方法作为StreamingQuery执行。 返回单个答案（例如，count（）或collect（））的方法将在存在流源时出现AnalysisException。

注意

试验

2.0版新功能

**join(other, on=None, how=None)**

**使用给定的连接表达式与另一个DataFrame连接。**

|  |  |
| --- | --- |
| Parameters: | other – 加入的右侧  on – 连接列名称的字符串，列名称列表，连接表达式（列）或列列表。 如果on是一个字符串或指示连接列的名称的字符串列表，则列必须存在于两侧，并且这将执行等于连接。  how –str，默认内。 必须是其中之一：inner，cross，outer，full，full\_outer，left，left\_outer，right，right\_outer，left\_semi和left\_anti。 |

以下执行df1和df2之间的完整外部连接。

>>> df.join(df2, df.name == df2.name, 'outer').select(df.name, df2.height).collect()

[Row(name=None, height=80), Row(name=u'Bob', height=85), Row(name=u'Alice', height=None)]

>>> df.join(df2, 'name', 'outer').select('name', 'height').collect()

[Row(name=u'Tom', height=80), Row(name=u'Bob', height=85), Row(name=u'Alice', height=None)]

>>> cond = [df.name == df3.name, df.age == df3.age]

>>> df.join(df3, cond, 'outer').select(df.name, df3.age).collect()

[Row(name=u'Alice', age=2), Row(name=u'Bob', age=5)]

>>> df.join(df2, 'name').select(df.name, df2.height).collect()

[Row(name=u'Bob', height=85)]

>>> df.join(df4, ['name', 'age']).select(df.name, df.age).collect()

[Row(name=u'Bob', age=5)]

New in version 1.3.

**limit(num)**

**将结果计数限制为指定的数字。**

>>> df.limit(1).collect()

[Row(age=2, name=u'Alice')]

>>> df.limit(0).collect()

[]

New in version 1.3.

**na**

**返回一个用于处理缺失值的DataFrameNaFunction。**

New in version 1.3.1.

**orderBy(\*cols, \*\*kwargs)**

返回按指定列排序的新DataFrame。

|  |  |
| --- | --- |
| Parameters: | cols – 要排序的列或列名称列表。  ascending – 布尔值或布尔值列表（默认为True）。 排序升序降序。 指定多个排序顺序的列表。 如果指定了列表，列表的长度必须等于列的长度。 |

>>> df.sort(df.age.desc()).collect()

[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]

>>> df.sort("age", ascending=False).collect()

[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]

>>> df.orderBy(df.age.desc()).collect()

[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]

>>> from pyspark.sql.functions import \*

>>> df.sort(asc("age")).collect()

[Row(age=2, name=u'Alice'), Row(age=5, name=u'Bob')]

>>> df.orderBy(desc("age"), "name").collect()

[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]

>>> df.orderBy(["age", "name"], ascending=[0, 1]).collect()

[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]

New in version 1.3.

**persist(storageLevel=StorageLevel(True, True, False, False, 1))**

设置存储级别，以便在第一次计算之后，跨DataGrame的内容持久化。 如果DataFrame尚未设置存储级别，则这只能用于分配新的存储级别。 如果没有指定存储级别默认值为（MEMORY\_AND\_DISK）。

注意

默认存储级别已更改为MEMORY\_AND\_DISK以匹配2.0中的Scala。

New in version 1.3.

**printSchema()**

**以树形式打印模式。**

>>> df.printSchema()

root

|-- age: integer (nullable = true)

|-- name: string (nullable = true)

New in version 1.3.

**randomSplit(weights, seed=None)**

**随机分配此DataFrame与提供的权重。**

|  |  |
| --- | --- |
| Parameters: | weights – 随机分配此DataFrame与提供的权重。  seed – The seed for sampling. |

>>> splits = df4.randomSplit([1.0, 2.0], 24)

>>> splits[0].count()

1

>>> splits[1].count()

3

New in version 1.4.

rdd

以Row的pyspark.RDD形式返回内容。

New in version 1.3.

**registerTempTable(name)**

**使用给定的名称将此RDD注册为临时表。**

此临时表的生命周期与用于创建此DataFrame的SQLContext相关。

>>> df.registerTempTable("people")

>>> df2 = spark.sql("select \* from people")

>>> sorted(df.collect()) == sorted(df2.collect())

True

>>> spark.catalog.dropTempView("people")

Note

**在2.0中弃用，请改用createOrReplaceTempView。**

New in version 1.3.

**repartition(numPartitions, \*cols)**

返回由给定的分区表达式分区的新DataFrame。 生成的DataFrame是哈希分区的。

numPartition可以是一个int来指定目标分区或一列。 如果是Column，它将被用作第一个分区列。 如果未指定，则使用默认分区数。

在版本1.6中更改：添加可选参数以指定分区列。 如果指定了分区列，那么也可以选择numPartitions。

>>> df.repartition(10).rdd.getNumPartitions()

10

>>> data = df.union(df).repartition("age")

>>> data.show()

+---+-----+

|age| name|

+---+-----+

| 5| Bob|

| 5| Bob|

| 2|Alice|

| 2|Alice|

+---+-----+

>>> data = data.repartition(7, "age")

>>> data.show()

+---+-----+

|age| name|

+---+-----+

| 2|Alice|

| 5| Bob|

| 2|Alice|

| 5| Bob|

+---+-----+

>>> data.rdd.getNumPartitions()

7

>>> data = data.repartition("name", "age")

>>> data.show()

+---+-----+

|age| name|

+---+-----+

| 5| Bob|

| 5| Bob|

| 2|Alice|

| 2|Alice|

+---+-----+

New in version 1.3.

**replace(to\_replace, value, subset=None)**

**返回一个新的DataFrame替换另一个值的值。 DataFrame.replace（）和DataFrameNaFunctions.replace（）是彼此的别名。**

|  |  |
| --- | --- |
| Parameters: | to\_replace –int，long，float，string或list。 要更换的价值 如果该值为dict，则忽略该值，并且to\_replace必须是从列名（字符串）到替换值的映射。 要替换的值必须是int，long，float或string。  value – int，long，float，string或list。 用来代替孔的价值。 替换值必须是int，long，float或string。 如果value是一个列表或元组，值应与to\_replace的长度相同。  subset – 需要考虑的列名称的可选列表。 子集中指定的不具有匹配数据类型的列将被忽略。 例如，如果value是字符串，并且子集包含非字符串列，则非字符串列将被简单地忽略。 |

>>> df4.na.replace(10, 20).show()

+----+------+-----+

| age|height| name|

+----+------+-----+

| 20| 80|Alice|

| 5| null| Bob|

|null| null| Tom|

|null| null| null|

+----+------+-----+

>>> df4.na.replace(['Alice', 'Bob'], ['A', 'B'], 'name').show()

+----+------+----+

| age|height|name|

+----+------+----+

| 10| 80| A|

| 5| null| B|

|null| null| Tom|

|null| null|null|

+----+------+----+

New in version 1.4.

**rollup(\*cols)**

**使用指定的列为当前DataFrame创建多维汇总，因此我们可以在其上运行汇总。**

>>> df.rollup("name", df.age).count().orderBy("name", "age").show()

+-----+----+-----+

| name| age|count|

+-----+----+-----+

| null|null| 2|

|Alice|null| 1|

|Alice| 2| 1|

| Bob|null| 1|

| Bob| 5| 1|

+-----+----+-----+

New in version 1.4.

**sample(withReplacement, fraction, seed=None) （样本子集）**

返回此DataFrame的采样子集。

注意

这并不能保证提供给定DataFrame的总计数所指定的分数。

>>> df.sample(False, 0.5, 42).count()

2

New in version 1.3.

**sampleBy(col, fractions, seed=None)**

基于每个层上给出的分数，返回没有替换的分层样本。

|  |  |
| --- | --- |
| Parameters: | col – 定义层的列  fractions – 每层采样分数。 如果没有指定层，我们将其分数视为零。  seed – 随机种子 |
| Returns: | 一个新的DataFrame，代表分层样本 |

>>> from pyspark.sql.functions import col

>>> dataset = sqlContext.range(0, 100).select((col("id") % 3).alias("key"))

>>> sampled = dataset.sampleBy("key", fractions={0: 0.1, 1: 0.2}, seed=0)

>>> sampled.groupBy("key").count().orderBy("key").show()

+---+-----+

|key|count|

+---+-----+

| 0| 5|

| 1| 9|

+---+-----+

New in version 1.5.

**schema**

返回此DataFrame的模式作为pyspark.sql.types.StructType。

>>> df.schema

**StructType（名单（StructField（年龄，IntegerType，真），StructField（姓名，StringType，真）））**

1.3版新功能

**select(\*cols)**

**投射一组表达式并返回一个新的DataFrame。**

|  |  |
| --- | --- |
| Parameters: | cols – 列名（字符串）或表达式（列）列表。 如果其中一个列名称为“\*”，那么该列将被扩展为包括当前DataFrame中的所有列。 |

>>> df.select('\*').collect()

[Row(age=2, name=u'Alice'), Row(age=5, name=u'Bob')]

>>> df.select('name', 'age').collect()

[Row(name=u'Alice', age=2), Row(name=u'Bob', age=5)]

>>> df.select(df.name, (df.age + 10).alias('age')).collect()

[Row(name=u'Alice', age=12), Row(name=u'Bob', age=15)]

New in version 1.3.

**selectExpr(\*expr) （一种数据选择表达式）**

**投射一组SQL表达式并返回一个新的DataFrame。**

**这是接受SQL表达式的select（）的变体。**

>>> df.selectExpr("age \* 2", "abs(age)").collect()

[Row((age \* 2)=4, abs(age)=2), Row((age \* 2)=10, abs(age)=5)]

New in version 1.3.

**show(n=20, truncate=True)**

将前n行打印到控制台。

|  |  |
| --- | --- |
| Parameters: | n –要显示的行数。  truncate –如果设置为True，则默认截断长度超过20个字符的字符串。 如果设置为大于1的数字，则截断长字符串以截取长度，并对齐对齐单元格。 |

>>> df

DataFrame[age: int, name: string]

>>> df.show()

+---+-----+

|age| name|

+---+-----+

| 2|Alice|

| 5| Bob|

+---+-----+

>>> df.show(truncate=3)

+---+----+

|age|name|

+---+----+

| 2| Ali|

| 5| Bob|

+---+----+

New in version 1.3.

sort(\*cols, \*\*kwargs)

返回按指定列排序的新DataFrame。

|  |  |
| --- | --- |
| Parameters: | cols – 要排序的列或列名称列表。  ascending –布尔值或布尔值列表（默认为True）。 排序升序降序。 指定多个排序顺序的列表。 如果指定了列表，列表的长度必须等于列的长度。 |

>>> df.sort(df.age.desc()).collect()

[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]

>>> df.sort("age", ascending=False).collect()

[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]

>>> df.orderBy(df.age.desc()).collect()

[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]

>>> from pyspark.sql.functions import \*

>>> df.sort(asc("age")).collect()

[Row(age=2, name=u'Alice'), Row(age=5, name=u'Bob')]

>>> df.orderBy(desc("age"), "name").collect()

[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]

>>> df.orderBy(["age", "name"], ascending=[0, 1]).collect()

[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]

New in version 1.3.

**sortWithinPartitions(\*cols, \*\*kwargs)**

返回一个新的DataFrame，每个分区按照指定的列排序。

|  |  |
| --- | --- |
| Parameters: | cols – 要排序的列或列名称列表。  ascending – 布尔值或布尔值列表（默认为True）。 排序升序降序。 指定多个排序顺序的列表。 如果指定了列表，列表的长度必须等于列的长度。 |

>>> df.sortWithinPartitions("age", ascending=False).show()

+---+-----+

|age| name|

+---+-----+

| 2|Alice|

| 5| Bob|

+---+-----+

New in version 1.6.

**stat**

**返回统计功能的DataFrameStatFunctions。**

版本1.4中的新功能。

**storageLevel （与persist）类似**

**获取DataFrame的当前存储级别。**

>>> df.storageLevel

StorageLevel(False, False, False, False, 1)

>>> df.cache().storageLevel

StorageLevel(True, True, False, True, 1)

>>> **df2.persist(StorageLevel.DISK\_ONLY\_2).storageLevel**

StorageLevel(True, False, False, False, 2)

New in version 2.1.

**subtract(other)**

**返回一个新的DataFrame，该框架中包含行，但不在另一个Frame中。**

这相当于SQL中的EXCEPT。

1.3版新功能

**take(num) ：取num行数据记录**

**返回第一个num行作为列的列。**

>>> df.take(2)

[Row(age=2, name=u'Alice'), Row(age=5, name=u'Bob')]

New in version 1.3.

**toDF(\*cols) （转为DataFrame）**

返回一个新类：具有新的指定列名称的DataFrame

|  |  |
| --- | --- |
| Parameters: | cols –新列名列表（字符串） |

>>> df.toDF('f1', 'f2').collect()

[Row(f1=2, f2=u'Alice'), Row(f1=5, f2=u'Bob')]

**toJSON(use\_unicode=True)**

将**DataFrame**转换为字符串的RDD。

**每一行都将转换为JSON文档作为返回的RDD中的一个元素。**

>>> df.toJSON().first()

u'{"age":2,"name":"Alice"}'

New in version 1.3.

**toLocalIterator()**

返回一个包含此DataFrame中所有行的迭代器。 迭代器将消耗与DataFrame中最大分区一样多的内存。

>>> list(df.toLocalIterator())

[Row(age=2, name=u'Alice'), Row(age=5, name=u'Bob')]

New in version 2.0.

toPandas()

将此DataFrame的内容返回为Pandas pandas.DataFrame。

这只有在熊猫安装和可用的情况下才可用。

Note

This method should only be used if the resulting Pandas’s DataFrame is expected to be small, as all the data is loaded into the driver’s memory.

**>>> df.toPandas()**

age name

0 2 Alice

1 5 Bob

New in version 1.3.

**union(other) （合并）**

返回一个新的DataFrame，其中包含该Frame和另一个Frame中的行的并集。

**这相当于SQL中的UNION ALL。 要做一个SQL风格的集合联合（它执行元素的重复数据删除），请使用此函数后跟一个distinct。**

New in version 2.0.

**unionAll(other)**

返回一个新的DataFrame，其中包含该Frame和另一个Frame中的行的并集。

Note

在2.0中不推荐使用union。

New in version 1.3.

**unpersist(blocking=False)**

将DataFrame标记为非持久性，并从内存和磁盘中删除所有块。

注意

在2.0中，阻止默认值已更改为False以匹配Scala。

1.3版新功能

**where(condition) 与filter一致**

**where（）是filter（）的别名。**

1.3版新功能

**withColumn(colName, col)**

**通过添加列或替换具有相同名称的现有列来返回新的DataFrame。**

|  |  |
| --- | --- |
| Parameters: | colName – 字符串，新列的名称。  col – 新列的列表达式。 |

>>> df.withColumn('age2', df.age + 2).collect()

[Row(age=2, name=u'Alice', age2=4), Row(age=5, name=u'Bob', age2=7)]

New in version 1.3.

**withColumnRenamed(existing, new)**

通过重命名现有列返回新的DataFrame。 如果模式不包含给定的列名，这是一个无操作的。

|  |  |
| --- | --- |
| Parameters: | existing – 字符串，要重命名的现有列的名称。  col – 字符串，列的新名称。 |

>>> df.withColumnRenamed('age', 'age2').collect()

[Row(age2=2, name=u'Alice'), Row(age2=5, name=u'Bob')]

New in version 1.3.

**withWatermark(eventTime, delayThreshold)**

为此DataFrame定义事件时间水印。 一个水印跟踪一个时间点，在此之前我们假设没有更多的后期数据将要到达。

Spark将使用此水印用于多种目的：

知道什么时候可以确定给定的时间窗口聚合，从而可以在使用不允许更新的输出模式时发出。

最小化我们需要保持持续聚合的状态量。

通过查看查询中所有分区中看到的MAX（eventTime）减去用户指定的delayThreshold来计算当前水印。 由于跨越分区协调此值的成本，所使用的实际水印仅保证在实际事件时间后至少为延迟阈值。 在某些情况下，我们仍然可以处理超过delayThreshold的记录。

|  |  |
| --- | --- |
| Parameters: | eventTime – 包含行的事件时间的列的名称。  delayThreshold –相对于以间隔（例如“1分钟”或“5小时”）的形式处理的最新记录，等待数据到达的最小延迟。 |

Note

Experimental

>>> sdf.select('name', sdf.time.cast('timestamp')).withWatermark('time', '10 minutes')

DataFrame[name: string, time: timestamp]

New in version 2.1.

write

用于将非流式传输DataFrame的内容保存到外部存储器中的接口。

|  |  |
| --- | --- |
| Returns: | [DataFrameWriter](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrameWriter) |

New in version 1.4.

**writeStream**

**用于将流式DataFrame的内容保存到外部存储器中的接口。**

注意

实验。

返回：

**DataStreamWriter**

2.0版新功能

# GroupedData

**class pyspark.sql.GroupedData(jgd, sql\_ctx)**

**由DataFrame.groupBy（）创建的一组DataFrame聚合方法。**

注意

试验

1.3版新功能

**agg(\*exprs)**

**计算聚合并将结果作为DataFrame返回。**

**可用的聚合函数是avg，max，min，sum，count。**

如果exprs是从字符串到字符串的单个dict映射，则该键是要执行聚合的列，该值是聚合函数。

或者，exprs也可以是聚合列表达式的列表。

|  |  |
| --- | --- |
| Parameters: | exprs – a dict mapping from column name (string) to aggregate functions (string), or a list of [Column](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.Column). |

>>> gdf = df.groupBy(df.name)

>>> sorted(gdf.agg({"\*": "count"}).collect())

[Row(name=u'Alice', count(1)=1), Row(name=u'Bob', count(1)=1)]

>>> from pyspark.sql import functions as F

>>> sorted(gdf.agg(F.min(df.age)).collect())

[Row(name=u'Alice', min(age)=2), Row(name=u'Bob', min(age)=5)]

New in version 1.3.

**avg(\*cols)**

**计算每个组的每个数字列的平均值。**

**mean（）是avg（）的别名。**

|  |  |
| --- | --- |
| Parameters: | cols – list of column names (string). Non-numeric columns are ignored. |

>>> df.groupBy().avg('age').collect()

[Row(avg(age)=3.5)]

>>> df3.groupBy().avg('age', 'height').collect()

[Row(avg(age)=3.5, avg(height)=82.5)]

New in version 1.3.

**count()**

计算每个组的记录数。

>>> sorted(df.groupBy(df.age).count().collect())

[Row(age=2, count=1), Row(age=5, count=1)]

New in version 1.3.

**max(\*cols)**

计算每个组的每个数字列的最大值。

>>> df.groupBy().max('age').collect()

[Row(max(age)=5)]

>>> df3.groupBy().max('age', 'height').collect()

[Row(max(age)=5, max(height)=85)]

New in version 1.3.

**mean(\*cols)**

计算每个组的每个数字列的平均值。

mean（）是avg（）的别名。

|  |  |
| --- | --- |
| Parameters: | cols – list of column names (string). Non-numeric columns are ignored. |

>>> df.groupBy().mean('age').collect()

[Row(avg(age)=3.5)]

>>> df3.groupBy().mean('age', 'height').collect()

[Row(avg(age)=3.5, avg(height)=82.5)]

New in version 1.3.

**min(\*cols)**

计算每个组的每个数字列的最小值。

|  |  |
| --- | --- |
| Parameters: | cols – 列名列表（字符串）。 非数字列被忽略。 |

>>> df.groupBy().min('age').collect()

[Row(min(age)=2)]

>>> df3.groupBy().min('age', 'height').collect()

[Row(min(age)=2, min(height)=80)]

New in version 1.3.

**pivot(pivot\_col, values=None)**

枢转当前[[DataFrame]]的一列，并执行指定的聚合。 有两个版本的pivot函数：一个要求调用者指定不同值的列表，而不是一个。 后者更简洁但效率更低，因为Spark需要首先在内部计算不同值的列表。

|  |  |
| --- | --- |
| Parameters: | pivot\_col – 要转动的列的名称。  values – 将转换为输出DataFrame中的列的值列表。 |

# 将每个课程的课程每年的收入总和计算为单独的列

>>> df4.groupBy("year").pivot("course", ["dotNET", "Java"]).sum("earnings").collect()

[Row(year=2012, dotNET=15000, Java=20000), Row(year=2013, dotNET=48000, Java=30000)]

# Or without specifying column values (less efficient)

>>> df4.groupBy("year").pivot("course").sum("earnings").collect()

[Row(year=2012, Java=20000, dotNET=15000), Row(year=2013, Java=30000, dotNET=48000)]

New in version 1.6.

**sum(\*cols)**

计算每个组的每个数字列的总和。

|  |  |
| --- | --- |
| Parameters: | cols – 列名列表（字符串）。 非数字列被忽略。 |

>>> df.groupBy().sum('age').collect()

[Row(sum(age)=7)]

>>> df3.groupBy().sum('age', 'height').collect()

[Row(sum(age)=7, sum(height)=165)]

New in version 1.3.

# Column

**class pyspark.sql.Column(jc)**

DataFrame中的一列。

列实例可以通过以下方式创建：

# 1. Select a column out of a DataFrame

df.colName

df["colName"]

# 2. Create from an expression

df.colName + 1

1 / df.colName

New in version 1.3.

**alias(\*alias)**

返回此列使用新的名称或名称（在返回多个列的表达式（例如爆炸）的情况下）别名。

>>> df.select(df.age.alias("age2")).collect()

[Row(age2=2), Row(age2=5)]

New in version 1.3.

**asc()**

**根据给定列名称的升序返回排序表达式。**

**astype(dataType)**

**astype（）是cast（）的别名。**

版本1.4中的新功能。

**between(lowerBound, upperBound) （可用于选值）**

**一个布尔表达式，如果此表达式的值位于给定列之间，则该值将被计算为true。**

>>> df.select(df.name, df.age.between(2, 4)).show()

+-----+---------------------------+

| name|((age >= 2) AND (age <= 4))|

+-----+---------------------------+

|Alice| true|

| Bob| false|

+-----+---------------------------+

New in version 1.3.

**bitwiseAND(other)**

**二进制运算符**

**bitwiseOR(other)**

**二进制运算符**

**bitwiseXOR(other)**

二进制运算符

**cast(dataType)**

将列转换为dataType类型。

>>> df.select(df.age.cast("string").alias('ages')).collect()

[Row(ages=u'2'), Row(ages=u'5')]

>>> df.select(df.age.cast(StringType()).alias('ages')).collect()

[Row(ages=u'2'), Row(ages=u'5')]

New in version 1.3.

**desc()**

**基于给定列名的降序返回排序表达式。**

**endswith(other)**

二进制运算符

**getField(name)**

一个在StructField中通过名称获取字段的表达式。

>>> from pyspark.sql import Row

>>> df = sc.parallelize([Row(r=Row(a=1, b="b"))]).toDF()

>>> df.select(df.r.getField("b")).show()

+---+

|r.b|

+---+

| b|

+---+

>>> df.select(df.r.a).show()

+---+

|r.a|

+---+

| 1|

+---+

New in version 1.3.

**getItem(key)**

一个表达式，它从一个列表中获得一个位置序号的项目，或者通过一个字母获取一个项目。

>>> df = sc.parallelize([([1, 2], {"key": "value"})]).toDF(["l", "d"])

>>> df.select(df.l.getItem(0), df.d.getItem("key")).show()

+----+------+

|l[0]|d[key]|

+----+------+

| 1| value|

+----+------+

>>> df.select(df.l[0], df.d["key"]).show()

+----+------+

|l[0]|d[key]|

+----+------+

| 1| value|

+----+------+

New in version 1.3.

**isNotNull()**

如果当前表达式不为空，则为真。

**isNull()**

如果当前表达式为空，则为真。

isin(\*cols)

一个布尔表达式，如果该表达式的值包含在参数的评估值中，则该值将被计算为true。

>>> df[df.name.isin("Bob", "Mike")].collect()

[Row(age=5, name=u'Bob')]

>>> df[df.age.isin([1, 2, 3])].collect()

[Row(age=2, name=u'Alice')]

New in version 1.5.

**like(other)**

二进制操作符

**name(\*alias)**

name（）是别名（）的别名。

2.0版新功能

**otherwise(value)**

评估条件列表并返回多个可能的结果表达式之一。 如果不调用Column.otherwise（），则返回无匹配条件的None。

请参阅pyspark.sql.functions.when（），例如用法。

|  |  |
| --- | --- |
| Parameters: | value –一个文字值或一个Column表达式。 |

>>> from pyspark.sql import functions as F

>>> df.select(df.name, F.when(df.age > 3, 1).otherwise(0)).show()

+-----+-------------------------------------+

| name|CASE WHEN (age > 3) THEN 1 ELSE 0 END|

+-----+-------------------------------------+

|Alice| 0|

| Bob| 1|

+-----+-------------------------------------+

New in version 1.4.

over(window)

定义一个窗口列。

|  |  |
| --- | --- |
| Parameters: | window – a [WindowSpec](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.WindowSpec) |
| Returns: | a Column |

>>> from pyspark.sql import Window

>>> window = Window.partitionBy("name").orderBy("age").rowsBetween(-1, 1)

>>> from pyspark.sql.functions import rank, min

>>> # df.select(rank().over(window), min('age').over(window))

New in version 1.4.

**rlike(other)**

二进制操作符

**startswith(other)**

二进制操作符

**substr(startPos, length)**

返回一列是子列的列。

|  |  |
| --- | --- |
| Parameters: | startPos – 开始位置（int或Column）  length – 子串的长度（int或Column） |

>>> df.select(df.name.substr(1, 3).alias("col")).collect()

[Row(col=u'Ali'), Row(col=u'Bob')]

New in version 1.3.

when(condition, value)

评估条件列表并返回多个可能的结果表达式之一。 如果不调用Column.otherwise（），则返回无匹配条件的None。

请参阅pyspark.sql.functions.when（），例如用法。

|  |  |
| --- | --- |
| Parameters: | condition –一个布尔的列表达式。  value – 一个文字值或一个Column表达式。 |

>>> from pyspark.sql import functions as F

>>> df.select(df.name, F.when(df.age > 4, 1).when(df.age < 3, -1).otherwise(0)).show()

+-----+------------------------------------------------------------+

| name|CASE WHEN (age > 4) THEN 1 WHEN (age < 3) THEN -1 ELSE 0 END|

+-----+------------------------------------------------------------+

|Alice| -1|

| Bob| 1|

+-----+------------------------------------------------------------+

New in version 1.4.

# Row

**class pyspark.sql.Row**

DataFrame中的一行。 可以访问其中的字段：

喜欢属性（row.key）

像字典值（row [key]）

行中的键将搜索行键。

**Row可用于通过使用命名参数来创建行对象，这些字段将按名称排序。**

>>> row = Row(name="Alice", age=11)

>>> row

Row(age=11, name='Alice')

>>> row['name'], row['age']

('Alice', 11)

>>> row.name, row.age

('Alice', 11)

>>> 'name' in row

True

>>> 'wrong\_key' in row

False

Row也可用于创建另一个Row类，然后可以用于创建Row对象，如

>>> Person = Row("name", "age")

>>> Person

<Row(name, age)>

>>> 'name' in Person

True

>>> 'wrong\_key' in Person

False

>>> Person("Alice", 11)

Row(name='Alice', age=11)

**asDict(recursive=False)**

Return as an dict

|  |  |
| --- | --- |
| Parameters: | recursive – 将嵌套的行转换为dict（默认值：False）。 |

>>> Row(name="Alice", age=11).asDict() == {'name': 'Alice', 'age': 11}

True

>>> row = Row(key=1, value=Row(name='a', age=2))

>>> row.asDict() == {'key': 1, 'value': Row(age=2, name='a')}

True

>>> row.asDict(True) == {'key': 1, 'value': {'name': 'a', 'age': 2}}

True

# DataFrameNaFunctions

**class pyspark.sql.DataFrameNaFunctions(df)**

在DataFrame中处理缺少数据的功能。

版本1.4中的新功能。

**drop(how='any', thresh=None, subset=None)**

返回一个新的DataFrame，省略具有空值的行。 DataFrame.dropna（）和DataFrameNaFunctions.drop（）是彼此的别名。

|  |  |
| --- | --- |
| Parameters: | how – ‘any’ or ‘all’. If ‘any’, 如果包含任何空值，请放一行。 如果“全部”，只有当所有值都为空时，才能放置一行。  thresh – int, default如果指定，则删除具有小于thresh非空值的行。 这将覆盖如何参数。  subset – 需要考虑的列名称的可选列表。 |

>>> df4.na.drop().show()

+---+------+-----+

|age|height| name|

+---+------+-----+

| 10| 80|Alice|

+---+------+-----+

New in version 1.3.1.

**fill(value, subset=None)**

**替换空值，na.fill（）的别名。 DataFrame.fillna（）和DataFrameNaFunctions.fill（）是彼此的别名。**

|  |  |
| --- | --- |
| Parameters: | value – int, long, float, string, or dict. 用以替换空值的值。 如果该值为dict，则忽略该子集，并且值必须是从列名（字符串）到替换值的映射。 替换值必须是int，long，float或string。  subset – 需要考虑的列名称的可选列表。 子集中指定的不具有匹配数据类型的列将被忽略。 例如，如果value是字符串，并且子集包含非字符串列，则非字符串列将被简单地忽略。 |

>>> df4.na.fill(50).show()

+---+------+-----+

|age|height| name|

+---+------+-----+

| 10| 80|Alice|

| 5| 50| Bob|

| 50| 50| Tom|

| 50| 50| null|

+---+------+-----+

>>> df4.na.fill({'age': 50, 'name': 'unknown'}).show()

+---+------+-------+

|age|height| name|

+---+------+-------+

| 10| 80| Alice|

| 5| null| Bob|

| 50| null| Tom|

| 50| null|unknown|

+---+------+-------+

New in version 1.3.1.

**replace(to\_replace, value, subset=None)**

**返回一个新的DataFrame替换另一个值的值。 DataFrame.replace（）和DataFrameNaFunctions.replace（）是彼此的别名。**

|  |  |
| --- | --- |
| Parameters: | to\_replace – int, long, float, string, or list. 要更换的价值 如果该值为dict，则忽略该值，并且to\_replace必须是从列名（字符串）到替换值的映射。 要替换的值必须是int，long，float或string。  value – int, long, float, string, or list. 用来代替的价值。 替换值必须是int，long，float或string。 如果value是一个列表或元组，值应与to\_replace的长度相同。  subset – 需要考虑的列名称的可选列表。 子集中指定的不具有匹配数据类型的列将被忽略。 例如，如果value是字符串，并且子集包含非字符串列，则非字符串列将被简单地忽略。 |

>>> df4.na.replace(10, 20).show()

+----+------+-----+

| age|height| name|

+----+------+-----+

| 20| 80|Alice|

| 5| null| Bob|

|null| null| Tom|

|null| null| null|

+----+------+-----+

>>> df4.na.replace(['Alice', 'Bob'], ['A', 'B'], 'name').show()

+----+------+----+

| age|height|name|

+----+------+----+

| 10| 80| A|

| 5| null| B|

|null| null| Tom|

|null| null|null|

+----+------+----+

New in version 1.4.

# DataFrameStatFunctions

**class pyspark.sql.DataFrameStatFunctions(df)**

数据帧统计功能的功能。

版本1.4中的新功能。

**approxQuantile(col, probabilities, relativeError)**

计算DataFrame的数值列的近似分位数。

该算法的结果具有以下确定性界限：如果DataFrame具有N个元素，并且如果我们以概率p请求分数位错误err，则该算法将从DataFrame返回一个样本x，使得x的确切等级为 接近（p \* N）。 更确切地说，

floor((p - err) \* N) <= rank(x) <= ceil((p + err) \* N).

该方法实现了Greenwald-Khanna算法的变体（具有一些速度优化）。 该算法首先出现在Greenwald和Khanna的[[http://dx.doi.org/10.1145/375663.375670空间效率在线计算分位数概要]中）。

|  |  |
| --- | --- |
| Parameters: | col –数字列的名称  probabilities – 分位数概率列表每个数字必须属于[0,1]。 例如0是最小值，0.5是中位数，1是最大值。  relativeError – 相对目标精度达到（> = 0）。 如果设置为零，则计算确切的分位数，这可能非常昂贵。 请注意，大于1的值被接受，但结果与1相同。 |
| Returns: | 给定概率的近似分位数 |

New in version 2.0.

**corr(col1, col2, method=None)**

将DataFrame的两列的**相关性**计算为双精度值。 目前只支持**Pearson相关系数**。 DataFrame.corr（）和DataFrameStatFunctions.corr（）是彼此的别名。

|  |  |
| --- | --- |
| Parameters: | col1 – 第一列的名称  col2 – 第二列的名称  method – 相关方法。 目前只支持“pearson” |

New in version 1.4.

**cov(col1, col2)**

计算由其名称指定的给定列的样**本协方差为双精度值**。 DataFrame.cov（）和DataFrameStatFunctions.cov（）是别名。

|  |  |
| --- | --- |
| Parameters: | col1 – The name of the first column  col2 – The name of the second column |

New in version 1.4.

crosstab(col1, col2)

计算给定列的成对频率表。 也称为应急表。 每列的不同值的数量应小于1e4。 最多将返回1e6非零对频率。 每行的第一列将是col1的不同值，列名称将是col2的不同值。 第一列的名称将为$ col1\_ $ col2。 没有事件的对将为零。 DataFrame.crosstab（）和DataFrameStatFunctions.crosstab（）是别名。

|  |  |
| --- | --- |
| Parameters: | col1 – 第一列的名称。 不同的项目将使每行的第一个项目。  col2 – 第二列的名称。 不同的项目将使DataFrame的列名称。 |

New in version 1.4.

**freqItems(cols, support=None)**

找到频繁的项目列，可能有误报。 使用由Karp，Schenker和Papadimitriou提出的“http://dx.doi.org/10.1145/762471.762473”中描述的频繁元素计数算法。 DataFrame.freqItems（）和DataFrameStatFunctions.freqItems（）是别名。

注意

此功能用于探索性数据分析，因为我们不保证生成的DataFrame的模式的向后兼容性。

|  |  |
| --- | --- |
| Parameters: | cols – 用于计算频繁项目作为列表或字符串元组的列的名称。  support – 考虑一个项目“频繁”的频率。 默认值为1％。 支撑必须大于1e-4。 |

New in version 1.4.

**sampleBy(col, fractions, seed=None)**

**基于每个层上给出的分数，返回没有替换的分层样本。**

|  |  |
| --- | --- |
| Parameters: | col – 定义层的列  fractions –每层采样分数。 如果没有指定层，我们将其分数视为零。  seed – random seed |
| Returns: | 一个新的DataFrame，代表分层样本 |

>>> from pyspark.sql.functions import col

>>> dataset = sqlContext.range(0, 100).select((col("id") % 3).alias("key"))

>>> sampled = dataset.sampleBy("key", fractions={0: 0.1, 1: 0.2}, seed=0)

>>> sampled.groupBy("key").count().orderBy("key").show()

+---+-----+

|key|count|

+---+-----+

| 0| 5|

| 1| 9|

+---+-----+

New in version 1.5.

class pyspark.sql.Window

在数据帧中定义窗口的实用功能。

例如：

>>> # ORDER BY date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

>>> window = Window.orderBy("date").rowsBetween(Window.unboundedPreceding, Window.currentRow)

>>> # PARTITION BY country ORDER BY date RANGE BETWEEN 3 PRECEDING AND 3 FOLLOWING

>>> window = Window.orderBy("date").partitionBy("country").rangeBetween(-3, 3)

Note

在实验中

New in version 1.4.

**currentRow = 0**

**static orderBy(\*cols)**

用定义的顺序创建一个WindowSpec。

版本1.4中的新功能。

**static partitionBy(\*cols)**

创建一个WindowSpec，并定义分区。

版本1.4中的新功能。

**static rangeBetween(start, end)**

创建一个WindowSpec，从起始（包括）到结束（包括）定义了框架边界。

开始和结束都是与当前行相对的。 例如，“0”表示“当前行”，而“-1”表示当前行之前的一个，“5”表示当前行之后的五个。

我们建议用户使用Window.unboundedPreceding，Window.unboundedFollowing和Window.currentRow来指定特殊边界值，而不是直接使用积分值。

|  |  |
| --- | --- |
| Parameters: | start – 边界开始，包括。 如果这是Window.unboundedPreceding，或任何小于或等于max（-sys.maxsize，-9223372036854775808）的值，则该框架是无界的。  end –边界，包括边界。 如果这是Window.unboundedFollowing，或任何大于或等于min（sys.maxsize，9223372036854775807）的值，则该框架是无界的。 |

New in version 2.1.

**static rowsBetween(start, end)**

创建一个WindowSpec，从起始（包括）到结束（包括）定义了框架边界。

开始和结束都是当前行的相对位置。 例如，“0”表示“当前行”，而“-1”表示当前行之前的行，“5”表示当前行之后的第五行。

我们建议用户使用Window.unboundedPreceding，Window.unboundedFollowing和Window.currentRow来指定特殊边界值，而不是直接使用积分值

|  |  |
| --- | --- |
| Parameters: | start –边界开始，包括。 如果这是Window.unboundedPreceding，或者小于或等于-9223372036854775808的任何值，则该框架是无界的。  end – 边界，包括边界。 如果这是Window.unboundedFollowing，或任何大于或等于9223372036854775807的值，该框架是无界的。 |

New in version 2.1.

unboundedFollowing = 9223372036854775807L

unboundedPreceding = -9223372036854775808L

# WindowSpec

**class pyspark.sql.WindowSpec(jspec)**

定义分区，排序和帧边界的窗口规范。

在Window中使用静态方法创建一个WindowSpec。

注意

试验

版本1.4中的新功能。

**orderBy(\*cols)**

定义WindowSpec中的排序列。

|  |  |
| --- | --- |
| Parameters: | cols –列或表达式的名称 |

New in version 1.4.

**partitionBy(\*cols)**

定义WindowSpec中的分区列。

|  |  |
| --- | --- |
| Parameters: | cols –列或表达式的名称 |

New in version 1.4.

**rangeBetween(start, end)**

定义从起始（包括）到结束（包括）的帧边界。

开始和结束都是与当前行相对的。 例如，“0”表示“当前行”，而“-1”表示当前行之前的一个，“5”表示当前行之后的五个。

**我们建议用户使用Window.unbounded Preceding，Window.unbounded Following和Window.currentRow来指定特殊边界值，而不是直接使用积分值。**

|  |  |
| --- | --- |
| Parameters: | start – 边界开始，包括。 如果这是Window.unboundedPreceding，或任何小于或等于max（-sys.maxsize，-9223372036854775808）的值，则该框架是无界的。  end –边界，包括边界。 如果这是Window.unboundedFollowing，或任何大于或等于min（sys.maxsize，9223372036854775807）的值，则该框架是无界的。 |

New in version 1.4.

**rowsBetween(start, end)**

定义从起始（包括）到结束（包括）的帧边界。

开始和结束都是当前行的相对位置。 例如，“0”表示“当前行”，而“-1”表示当前行之前的行，“5”表示当前行之后的第五行。

我们建议用户使用Window.unboundedPreceding，Window.unboundedFollowing和Window.currentRow来指定特殊边界值，而不是直接使用积分值。

|  |  |
| --- | --- |
| Parameters: | start – 边界开始，包括。 如果这是Window.unboundedPreceding，或任何小于或等于max（-sys.maxsize，-9223372036854775808）的值，则该框架是无界的。  end – 边界，包括边界。 如果这是Window.unboundedFollowing，或任何大于或等于min（sys.maxsize，9223372036854775807）的值，则该框架是无界的。 |

New in version 1.4.

# DataFrameReader

**class pyspark.sql.DataFrameReader(spark)**

用于从外部存储系统（例如文件系统，键值存储等）加载DataFrame的接口。 使用spark.read（）来访问它。

版本1.4中的新功能。

**csv(path, schema=None, sep=None, encoding=None, quote=None, escape=None, comment=None, header=None, inferSchema=None, ignoreLeadingWhiteSpace=None, ignoreTrailingWhiteSpace=None, nullValue=None, nanValue=None, positiveInf=None, negativeInf=None, dateFormat=None, timestampFormat=None, maxColumns=None, maxCharsPerColumn=None, maxMalformedLogPerPartition=None, mode=None)**

加载CSV文件并将结果作为DataFrame返回。

如果inferSchema被启用，此功能将通过输入一次来确定输入模式。 为避免遍历整个数据一次，请禁用inferSchema选项或使用模式显式指定模式。

|  |  |
| --- | --- |
| Parameters: | path – string, or list of strings, for input path(s).  schema –输入模式的可选pyspark.sql.types.StructType。  sep – 将单个字符设置为每个字段和值的分隔符。 如果设置为None，它将使用默认值，。  encoding –按照给定的编码类型对CSV文件进行解码。 如果设置为None，则使用默认值UTF-8。  quote – 设置用于转义引用值的单个字符，其中分隔符可以作为值的一部分。 如果设置为None，则使用默认值“。如果要关闭报价，则需要设置一个空字符串。  escape –. 将用于转义引号的单个字符设置为已引用的值。 如果设置为None，它将使用默认值\  comment – 设置用于跳过以此字符开头的行的单个字符。 默认情况下（无），它被禁用。标题 - 使用第一行作为列的名称。 如果设置为None，它将使用默认值false。  inferSchema – 从数据自动推断输入模式。 它需要一个额外的数据传递。 如果设置为None，它将使用默认值false。  ignoreLeadingWhiteSpace – 定义是否应该跳过正在读取的值的引导空格。 如果设置为None，它将使用默认值false。  ignoreTrailingWhiteSpace – 定义是否应跳过正在读取的值的尾随空格。 如果设置为None，它将使用默认值false。  nullValue – 设置一个空值的字符串表示形式。 如果设置为None，它将使用默认值空字符串。 自2.0.1起，此null值参数适用于所有支持的类型，包括字符串类型。  nanValue –**设置非数字值的字符串表示形式。 如果设置为None，则使用默认值NaN。**  positiveInf –**设置正无穷大值的字符串表示。 如果设置为None，它将使用默认值Inf。**  negativeInf **–****设置负无穷大值的字符串表示。 如果设置为None，它将使用默认值Inf。**  dateFormat –设置表示日期格式的字符串。 自定义日期格式遵循java.text.SimpleDateFormat的格式。 这适用于日期类型。 如果设置为None，则**使用默认值yyyy-MM-dd**。  timestampFormat – 设置指示时间戳格式的字符串。 自定义日期格式遵循java.text.SimpleDateFormat的格式。 这适用于时间戳类型。 如果设置为None，则使用默认值yyyy-MM-dd'T'HH：mm：ss.SSSZZ。  maxColumns – 定义了一条记录可以拥有多少列的硬限制。 如果设置为None，它将使用默认值20480。  maxCharsPerColumn – 定义读取任何给定值允许的最大字符数。 如果设置为None，则使用默认值，-1表示无限长度。  maxMalformedLogPerPartition – 设置Spark将为每个分区记录的最大不正确的行数。 超出这个数字的畸形记录将被忽略。 如果设置为None，则使用默认值10。  mode –  允许在解析期间处理损坏记录的模式。 如果没有  设置，它使用默认值PERMISSIVE。  PERMISSIVE : 将其他字段设置为null，当它遇到损坏的记录。  当用户设置模式时，它为额外的字段设置为空。  DROPMALFORMED : 忽略整个损坏的记录。  FAILFAST :遇到损坏的记录时会引发异常。 |

>>> df = spark.read.csv('python/test\_support/sql/ages.csv')

>>> df.dtypes

[('\_c0', 'string'), ('\_c1', 'string')]

New in version 2.0.

**format(source)**

指定输入数据源格式。

|  |  |
| --- | --- |
| Parameters: | source – 字符串，数据源的名称，例如。 'json'，'parquet'。 |

>>> df = spark.read.format('json').load('python/test\_support/sql/people.json')

>>> df.dtypes

[('age', 'bigint'), ('name', 'string')]

New in version 1.4.

**jdbc(url, table, column=None, lowerBound=None, upperBound=None, numPartitions=None, predicates=None, properties=None)**

构造一个DataFrame，表示通过JDBC URL url和连接属性访问的名为表的数据库表。

如果指定了列或谓词，则表的分区将被并行检索。

如果指定了列和谓词，将使用列。

注意

不要在大型集群上并行创建太多分区; 否则Spark可能会崩溃您的外部数据库系统。

|  |  |
| --- | --- |
| Parameters: | url – a JDBC URL of the form jdbc:subprotocol:subname  table – the name of the table  column – 将用于分区的整数列的名称; 如果指定了此参数，则numPartitions，lowerBound（包括）和upperBound（exclusive）将为生成的WHERE子句表达式构建分区步骤，用于将列列均匀分解  lowerBound –用于决定分区步幅的列的最小值  upperBound – 用于决定分区步幅的列的最大值  numPartitions – 分区个数  predicates – 适用于WHERE子句的表达式列表; 每个定义DataFrame的一个分区  properties – JDBC数据库连接参数的字典。 通常至少有属性“user”和“password”及其相应的值。 例如{'user'：'SYSTEM'，'password'：'mypassword'} |
| Returns: | a DataFrame |

New in version 1.4.

**json(path, schema=None, primitivesAsString=None, prefersDecimal=None, allowComments=None, allowUnquotedFieldNames=None, allowSingleQuotes=None, allowNumericLeadingZero=None, allowBackslashEscapingAnyCharacter=None, mode=None, columnNameOfCorruptRecord=None, dateFormat=None, timestampFormat=None)**

加载JSON文件（JSON线文本格式或换行符分隔的JSON）或存储JSON对象的字符串RDD（每个记录一个对象），并将结果作为一个类#DataFrame返回。

如果未指定模式参数，则此函数会经过输入一次以确定输入模式。

|  |  |
| --- | --- |
| Parameters: | path – 字符串表示JSON数据集的路径，或者表示存储JSON对象的字符串的RDD。  schema – 输入模式的可选pyspark.sql.types.StructType。  primitivesAsString – 将所有原始值推断为字符串类型。 如果设置为None，它将使用默认值false。  prefersDecimal – 将所有浮点值推定为十进制类型。 如果值不符合十进制，则将它们推定为双精度。 如果设置为None，它将使用默认值false。  allowComments – 忽略JSON记录中的Java / C ++风格注释。 如果设置为None，它将使用默认值false。  allowUnquotedFieldNames – 允许无引号的JSON字段名称。 如果设置为None，它将使用默认值false。  allowSingleQuotes – 允许单引号除双引号之外。 如果设置为None，它将使用默认值true。  allowNumericLeadingZero –允许数字前导零（例如00012）。 如果设置为None，它将使用默认值false。  allowBackslashEscapingAnyCharacter –  mode – 允许使用反斜杠引用机制接受所有字符的引用。 如果设置为None，它将使用默认值false。  允许在解析期间处理损坏记录的模式。 如果没有  设置，它使用默认值PERMISSIVE。  PERMISSIVE :将其他字段设置为null，当它遇到损坏的记录，并将格式错误的字符串放入由columnNameOfCorruptRecord配置的新字段。 当用户设置模式时，它为额外的字段设置为空。  DROPMALFORMED：忽略整个损坏的记录。  FAILFAST：遇到损坏的记录时抛出异常。  columnNameOfCorruptRecord –允许重命名具有由PERMISSIVE模式创建的畸形字符串的新字段。 这将覆盖spark.sql.columnNameOfCorruptRecord。 如果设置为None，它将使用spark.sql.columnNameOfCorruptRecord中指定的值。  dateFormat –设置表示日期格式的字符串。 自定义日期格式遵循java.text.SimpleDateFormat的格式。 这适用于日期类型。 如果设置为None，则使用默认值yyyy-MM-dd。  timestampFormat – 设置指示时间戳格式的字符串。 自定义日期格式遵循java.text.SimpleDateFormat的格式。 这适用于时间戳类型。 如果设置为None，则使用默认值yyyy-MM-dd'T'HH：mm：ss.SSSZZ。 |

>>> df1 = spark.read.json('python/test\_support/sql/people.json')

>>> df1.dtypes

[('age', 'bigint'), ('name', 'string')]

>>> rdd = sc.textFile('python/test\_support/sql/people.json')

>>> df2 = spark.read.json(rdd)

>>> df2.dtypes

[('age', 'bigint'), ('name', 'string')]

New in version 1.4.

**load(path=None, format=None, schema=None, \*\*options)**

从数据源加载数据，并将其作为：class`DataFrame返回。

|  |  |
| --- | --- |
| Parameters: | path – 可选字符串或文件系统支持的数据源的字符串列表。  format – 可选字符串，用于数据源的格式。 默认为“parquet”。  schema – 输入模式的可选pyspark.sql.types.StructType。  options – 所有其他字符串选项 |

>>> df = spark.read.load('python/test\_support/sql/parquet\_partitioned', opt1=True,

... opt2=1, opt3='str')

>>> df.dtypes

[('name', 'string'), ('year', 'int'), ('month', 'int'), ('day', 'int')]

>>> df = spark.read.format('json').load(['python/test\_support/sql/people.json',

... 'python/test\_support/sql/people1.json'])

>>> df.dtypes

[('age', 'bigint'), ('aka', 'string'), ('name', 'string')]

New in version 1.4.

**option(key, value)**

**为底层数据源添加输入选项。**

1.5版新功能

**options(\*\*options)**

**添加底层数据源的输入选项。**

版本1.4中的新功能。

**orc(path)**

**加载ORC文件，将结果作为DataFrame返回。**

Note

目前，ORC支持仅与Hive支持一起使用。

>>> df = spark.read.orc('python/test\_support/sql/orc\_partitioned')

>>> df.dtypes

[('a', 'bigint'), ('b', 'int'), ('c', 'int')]

New in version 1.5.

**parquet(\*paths)**

**加载Parquet文件，将结果作为DataFrame返回。**

您可以设置以下Parquet特定选项来阅读Parquet文件：

mergeSchema：设置是否合并从所有Parquet部件文件中收集的模式。 这将覆盖spark.sql.parquet.mergeSchema。 默认值在spark.sql.parquet.mergeSchema中指定。

>>> df = spark.read.parquet('python/test\_support/sql/parquet\_partitioned')

>>> df.dtypes

[('name', 'string'), ('year', 'int'), ('month', 'int'), ('day', 'int')]

New in version 1.4.

**schema(schema)**

指定输入模式。

一些数据源（例如JSON）可以从数据自动推断输入模式。 通过在此处指定模式，底层数据源可以跳过模式推理步骤，从而加快数据加载速度。

|  |  |
| --- | --- |
| Parameters: | schema – a [pyspark.sql.types.StructType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.StructType) object |

New in version 1.4.

**table(tableName)**

作为DataFrame返回指定的表。

|  |  |
| --- | --- |
| Parameters: | tableName – string, name of the table. |

>>> df = spark.read.parquet('python/test\_support/sql/parquet\_partitioned')

>>> df.createOrReplaceTempView('tmpTable')

>>> spark.read.table('tmpTable').dtypes

[('name', 'string'), ('year', 'int'), ('month', 'int'), ('day', 'int')]

New in version 1.4.

**text(paths)**

**加载文本文件并返回一个DataFrame，其架构以名为“value”的字符串列开始，如果有的话，则返回分区列。**

文本文件中的每一行都是生成的DataFrame中的一行。

|  |  |
| --- | --- |
| Parameters: | paths – string, or list of strings, for input path(s). |

>>> df = spark.read.text('python/test\_support/sql/text-test.txt')

>>> df.collect()

[Row(value=u'hello'), Row(value=u'this')]

New in version 1.6.

# DataFrameWriter

**class pyspark.sql.DataFrameWriter(df)**

用于将DataFrame写入外部存储系统（例如文件系统，键值存储等）的接口。 使用DataFrame.write（）来访问它。

版本1.4中的新功能。

**csv(path, mode=None, compression=None, sep=None, quote=None, escape=None, header=None, nullValue=None, escapeQuotes=None, quoteAll=None, dateFormat=None, timestampFormat=None)**

Saves the content of the [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) in CSV format at the specified path.

|  |  |
| --- | --- |
| Parameters: | path – 任何Hadoop支持的文件系统中的路径  mode –  指定数据已存在时的保存操作的行为。  append：将此DataFrame的内容附加到现有数据。  覆盖：覆盖现有数据。  ignore：如果数据已经存在，则静默地忽略此操作。  错误（默认情况）：如果数据已存在，则抛出异常。  compression – 压缩编解码器在保存到文件时使用。 这可以是已知的不区分大小写的缩写名称之一（none，bzip2，gzip，lz4，snappy和deflate）。  sep –将单个字符设置为每个字段和值的分隔符。 如果设置为None，它将使用默认值，。  quote –设置用于转义引用值的单个字符，其中分隔符可以作为值的一部分。 如果设置为None，则使用默认值“。如果要关闭报价，则需要设置一个空字符串。  escape – 将用于转义引号的单个字符设置为已引用的值。 如果设置为None，它将使用默认值\  escapeQuotes – 一个标志，指示包含引号的值是否应该始终用引号括起来。 如果设置为None，它将使用默认值true，转义包含引号字符的所有值。  quoteAll –一个标志，表示所有值是否应该始终用引号括起来。 如果设置为None，它将使用默认值false，只转义包含引号字符的值。  header – 将列的名称写入第一行。 如果设置为None，它将使用默认值false。  nullValue – 设置一个空值的字符串表示形式。 如果设置为None，它将使用默认值空字符串。  dateFormat –设置表示日期格式的字符串。 自定义日期格式遵循格式 java.text.SimpleDateFormat. 这适用于日期类型。 如果设置为None，则使用默认值yyyy-MM-dd。  timestampFormat – 设置指示时间戳格式的字符串。 自定义日期格式遵循java.text.SimpleDateFormat的格式。 这适用于时间戳类型。 如果设置为None，则使用默认值yyyy-MM-dd'T'HH：mm：ss.SSSZZ。 |

>>> df.write.csv(os.path.join(tempfile.mkdtemp(), 'data'))

New in version 2.0.

**format(source)**

指定底层输出数据源。

|  |  |
| --- | --- |
| Parameters: | source – string, name of the data source, e.g. ‘json’, ‘parquet’. |

>>> df.write.format('json').save(os.path.join(tempfile.mkdtemp(), 'data'))

New in version 1.4.

**insertInto(tableName, overwrite=False)**

将DataFrame的内容插入指定的表。

它要求类的模式：DataFrame与表的模式相同。

可以覆盖任何现有数据。

版本1.4中的新功能。

**jdbc(url, table, mode=None, properties=None)**

通过JDBC将DataFrame的内容保存到外部数据库表。

注意

不要在大型集群上并行创建太多分区; 否则Spark可能会崩溃您的外部数据库系统。

|  |  |
| --- | --- |
| Parameters: | url – a JDBC URL of the form jdbc:subprotocol:subname  table –外部数据库中的表的名称。  mode –  指定数据已存在时的保存操作的行为。  append：将此DataFrame的内容附加到现有数据。  覆盖：覆盖现有数据。  ignore：如果数据已经存在，则静默地忽略此操作。  error（默认情况）：如果数据已存在，则抛出异常。  properties –JDBC数据库连接参数的字典。 通常至少有属性“user”和“password”及其相应的值。 例如{'user'：'SYSTEM'，'password'：'mypassword'} |

New in version 1.4.

**json(path, mode=None, compression=None, dateFormat=None, timestampFormat=None)**

以指定路径的JSON格式保存DataFrame的内容。

|  |  |
| --- | --- |
| Parameters: | path – the path in any Hadoop supported file system  mode –  指定数据已存在时的保存操作的行为。  append: 将此DataFrame的内容附加到现有数据。  overwrite: 覆盖原始数据  ignore:如果数据已经存在，请静默忽略此操作。  error (default case): 如果数据已经存在，则抛出异常。  compression – 压缩编解码器在保存到文件时使用。 这可以是已知的不区分大小写的缩写名称之一（none，bzip2，gzip，lz4，snappy和deflate）。  dateFormat – 设置表示日期格式的字符串。 自定义日期格式遵循java.text.SimpleDateFormat的格式。 这适用于日期类型。 如果设置为None，则使用默认值yyyy-MM-dd。  timestampFormat –设置指示时间戳格式的字符串。 自定义日期格式遵循java.text.SimpleDateFormat的格式。 这适用于时间戳类型。 如果设置为None，则使用默认值yyyy-MM-dd'T'HH：mm：ss.SSSZZ。 |

>>> df.write.json(os.path.join(tempfile.mkdtemp(), 'data'))

New in version 1.4.

**mode(saveMode)**

指定数据或表已存在时的行为。

Options include:

**append:** **将此DataFrame的内容附加到现有数据。**

**overwrite: 覆盖已有数据.**

**error: 若数据存在则抛出异常**

**ignore:** **如果数据已经存在，请静默忽略此操作。**

>>> df.write.mode('append').parquet(os.path.join(tempfile.mkdtemp(), 'data'))

New in version 1.4.

**option(key, value)**

为底层数据源添加一个输出选项。

New in version 1.5.

**options(\*\*options)**

添加底层数据源的输出选项。

New in version 1.4.

**orc(path, mode=None, partitionBy=None, compression=None)**

以指定路径的ORC格式保存DataFrame的内容。

注意

Currently ORC support is only available together with Hive support.

|  |  |
| --- | --- |
| Parameters: | path – the path in any Hadoop supported file system  mode –  specifies the behavior of the save operation when data already exists.  append: Append contents of this [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) to existing data.  overwrite: Overwrite existing data.  ignore: Silently ignore this operation if data already exists.  error (default case): Throw an exception if data already exists.  partitionBy – 分区列的名称  compression – 压缩编解码器在保存到文件时使用。 这可以是已知的不区分大小写的缩写名称之一（none，snappy，zlib和lzo）。 这将覆盖orc.compress。 如果设置为None，它将使用默认值snappy。 |

>>> orc\_df = spark.read.orc('python/test\_support/sql/orc\_partitioned')

>>> orc\_df.write.orc(os.path.join(tempfile.mkdtemp(), 'data'))

New in version 1.5.

**parquet(path, mode=None, partitionBy=None, compression=None)**

**以指定路径的Parquet格式保存DataFrame的内容。**

|  |  |
| --- | --- |
| Parameters: | path – the path in any Hadoop supported file system  mode –  specifies the behavior of the save operation when data already exists.  append: Append contents of this [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) to existing data.  overwrite: Overwrite existing data.  ignore: Silently ignore this operation if data already exists.  error (default case): Throw an exception if data already exists.  partitionBy – names of partitioning columns  compression – 压缩编解码器在保存到文件时使用。 这可以是已知的不区分大小写的缩写名称之一（none，snappy，gzip和lzo）。 这将覆盖spark.sql.parquet.compression.codec。 如果设置为None，它将使用spark.sql.parquet.compression.codec中指定的值。 |

>>> df.write.parquet(os.path.join(tempfile.mkdtemp(), 'data'))

New in version 1.4.

**partitionBy(\*cols)**

按照文件系统上给定列的输出进行分区。

如果指定，输出将放置在文件系统上，类似于Hive的分区方案。

|  |  |
| --- | --- |
| Parameters: | cols – name of columns |

>>> df.write.partitionBy('year', 'month').parquet(os.path.join(tempfile.mkdtemp(), 'data'))

New in version 1.4.

**save(path=None, format=None, mode=None, partitionBy=None, \*\*options)**

将DataFrame的内容保存到数据源。

数据源由格式和一组选项指定。 如果未指定格式，将使用由spark.sql.sources.default配置的默认数据源。

|  |  |
| --- | --- |
| Parameters: | path – the path in a Hadoop supported file system  format – the format used to save  mode –  specifies the behavior of the save operation when data already exists.  append: Append contents of this [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) to existing data.  overwrite: Overwrite existing data.  ignore: Silently ignore this operation if data already exists.  error (default case): Throw an exception if data already exists.  partitionBy – names of partitioning columns  options – all other string options |

>>> df.write.mode('append').parquet(os.path.join(tempfile.mkdtemp(), 'data'))

New in version 1.4.

**saveAsTable(name, format=None, mode=None, partitionBy=None, \*\*options)**

将DataFrame的内容保存为指定的表。

在表已经存在的情况下，此功能的行为取决于由mode函数指定的保存模式（默认为抛出异常）。 当模式为覆盖时，DataFrame的模式不需要与现有表的模式相同。

append: Append contents of this [DataFrame](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) to existing data.

overwrite: Overwrite existing data.

error: Throw an exception if data already exists.

ignore: Silently ignore this operation if data already exists.

|  |  |
| --- | --- |
| Parameters: | name – the table name  format – the format used to save  mode – one of append, overwrite, error, ignore (default: error)  partitionBy – names of partitioning columns  options – all other string options |

New in version 1.4.

**text(path, compression=None)**

将DataFrame的内容保存在指定路径的文本文件中。

|  |  |
| --- | --- |
| Parameters: | path – the path in any Hadoop supported file system  compression – 压缩编解码器在保存到文件时使用。 这可以是已知的不区分大小写的缩写名称之一（none，bzip2，gzip，lz4，snappy和deflate）。 |

DataFrame必须只有一个字符串类型的列。 每行在输出文件中成为新行。

新版本1.6。

pyspark.sql.types module

# DataType

**class pyspark.sql.types.DataType**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DataType)**数据类型的基类。**

**fromInternal(obj)**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DataType.fromInternal)

将内部SQL对象转换为本机Python对象。

**json()**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DataType.json)

**jsonValue()**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DataType.jsonValue)

**needConversion()**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DataType.needConversion)

这种类型需要在Python对象和内部SQL对象之间进行转换。

这用于避免对ArrayType / MapType / StructType进行不必要的转换。

**simpleString()**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DataType.simpleString)

**toInternal(obj)**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DataType.toInternal)

**将Python对象转换为内部SQL对象。**

**classmethod typeName()**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DataType.typeName)

class pyspark.sql.types.**NullType**[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#NullType)

Null type.

The data type representing None, used for the types that cannot be inferred.

class pyspark.sql.types.**StringType**[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StringType)

**String** data type.

class pyspark.sql.types.**BinaryType**[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#BinaryType)

**Binary** (byte array) data type.

class pyspark.sql.types.**BooleanType**[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#BooleanType)

**Boolean** data type.

class pyspark.sql.types.**DateType**[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DateType)

**Date** (datetime.date) data type.

EPOCH\_ORDINAL = 719163

fromInternal(v)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DateType.fromInternal)

needConversion()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DateType.needConversion)

toInternal(d)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DateType.toInternal)

class pyspark.sql.types.TimestampType[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#TimestampType)

Timestamp (datetime.datetime) data type.

fromInternal(ts)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#TimestampType.fromInternal)

needConversion()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#TimestampType.needConversion)

toInternal(dt)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#TimestampType.toInternal)

# DecimalType

**class pyspark.sql.types.DecimalType(precision=10, scale=0)**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DecimalType)

十进制（decimal.Decimal）数据类型。

DecimalType必须具有固定精度（最大总数）和scale（点右侧的位数）。 例如，（5,2）可以支持[-999.99到999.99]的值。

精度可达38度，刻度必须小于或等于精度。

创建DecimalType时，默认的精度和比例为（10，0）。 当从decimal.Decimal对象推断模式时，它将是DecimalType（38,18）。

|  |  |
| --- | --- |
| Parameters: | precision – 最大总数（默认值：10）  scale –点右侧的位数。 （默认值：0） |

**jsonValue()**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DecimalType.jsonValue)

**simpleString()**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DecimalType.simpleString)

**class pyspark.sql.types.DoubleType**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#DoubleType)

**双数据类型，代表双精度浮点数。**

**class pyspark.sql.types.FloatType**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#FloatType)

**浮点数据类型，表示单精度浮点数。**

class pyspark.sql.types.ByteType[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#ByteType)

Byte data type, i.e. a signed integer in a single byte.

simpleString()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#ByteType.simpleString)

class pyspark.sql.types.IntegerType[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#IntegerType)

Int data type, i.e. a signed 32-bit integer.

simpleString()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#IntegerType.simpleString)

class pyspark.sql.types.LongType[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#LongType)

Long data type, i.e. a signed 64-bit integer.

If the values are beyond the range of [-9223372036854775808, 9223372036854775807], please use [DecimalType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.DecimalType).

simpleString()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#LongType.simpleString)

# **ShortType**

class pyspark.sql.types.**ShortType**[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#ShortType)

Short data type, i.e. a signed 16-bit integer.

simpleString()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#ShortType.simpleString)

class pyspark.sql.types.ArrayType(elementType, containsNull=True)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#ArrayType)

Array data type.

|  |  |
| --- | --- |
| Parameters: | elementType – [DataType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.DataType) of each element in the array.  containsNull – boolean, whether the array can contain null (None) values. |

fromInternal(obj)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#ArrayType.fromInternal)

classmethod fromJson(json)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#ArrayType.fromJson)

jsonValue()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#ArrayType.jsonValue)

needConversion()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#ArrayType.needConversion)

simpleString()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#ArrayType.simpleString)

toInternal(obj)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#ArrayType.toInternal)

class pyspark.sql.types.MapType(keyType, valueType, valueContainsNull=True)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#MapType)

Map data type.

|  |  |
| --- | --- |
| Parameters: | keyType – [DataType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.DataType) of the keys in the map.  valueType – [DataType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.DataType) of the values in the map.  valueContainsNull – indicates whether values can contain null (None) values. |

Keys in a map data type are not allowed to be null (None).

fromInternal(obj)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#MapType.fromInternal)

classmethod fromJson(json)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#MapType.fromJson)

jsonValue()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#MapType.jsonValue)

needConversion()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#MapType.needConversion)

simpleString()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#MapType.simpleString)

toInternal(obj)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#MapType.toInternal)

class pyspark.sql.types.StructField(name, dataType, nullable=True, metadata=None)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructField)

A field in [StructType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.StructType).

|  |  |
| --- | --- |
| Parameters: | name – string, name of the field.  dataType – [DataType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.DataType) of the field.  nullable – boolean, whether the field can be null (None) or not.  metadata – a dict from string to simple type that can be toInternald to JSON automatically |

fromInternal(obj)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructField.fromInternal)

classmethod fromJson(json)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructField.fromJson)

jsonValue()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructField.jsonValue)

needConversion()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructField.needConversion)

simpleString()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructField.simpleString)

toInternal(obj)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructField.toInternal)

**class pyspark.sql.types.StructType(fields=None)**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructType)

Struct type, consisting of a list of [StructField](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.StructField).

This is the data type representing a Row.

Iterating a [StructType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.StructType) will iterate its StructField`s. A contained :class:`StructField can be accessed by name or position.

>>> struct1 = StructType([StructField("f1", StringType(), True)])

>>> struct1["f1"]

StructField(f1,StringType,true)

>>> struct1[0]

StructField(f1,StringType,true)

**add(field, data\_type=None, nullable=True, metadata=None)**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructType.add)

Construct a StructType by adding new elements to it to define the schema. The method accepts either:

A single parameter which is a StructField object.

Between 2 and 4 parameters as (name, data\_type, nullable (optional), metadata(optional). The data\_type parameter may be either a String or a DataType object.

>>> struct1 = StructType().add("f1", StringType(), True).add("f2", StringType(), True, None)

>>> struct2 = StructType([StructField("f1", StringType(), True), \

... StructField("f2", StringType(), True, None)])

>>> struct1 == struct2

True

>>> struct1 = StructType().add(StructField("f1", StringType(), True))

>>> struct2 = StructType([StructField("f1", StringType(), True)])

>>> struct1 == struct2

True

>>> struct1 = StructType().add("f1", "string", True)

>>> struct2 = StructType([StructField("f1", StringType(), True)])

>>> struct1 == struct2

True

|  |  |
| --- | --- |
| Parameters: | field – Either the name of the field or a StructField object  data\_type – If present, the DataType of the StructField to create  nullable – Whether the field to add should be nullable (default True)  metadata – Any additional metadata (default None) |
| Returns: | a new updated StructType |

**fromInternal(obj)**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructType.fromInternal)

classmethod fromJson(json)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructType.fromJson)

jsonValue()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructType.jsonValue)

needConversion()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructType.needConversion)

simpleString()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructType.simpleString)

toInternal(obj)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/types.html#StructType.toInternal)

pyspark.sql.functions module

A collections of builtin functions

pyspark.sql.functions.abs(col)

Computes the absolute value.

New in version 1.3.

pyspark.sql.functions.acos(col)

Computes the cosine inverse of the given value; the returned angle is in the range0.0 through pi.

# 数学函数

New in version 1.4.

pyspark.sql.functions.add\_months(start, months)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#add_months)

Returns the date that is months months after start

>>> df = spark.createDataFrame([('2015-04-08',)], ['d'])

>>> df.select(add\_months(df.d, 1).alias('d')).collect()

[Row(d=datetime.date(2015, 5, 8))]

New in version 1.5.

pyspark.sql.functions.approxCountDistinct(col, rsd=None)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#approxCountDistinct)

Note

Deprecated in 2.1, use approx\_count\_distinct instead.

New in version 1.3.

pyspark.sql.functions.approx\_count\_distinct(col, rsd=None)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#approx_count_distinct)

Returns a new Column for approximate distinct count of col.

>>> df.agg(approx\_count\_distinct(df.age).alias('c')).collect()

[Row(c=2)]

New in version 2.1.

pyspark.sql.functions.array(\*cols)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#array)

创建一个新的数组列。

|  |  |
| --- | --- |
| Parameters: | cols – list of column names (string) or list of Column expressions that have the same data type. |

>>> df.select(array('age', 'age').alias("arr")).collect()

[Row(arr=[2, 2]), Row(arr=[5, 5])]

>>> df.select(array([df.age, df.age]).alias("arr")).collect()

[Row(arr=[2, 2]), Row(arr=[5, 5])]

New in version 1.4.

pyspark.sql.functions.array\_contains(col, value)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#array_contains)

集合函数：如果数组包含给定值，则返回True。 集合元素和值必须是相同的类型。

|  |  |
| --- | --- |
| Parameters: | col – name of column containing array  value – value to check for in array |

>>> df = spark.createDataFrame([(["a", "b", "c"],), ([],)], ['data'])

>>> df.select(array\_contains(df.data, "a")).collect()

[Row(array\_contains(data, a)=True), Row(array\_contains(data, a)=False)]

New in version 1.5.

pyspark.sql.functions.asc(col)

根据给定列名称的升序返回排序表达式。

1.3版新功能

pyspark.sql.functions.ascii(col)

计算字符串列的第一个字符的数值。

New in version 1.5.

pyspark.sql.functions.asin(col)

计算给定值的正弦逆; 返回的角度在-pi / 2到pi / 2的范围内。

New in version 1.4.

pyspark.sql.functions.atan(col)

Computes the tangent inverse of the given value.

New in version 1.4.

**pyspark.sql.functions.atan2(col1, col2)**

从直角坐标（x，y）的极坐标（r，theta）的转换返回角度θ。

New in version 1.4.

pyspark.sql.functions.avg(col)

聚合函数：返回组中的值的平均值。

1.3版新功能

pyspark.sql.functions.base64(col)

计算二进制列的BASE64编码，并将其作为字符串列返回。

1.5版新功能

pyspark.sql.functions.bin(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#bin)

返回给定列的二进制值的字符串表示形式。

>>> df.select(bin(df.age).alias('c')).collect()

[Row(c=u'10'), Row(c=u'101')]

New in version 1.5.

pyspark.sql.functions.bitwiseNOT(col)

按位计算Not。

New in version 1.4.

pyspark.sql.functions.broadcast(df)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#broadcast)

标记一个DataFrame足够小，用于广播连接。

New in version 1.6.

pyspark.sql.functions.bround(col, scale=0)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#bround)

Round the given value to scale decimal places using HALF\_EVEN rounding mode if scale >= 0 or at integral part when scale < 0.

>>> spark.createDataFrame([(2.5,)], ['a']).select(bround('a', 0).alias('r')).collect()

[Row(r=2.0)]

New in version 2.0.

pyspark.sql.functions.cbrt(col)

Computes the cube-root of the given value.

New in version 1.4.

pyspark.sql.functions.ceil(col)

Computes the ceiling of the given value.

New in version 1.4.

pyspark.sql.functions.coalesce(\*cols)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#coalesce)

Returns the first column that is not null.

>>> cDf = spark.createDataFrame([(None, None), (1, None), (None, 2)], ("a", "b"))

>>> cDf.show()

+----+----+

| a| b|

+----+----+

|null|null|

| 1|null|

|null| 2|

+----+----+

>>> cDf.select(coalesce(cDf["a"], cDf["b"])).show()

+--------------+

|coalesce(a, b)|

+--------------+

| null|

| 1|

| 2|

+--------------+

>>> cDf.select('\*', coalesce(cDf["a"], lit(0.0))).show()

+----+----+----------------+

| a| b|coalesce(a, 0.0)|

+----+----+----------------+

|null|null| 0.0|

| 1|null| 1.0|

|null| 2| 0.0|

+----+----+----------------+

New in version 1.4.

pyspark.sql.functions.col(col)

Returns a Column based on the given column name.

New in version 1.3.

pyspark.sql.functions.collect\_list(col)

Aggregate function: returns a list of objects with duplicates.

New in version 1.6.

pyspark.sql.functions.collect\_set(col)

Aggregate function: returns a set of objects with duplicate elements eliminated.

New in version 1.6.

pyspark.sql.functions.column(col)

Returns a Column based on the given column name.

New in version 1.3.

pyspark.sql.functions.concat(\*cols)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#concat)

Concatenates multiple input string columns together into a single string column.

>>> df = spark.createDataFrame([('abcd','123')], ['s', 'd'])

>>> df.select(concat(df.s, df.d).alias('s')).collect()

[Row(s=u'abcd123')]

New in version 1.5.

pyspark.sql.functions.concat\_ws(sep, \*cols)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#concat_ws)

使用给定的分隔符将多个输入字符串列连接成一个字符串列。

>>> df = spark.createDataFrame([('abcd','123')], ['s', 'd'])

>>> df.select(concat\_ws('-', df.s, df.d).alias('s')).collect()

[Row(s=u'abcd-123')]

New in version 1.5.

pyspark.sql.functions.conv(col, fromBase, toBase)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#conv)

将字符串列中的数字从一个基数转换为另一个。

>>> df = spark.createDataFrame([("010101",)], ['n'])

>>> df.select(conv(df.n, 2, 16).alias('hex')).collect()

[Row(hex=u'15')]

New in version 1.5.

pyspark.sql.functions.corr(col1, col2)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#corr)

返回col1和col2的Pearson相关系数的新列。

>>> a = range(20)

>>> b = [2 \* x for x in range(20)]

>>> df = spark.createDataFrame(zip(a, b), ["a", "b"])

>>> df.agg(corr("a", "b").alias('c')).collect()

[Row(c=1.0)]

New in version 1.6.

**pyspark.sql.functions.cos(col)**

计算给定值的余弦值。

New in version 1.4.

pyspark.sql.functions.cosh(col)

计算给定值的双曲余弦值。

New in version 1.4.

pyspark.sql.functions.count(col)

聚合函数：返回组中的项目数。

New in version 1.3.

pyspark.sql.functions.countDistinct(col, \*cols)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#countDistinct)

返回一个不同的col或cols的列的新列。

>>> df.agg(countDistinct(df.age, df.name).alias('c')).collect()

[Row(c=2)]

>>> df.agg(countDistinct("age", "name").alias('c')).collect()

[Row(c=2)]

New in version 1.3.

pyspark.sql.functions.covar\_pop(col1, col2)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#covar_pop)

返回col1和col2的总体协方差的新列。

>>> a = [1] \* 10

>>> b = [1] \* 10

>>> df = spark.createDataFrame(zip(a, b), ["a", "b"])

>>> df.agg(covar\_pop("a", "b").alias('c')).collect()

[Row(c=0.0)]

New in version 2.0.

pyspark.sql.functions.covar\_samp(col1, col2)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#covar_samp)

返回col1和col2的样本协方差的新列。

>>> a = [1] \* 10

>>> b = [1] \* 10

>>> df = spark.createDataFrame(zip(a, b), ["a", "b"])

>>> df.agg(covar\_samp("a", "b").alias('c')).collect()

[Row(c=0.0)]

New in version 2.0.

pyspark.sql.functions.crc32(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#crc32)

计算二进制列的循环冗余校验值（CRC32），并将该值作为bigint返回。

>>> spark.createDataFrame([('ABC',)], ['a']).select(crc32('a').alias('crc32')).collect()

[Row(crc32=2743272264)]

New in version 1.5.

pyspark.sql.functions.create\_map(\*cols)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#create_map)

Creates a new map column.

|  |  |
| --- | --- |
| Parameters: | cols –列名称（字符串）或分组为键值对的列表达式的列表，例如。 （key1，value1，key2，value2，...）。 |

>>> df.select(create\_map('name', 'age').alias("map")).collect()

[Row(map={u'Alice': 2}), Row(map={u'Bob': 5})]

>>> df.select(create\_map([df.name, df.age]).alias("map")).collect()

[Row(map={u'Alice': 2}), Row(map={u'Bob': 5})]

New in version 2.0.

pyspark.sql.functions.cume\_dist()

窗口函数：返回窗口分区内的值的累积分布，即低于当前行的行的分数。

新版本1.6。

pyspark.sql.functions.current\_date()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#current_date)

返回当前日期作为日期列。

New in version 1.5.

pyspark.sql.functions.current\_timestamp()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#current_timestamp)

返回当前时间戳作为时间戳列。

pyspark.sql.functions.date\_add(start, days)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#date_add)

Returns the date that is days days after start

>>> df = spark.createDataFrame([('2015-04-08',)], ['d'])

>>> df.select(date\_add(df.d, 1).alias('d')).collect()

[Row(d=datetime.date(2015, 4, 9))]

New in version 1.5.

**pyspark.sql.functions.date\_format(date, format)**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#date_format)

将日期/时间戳/字符串转换为由第二个参数给定的日期格式指定的格式的字符串值。

一个模式可以是例如dd.MM.yyyy，并且可以返回一个字符串，如'18 .03.1993'。 可以使用Java类java.text.SimpleDateFormat的所有模式字母。

注意

使用时可能的专业功能像年。 这些都受益于专门的实施。

>>> df = spark.createDataFrame([('2015-04-08',)], ['a'])

>>> df.select(date\_format('a', 'MM/dd/yyy').alias('date')).collect()

[Row(date=u'04/08/2015')]

New in version 1.5.

pyspark.sql.functions.date\_sub(start, days)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#date_sub)

返回开始前天天的日期

>>> df = spark.createDataFrame([('2015-04-08',)], ['d'])

>>> df.select(date\_sub(df.d, 1).alias('d')).collect()

[Row(d=datetime.date(2015, 4, 7))]

New in version 1.5.

pyspark.sql.functions.datediff(end, start)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#datediff)

返回从头到尾的天数。

>>> df = spark.createDataFrame([('2015-04-08','2015-05-10')], ['d1', 'd2'])

>>> df.select(datediff(df.d2, df.d1).alias('diff')).collect()

[Row(diff=32)]

New in version 1.5.

pyspark.sql.functions.dayofmonth(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#dayofmonth)

将给定日期的月份中的日期提取为整数。

>>> df = spark.createDataFrame([('2015-04-08',)], ['a'])

>>> df.select(dayofmonth('a').alias('day')).collect()

[Row(day=8)]

New in version 1.5.

pyspark.sql.functions.dayofyear(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#dayofyear)

Extract the day of the year of a given date as integer.

>>> df = spark.createDataFrame([('2015-04-08',)], ['a'])

>>> df.select(dayofyear('a').alias('day')).collect()

[Row(day=98)]

New in version 1.5.

**pyspark.sql.functions.decode(col, charset)**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#decode)

使用提供的字符集（“US-ASCII”，“ISO-8859-1”，“UTF-8”，“UTF-16BE”，“UTF-16LE”之一）从二进制计算第一个参数，'UTF-16'）。

1.5版新功能

pyspark.sql.functions.degrees(col)

将以弧度测量的角度转换为以度为单位的近似等效角度。

2.1版新功能

pyspark.sql.functions.dense\_rank()

窗口函数：返回窗口分区中的行的排名，没有任何间隙。

Rank和dense\_rank之间的区别是，当有关系时，dense\_rank在排序顺序中没有留下空白。 也就是说，如果你是使用dense\_rank排名的比赛，并且有三个人排在第二位，那么你会说三个人都排在第二位，而下一个人排名第三。 排名将给我序列号，使得第三名的人（关系之后）将注册为第五名。

这相当于SQL中的DENSE\_RANK函数。

新版本1.6。

pyspark.sql.functions.desc(col)

基于给定列名的降序返回排序表达式。

1.3版新功能

pyspark.sql.functions.encode(col, charset)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#encode)

使用提供的字符集（'US-ASCII'，'ISO-8859-1'，'UTF-8'，'UTF-16BE'，'UTF-16LE'）中的一个从字符串计算第二个参数，'UTF-16'）。

1.5版新功能

pyspark.sql.functions.exp(col)

计算给定值的指数。

版本1.4中的新功能。

pyspark.sql.functions.explode(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#explode)

Returns a new row for each element in the given array or map.

>>> from pyspark.sql import Row

>>> eDF = spark.createDataFrame([Row(a=1, intlist=[1,2,3], mapfield={"a": "b"})])

>>> eDF.select(explode(eDF.intlist).alias("anInt")).collect()

[Row(anInt=1), Row(anInt=2), Row(anInt=3)]

>>> eDF.select(explode(eDF.mapfield).alias("key", "value")).show()

+---+-----+

|key|value|

+---+-----+

| a| b|

+---+-----+

New in version 1.4.

pyspark.sql.functions.expm1(col)

Computes the exponential of the given value minus one.

New in version 1.4.

pyspark.sql.functions.expr(str)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#expr)

Parses the expression string into the column that it represents

>>> df.select(expr("length(name)")).collect()

[Row(length(name)=5), Row(length(name)=3)]

New in version 1.5.

pyspark.sql.functions.factorial(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#factorial)

Computes the factorial of the given value.

>>> df = spark.createDataFrame([(5,)], ['n'])

>>> df.select(factorial(df.n).alias('f')).collect()

[Row(f=120)]

New in version 1.5.

pyspark.sql.functions.first(col, ignorenulls=False)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#first)

聚合函数：返回组中的第一个值。

该函数默认返回它看到的第一个值。 当ignoreNulls设置为true时，它将返回它看到的第一个非空值。 如果所有值都为空，则返回null。

1.3版新功能

pyspark.sql.functions.floor(col)

Computes the floor of the given value.

New in version 1.4.

pyspark.sql.functions.format\_number(col, d)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#format_number)

将数字X格式化为格式，如“＃， - ＃， - ＃.-”，舍入为小数位数，并以字符串形式返回结果。

|  |  |
| --- | --- |
| Parameters: | col – 要格式化的数值的列名  d – the N decimal places |

>>> spark.createDataFrame([(5,)], ['a']).select(format\_number('a', 4).alias('v')).collect()

[Row(v=u'5.0000')]

New in version 1.5.

pyspark.sql.functions.format\_string(format, \*cols)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#format_string)

以printf-style格式化参数，并将结果作为字符串列返回。

|  |  |
| --- | --- |
| Parameters: | col – the column name of the numeric value to be formatted  d – the N decimal places |

>>> df = spark.createDataFrame([(5, "hello")], ['a', 'b'])

>>> df.select(format\_string('%d %s', df.a, df.b).alias('v')).collect()

[Row(v=u'5 hello')]

New in version 1.5.

**pyspark.sql.functions.from\_json(col, schema, options={})**[**[source]**](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#from_json)

将包含JSON字符串的列解析为具有指定架构的[[StructType]]。 如果是不可稀释的字符串，则返回null。

|  |  |
| --- | --- |
| Parameters: | col – string column in json format  schema – a StructType to use when parsing the json column  options – options to control parsing. accepts the same options as the json datasource |

>>> from pyspark.sql.types import \*

>>> data = [(1, '''{"a": 1}''')]

>>> schema = StructType([StructField("a", IntegerType())])

>>> df = spark.createDataFrame(data, ("key", "value"))

>>> df.select(from\_json(df.value, schema).alias("json")).collect()

[Row(json=Row(a=1))]

New in version 2.1.

pyspark.sql.functions.from\_unixtime(timestamp, format='yyyy-MM-dd HH:mm:ss')[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#from_unixtime)

将从unix epoch（1970-01-01 00:00:00 UTC）的秒数转换为表示给定格式的当前系统时区中该时刻的时间戳的字符串。

New in version 1.5.

pyspark.sql.functions.from\_utc\_timestamp(timestamp, tz)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#from_utc_timestamp)

给定对应于UTC中某个特定时间的时间戳，返回对应于给定时区中同一时间的另一个时间戳。

>>> df = spark.createDataFrame([('1997-02-28 10:30:00',)], ['t'])

>>> df.select(from\_utc\_timestamp(df.t, "PST").alias('t')).collect()

[Row(t=datetime.datetime(1997, 2, 28, 2, 30))]

New in version 1.5.

pyspark.sql.functions.get\_json\_object(col, path)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#get_json_object)

基于指定的json路径从json字符串中提取json对象，并返回提取的json对象的json字符串。 如果输入的json字符串无效，它将返回null。

|  |  |
| --- | --- |
| Parameters: | col – string column in json format  path – path to the json object to extract |

>>> data = [("1", '''{"f1": "value1", "f2": "value2"}'''), ("2", '''{"f1": "value12"}''')]

>>> df = spark.createDataFrame(data, ("key", "jstring"))

>>> df.select(df.key, get\_json\_object(df.jstring, '$.f1').alias("c0"), \

... get\_json\_object(df.jstring, '$.f2').alias("c1") ).collect()

[Row(key=u'1', c0=u'value1', c1=u'value2'), Row(key=u'2', c0=u'value12', c1=None)]

New in version 1.6.

pyspark.sql.functions.greatest(\*cols)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#greatest)

返回列名列表的最大值，跳过空值。 此功能至少需要2个参数。 如果所有参数都为null，它将返回null。

>>> df = spark.createDataFrame([(1, 4, 3)], ['a', 'b', 'c'])

>>> df.select(greatest(df.a, df.b, df.c).alias("greatest")).collect()

[Row(greatest=4)]

New in version 1.5.

pyspark.sql.functions.grouping(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#grouping)

聚合函数：指示GROUP BY列表中的指定列是否聚合，在聚合中返回1，在结果集中返回0，否则不进行聚合。

>>> df.cube("name").agg(grouping("name"), sum("age")).orderBy("name").show()

+-----+--------------+--------+

| name|grouping(name)|sum(age)|

+-----+--------------+--------+

| null| 1| 7|

|Alice| 0| 2|

| Bob| 0| 5|

+-----+--------------+--------+

New in version 2.0.

pyspark.sql.functions.grouping\_id(\*cols)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#grouping_id)

Aggregate function: returns the level of grouping, equals to

(grouping(c1) << (n-1)) + (grouping(c2) << (n-2)) + ... + grouping(cn)

Note

The list of columns should match with grouping columns exactly, or empty (means all the grouping columns).

>>> df.cube("name").agg(grouping\_id(), sum("age")).orderBy("name").show()

+-----+-------------+--------+

| name|grouping\_id()|sum(age)|

+-----+-------------+--------+

| null| 1| 7|

|Alice| 0| 2|

| Bob| 0| 5|

+-----+-------------+--------+

New in version 2.0.

pyspark.sql.functions.hash(\*cols)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#hash)

Calculates the hash code of given columns, and returns the result as an int column.

>>> spark.createDataFrame([('ABC',)], ['a']).select(hash('a').alias('hash')).collect()

[Row(hash=-757602832)]

New in version 2.0.

pyspark.sql.functions.hex(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#hex)

Computes hex value of the given column, which could be [pyspark.sql.types.StringType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.StringType), [pyspark.sql.types.BinaryType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.BinaryType), [pyspark.sql.types.IntegerType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.IntegerType) or [pyspark.sql.types.LongType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.LongType).

>>> spark.createDataFrame([('ABC', 3)], ['a', 'b']).select(hex('a'), hex('b')).collect()

[Row(hex(a)=u'414243', hex(b)=u'3')]

New in version 1.5.

pyspark.sql.functions.hour(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#hour)

Extract the hours of a given date as integer.

>>> df = spark.createDataFrame([('2015-04-08 13:08:15',)], ['a'])

>>> df.select(hour('a').alias('hour')).collect()

[Row(hour=13)]

New in version 1.5.

pyspark.sql.functions.hypot(col1, col2)

Computes sqrt(a^2 + b^2) without intermediate overflow or underflow.

New in version 1.4.

pyspark.sql.functions.initcap(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#initcap)

将每个单词的第一个字母翻译成大写字母。

>>> spark.createDataFrame([('ab cd',)], ['a']).select(initcap("a").alias('v')).collect()

[Row(v=u'Ab Cd')]

New in version 1.5.

pyspark.sql.functions.input\_file\_name()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#input_file_name)

为当前Spark任务的文件名创建一个字符串列。

New in version 1.6.

pyspark.sql.functions.instr(str, substr)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#instr)

在给定的字符串中找到第一个substr列的位置。 如果任一参数为空，则返回null。

注意

位置不是零，而是基于1的索引。 如果在str中找不到substr，则返回0。

>>> df = spark.createDataFrame([('abcd',)], ['s',])

>>> df.select(instr(df.s, 'b').alias('s')).collect()

[Row(s=2)]

New in version 1.5.

pyspark.sql.functions.isnan(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#isnan)

如果列为NaN，则返回true的表达式。

>>> df = spark.createDataFrame([(1.0, float('nan')), (float('nan'), 2.0)], ("a", "b"))

>>> df.select(isnan("a").alias("r1"), isnan(df.a).alias("r2")).collect()

[Row(r1=False, r2=False), Row(r1=True, r2=True)]

New in version 1.6.

pyspark.sql.functions.isnull(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#isnull)

如果列为空，则返回true的表达式。

>>> df = spark.createDataFrame([(1, None), (None, 2)], ("a", "b"))

>>> df.select(isnull("a").alias("r1"), isnull(df.a).alias("r2")).collect()

[Row(r1=False, r2=False), Row(r1=True, r2=True)]

New in version 1.6.

pyspark.sql.functions.json\_tuple(col, \*fields)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#json_tuple)

根据给定的字段名称为json列创建一个新行。

|  |  |
| --- | --- |
| Parameters: | col – string column in json format  fields – list of fields to extract |

>>> data = [("1", '''{"f1": "value1", "f2": "value2"}'''), ("2", '''{"f1": "value12"}''')]

>>> df = spark.createDataFrame(data, ("key", "jstring"))

>>> df.select(df.key, json\_tuple(df.jstring, 'f1', 'f2')).collect()

[Row(key=u'1', c0=u'value1', c1=u'value2'), Row(key=u'2', c0=u'value12', c1=None)]

New in version 1.6.

pyspark.sql.functions.kurtosis(col)

聚合函数：返回组中值的峰度。

New in version 1.6.

pyspark.sql.functions.lag(col, count=1, default=None)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#lag)

窗口函数：返回当前行之前偏移行的值，如果在当前行之前有少于偏移量行，则返回defaultValue。 例如，一个偏移量将返回窗口分区中任何给定点的上一行。

这相当于SQL中的LAG函数。

|  |  |
| --- | --- |
| Parameters: | col – name of column or expression  count – number of row to extend  default – default value |

New in version 1.4.

pyspark.sql.functions.last(col, ignorenulls=False)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#last)

聚合函数：返回组中的最后一个值。

该函数默认返回它看到的最后一个值。 当ignoreNulls设置为true时，它将返回它看到的最后一个非空值。 如果所有值都为空，则返回null。

1.3版新功能

pyspark.sql.functions.last\_day(date)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#last_day)

返回给定日期所属的月份的最后一天。

>>> df = spark.createDataFrame([('1997-02-10',)], ['d'])

>>> df.select(last\_day(df.d).alias('date')).collect()

[Row(date=datetime.date(1997, 2, 28))]

New in version 1.5.

pyspark.sql.functions.lead(col, count=1, default=None)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#lead)

窗口函数：返回当前行之后的偏移行的值，如果当前行之后有少于偏移行，则返回defaultValue。 例如，一个偏移量将返回窗口分区中任何给定点的下一行。

这相当于SQL中的LEAD函数。

|  |  |
| --- | --- |
| Parameters: | col – name of column or expression  count – number of row to extend  default – default value |

New in version 1.4.

pyspark.sql.functions.least(\*cols)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#least)

返回列名列表的最小值，跳过空值。 此功能至少需要2个参数。 如果所有参数都为null，它将返回null。

>>> df = spark.createDataFrame([(1, 4, 3)], ['a', 'b', 'c'])

>>> df.select(least(df.a, df.b, df.c).alias("least")).collect()

[Row(least=1)]

New in version 1.5.

pyspark.sql.functions.length(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#length)

计算字符串或二进制表达式的长度。

>>> spark.createDataFrame([('ABC',)], ['a']).select(length('a').alias('length')).collect()

[Row(length=3)]

New in version 1.5.

pyspark.sql.functions.levenshtein(left, right)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#levenshtein)

计算两个给定字符串的Levenshtein距离。

>>> df0 = spark.createDataFrame([('kitten', 'sitting',)], ['l', 'r'])

>>> df0.select(levenshtein('l', 'r').alias('d')).collect()

[Row(d=3)]

New in version 1.5.

pyspark.sql.functions.lit(col)

创建一个字面值列。

New in version 1.3.

pyspark.sql.functions.locate(substr, str, pos=1)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#locate)

在位置pos之后，找到字符串列中第一次出现substr的位置。

Note

位置不是零，而是基于1的索引。 如果在str中找不到substr，则返回0。

|  |  |
| --- | --- |
| Parameters: | substr – a string  str – a Column of [pyspark.sql.types.StringType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.StringType)  pos – start position (zero based) |

>>> df = spark.createDataFrame([('abcd',)], ['s',])

>>> df.select(locate('b', df.s, 1).alias('s')).collect()

[Row(s=2)]

New in version 1.5.

pyspark.sql.functions.log(arg1, arg2=None)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#log)

返回第二个参数的第一个基于参数的对数。

如果只有一个参数，那么这将取自变量的自然对数。

>>> df.select(log(10.0, df.age).alias('ten')).rdd.map(lambda l: str(l.ten)[:7]).collect()

['0.30102', '0.69897']

>>> df.select(log(df.age).alias('e')).rdd.map(lambda l: str(l.e)[:7]).collect()

['0.69314', '1.60943']

New in version 1.5.

pyspark.sql.functions.log10(col)

计算Base 10中给定值的对数。

New in version 1.4.

pyspark.sql.functions.log1p(col)

计算给定值的自然对数加1。

New in version 1.4.

pyspark.sql.functions.log2(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#log2)

返回参数的基数2对数。

>>> spark.createDataFrame([(4,)], ['a']).select(log2('a').alias('log2')).collect()

[Row(log2=2.0)]

New in version 1.5.

pyspark.sql.functions.lower(col)

Converts a string column to lower case.

New in version 1.5.

pyspark.sql.functions.lpad(col, len, pad)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#lpad)

Left-pad the string column to width len with pad.

>>> df = spark.createDataFrame([('abcd',)], ['s',])

>>> df.select(lpad(df.s, 6, '#').alias('s')).collect()

[Row(s=u'##abcd')]

New in version 1.5.

pyspark.sql.functions.ltrim(col)

Trim the spaces from left end for the specified string value.

New in version 1.5.

pyspark.sql.functions.max(col)

聚合函数：返回组中表达式的最大值。

New in version 1.3.

pyspark.sql.functions.md5(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#md5)

计算MD5摘要，并将值作为32个字符的十六进制字符串返回。

>>> spark.createDataFrame([('ABC',)], ['a']).select(md5('a').alias('hash')).collect()

[Row(hash=u'902fbdd2b1df0c4f70b4a5d23525e932')]

New in version 1.5.

pyspark.sql.functions.mean(col)

聚合函数：返回组中的值的平均值。

New in version 1.3.

pyspark.sql.functions.min(col)

聚合函数：返回组中表达式的最小值。

New in version 1.3.

pyspark.sql.functions.minute(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#minute)

将给定日期的分钟提取为整数。

>>> df = spark.createDataFrame([('2015-04-08 13:08:15',)], ['a'])

>>> df.select(minute('a').alias('minute')).collect()

[Row(minute=8)]

New in version 1.5.

pyspark.sql.functions.monotonically\_increasing\_id()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#monotonically_increasing_id)

一列产生单调递增的64位整数。

生成的ID被保证是单调增加和唯一的，但不是连续的。 当前的实现将分区ID放在高31位，每个分区中的记录号在较低的33位中。 假设数据帧具有不到10亿个分区，每个分区的记录少于80亿。

例如，考虑一个具有两个分区的DataFrame，每个分区有3个记录。 该表达式将返回以下ID：0，1，2，8589934592（1L << 33），8589934593,8859934594。

>>> df0 = sc.parallelize(range(2), 2).mapPartitions(lambda x: [(1,), (2,), (3,)]).toDF(['col1'])

>>> df0.select(monotonically\_increasing\_id().alias('id')).collect()

[Row(id=0), Row(id=1), Row(id=2), Row(id=8589934592), Row(id=8589934593), Row(id=8589934594)]

New in version 1.6.

pyspark.sql.functions.month(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#month)

将给定日期的月份提取为整数。

>>> df = spark.createDataFrame([('2015-04-08',)], ['a'])

>>> df.select(month('a').alias('month')).collect()

[Row(month=4)]

New in version 1.5.

pyspark.sql.functions.months\_between(date1, date2)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#months_between)

返回date1和date2之间的月份数。

>>> df = spark.createDataFrame([('1997-02-28 10:30:00', '1996-10-30')], ['t', 'd'])

>>> df.select(months\_between(df.t, df.d).alias('months')).collect()

[Row(months=3.9495967...)]

New in version 1.5.

pyspark.sql.functions.nanvl(col1, col2)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#nanvl)

如果不是NaN，则返回col1，如果col1是NaN，则返回col2。

两个输入都应为浮点列（DoubleType或FloatType）。

>>> df = spark.createDataFrame([(1.0, float('nan')), (float('nan'), 2.0)], ("a", "b"))

>>> df.select(nanvl("a", "b").alias("r1"), nanvl(df.a, df.b).alias("r2")).collect()

[Row(r1=1.0, r2=1.0), Row(r1=2.0, r2=2.0)]

New in version 1.6.

pyspark.sql.functions.next\_day(date, dayOfWeek)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#next_day)

返回晚于date列值的第一个日期。

星期几参数不区分大小写，并接受：

“星期一”，“星期二”，“星期三”，“星期四”，“星期五”，“星期六”，“太阳”。

>>> df = spark.createDataFrame([('2015-07-27',)], ['d'])

>>> df.select(next\_day(df.d, 'Sun').alias('date')).collect()

[Row(date=datetime.date(2015, 8, 2))]

New in version 1.5.

pyspark.sql.functions.ntile(n)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#ntile)

窗口函数：在有序窗口分区中返回ntile组id（从1到n）。 例如，如果n是4，那么第一个四分之一行会得到1的值，第二个季度会得到2个，第三个季度会得到3个，而最后一个季度会得到4个。

这相当于SQL中的NTILE函数。

参数：

n - 整数

版本1.4中的新功能。

pyspark.sql.functions.percent\_rank()

窗口函数：返回窗口分区中的行的相对等级（即百分位数）。

新版本1.6。

pyspark.sql.functions.posexplode(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#posexplode)

为给定数组或地图中的位置返回每个元素的新行。

>>> from pyspark.sql import Row

>>> eDF = spark.createDataFrame([Row(a=1, intlist=[1,2,3], mapfield={"a": "b"})])

>>> eDF.select(posexplode(eDF.intlist)).collect()

[Row(pos=0, col=1), Row(pos=1, col=2), Row(pos=2, col=3)]

>>> eDF.select(posexplode(eDF.mapfield)).show()

+---+---+-----+

|pos|key|value|

+---+---+-----+

| 0| a| b|

+---+---+-----+

New in version 2.1.

pyspark.sql.functions.pow(col1, col2)

将第一个参数的值返回到第二个参数的幂。

New in version 1.4.

pyspark.sql.functions.quarter(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#quarter)

将给定日期的四分之一提取为整数。

>>> df = spark.createDataFrame([('2015-04-08',)], ['a'])

>>> df.select(quarter('a').alias('quarter')).collect()

[Row(quarter=2)]

New in version 1.5.

pyspark.sql.functions.radians(col)

将以度为单位的角度转换为以弧度测量的大致相等的角度。

New in version 2.1.

pyspark.sql.functions.rand(seed=None)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#rand)

从U [0.0，1.0]生成具有独立且相同分布（i.i.d.）样本的随机列。

New in version 1.4.

pyspark.sql.functions.randn(seed=None)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#randn)

从标准正态分布生成具有独立且相同分布（i.i.d.）样本的列。

版本1.4中的新功能。

pyspark.sql.functions.rank()

窗口函数：返回窗口分区中的行的排名。

Rank和dense\_rank之间的区别是，当有关系时，dense\_rank在排序顺序中没有留下空白。 也就是说，如果你是使用dense\_rank排名的比赛，并且有三个人排在第二位，那么你会说三个人都排在第二位，而下一个人排名第三。 排名将给我序列号，使得第三名的人（关系之后）将注册为第五名。

这相当于SQL中的RANK函数。

新版本1.6。

pyspark.sql.functions.regexp\_extract(str, pattern, idx)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#regexp_extract)

从指定的字符串列中提取与Java正则表达式匹配的特定组。 如果正则表达式不匹配，或指定的组不匹配，则返回一个空字符串。

>>> df = spark.createDataFrame([('100-200',)], ['str'])

>>> df.select(regexp\_extract('str', '(\d+)-(\d+)', 1).alias('d')).collect()

[Row(d=u'100')]

>>> df = spark.createDataFrame([('foo',)], ['str'])

>>> df.select(regexp\_extract('str', '(\d+)', 1).alias('d')).collect()

[Row(d=u'')]

>>> df = spark.createDataFrame([('aaaac',)], ['str'])

>>> df.select(regexp\_extract('str', '(a+)(b)?(c)', 2).alias('d')).collect()

[Row(d=u'')]

New in version 1.5.

pyspark.sql.functions.regexp\_replace(str, pattern, replacement)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#regexp_replace)

.将与regexp匹配的指定字符串值的所有子字符串替换为rep

>>> df = spark.createDataFrame([('100-200',)], ['str'])

>>> df.select(regexp\_replace('str', '(\d+)', '--').alias('d')).collect()

[Row(d=u'-----')]

New in version 1.5.

pyspark.sql.functions.repeat(col, n)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#repeat)

重复字符串列n次，并将其作为新的字符串列返回。

>>> df = spark.createDataFrame([('ab',)], ['s',])

>>> df.select(repeat(df.s, 3).alias('s')).collect()

[Row(s=u'ababab')]

New in version 1.5.

pyspark.sql.functions.reverse(col)

反转字符串列，并将其作为新的字符串列返回。

New in version 1.5.

pyspark.sql.functions.rint(col)

返回值与参数最接近的double值，并等于一个数学整数。

New in version 1.4.

pyspark.sql.functions.round(col, scale=0)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#round)

如果scale> = 0，则使用HALF\_UP舍入模式舍入给定值以缩放小数位数，或在scale <0时为整数部分。

>>> spark.createDataFrame([(2.5,)], ['a']).select(round('a', 0).alias('r')).collect()

[Row(r=3.0)]

New in version 1.5.

pyspark.sql.functions.row\_number()

窗口函数：返回从窗口分区开始的顺序号。

New in version 1.6.

pyspark.sql.functions.rpad(col, len, pad)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#rpad)

右键将字符串列填充至带宽的len。

>>> df = spark.createDataFrame([('abcd',)], ['s',])

>>> df.select(rpad(df.s, 6, '#').alias('s')).collect()

[Row(s=u'abcd##')]

New in version 1.5.

pyspark.sql.functions.rtrim(col)

从右端修剪指定字符串值的空格。

1.5版新功能

pyspark.sql.functions.second(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#second)

将给定日期的秒提取为整数。

>>> df = spark.createDataFrame([('2015-04-08 13:08:15',)], ['a'])

>>> df.select(second('a').alias('second')).collect()

[Row(second=15)]

New in version 1.5.

pyspark.sql.functions.sha1(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#sha1)

返回SHA-1的十六进制字符串结果。

>>> spark.createDataFrame([('ABC',)], ['a']).select(sha1('a').alias('hash')).collect()

[Row(hash=u'3c01bdbb26f358bab27f267924aa2c9a03fcfdb8')]

New in version 1.5.

pyspark.sql.functions.sha2(col, numBits)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#sha2)

返回SHA-2系列散列函数（SHA-224，SHA-256，SHA-384和SHA-512）的十六进制字符串结果。 numBits表示结果的所需位长度，其值必须为224,256,384,512或0（相当于256）。

>>> digests = df.select(sha2(df.name, 256).alias('s')).collect()

>>> digests[0]

Row(s=u'3bc51062973c458d5a6f2d8d64a023246354ad7e064b1e4e009ec8a0699a3043')

>>> digests[1]

Row(s=u'cd9fb1e148ccd8442e5aa74904cc73bf6fb54d1d54d333bd596aa9bb4bb4e961')

New in version 1.5.

pyspark.sql.functions.shiftLeft(col, numBits)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#shiftLeft)

移动给定的值numBits。

>>> spark.createDataFrame([(21,)], ['a']).select(shiftLeft('a', 1).alias('r')).collect()

[Row(r=42)]

New in version 1.5.

pyspark.sql.functions.shiftRight(col, numBits)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#shiftRight)

（签名）将给定值numBits移动到右边。

>>> spark.createDataFrame([(42,)], ['a']).select(shiftRight('a', 1).alias('r')).collect()

[Row(r=21)]

New in version 1.5.

pyspark.sql.functions.shiftRightUnsigned(col, numBits)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#shiftRightUnsigned)

无符号移位给定值numBits。

>>> df = spark.createDataFrame([(-42,)], ['a'])

>>> df.select(shiftRightUnsigned('a', 1).alias('r')).collect()

[Row(r=9223372036854775787)]

New in version 1.5.

pyspark.sql.functions.signum(col)

计算给定值的符号。

版本1.4中的新功能。

pyspark.sql.functions.sin(col)

计算给定值的正弦值。

版本1.4中的新功能。

pyspark.sql.functions.sinh(col)

计算给定值的双曲正弦。

版本1.4中的新功能。

pyspark.sql.functions.size(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#size)

集合函数：返回存储在列中的数组或映射的长度。

|  |  |
| --- | --- |
| Parameters: | col – name of column or expression |

>>> df = spark.createDataFrame([([1, 2, 3],),([1],),([],)], ['data'])

>>> df.select(size(df.data)).collect()

[Row(size(data)=3), Row(size(data)=1), Row(size(data)=0)]

New in version 1.5.

pyspark.sql.functions.skewness(col)

聚合函数：返回组中的值的偏度。

New in version 1.6.

pyspark.sql.functions.sort\_array(col, asc=True)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#sort_array)

集合函数：根据数组元素的自然排序，以升序或降序对输入数组进行排序。

|  |  |
| --- | --- |
| Parameters: | col – name of column or expression |

>>> df = spark.createDataFrame([([2, 1, 3],),([1],),([],)], ['data'])

>>> df.select(sort\_array(df.data).alias('r')).collect()

[Row(r=[1, 2, 3]), Row(r=[1]), Row(r=[])]

>>> df.select(sort\_array(df.data, asc=False).alias('r')).collect()

[Row(r=[3, 2, 1]), Row(r=[1]), Row(r=[])]

New in version 1.5.

pyspark.sql.functions.soundex(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#soundex)

返回一个字符串的SoundEx编码

>>> df = spark.createDataFrame([("Peters",),("Uhrbach",)], ['name'])

>>> df.select(soundex(df.name).alias("soundex")).collect()

[Row(soundex=u'P362'), Row(soundex=u'U612')]

New in version 1.5.

pyspark.sql.functions.spark\_partition\_id()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#spark_partition_id)

分区ID列。

Note

这是不确定的，因为它取决于数据分区和任务调度。

>>> df.repartition(1).select(spark\_partition\_id().alias("pid")).collect()

[Row(pid=0), Row(pid=0)]

New in version 1.6.

pyspark.sql.functions.split(str, pattern)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#split)

拆分函数，根据 pattern进行拆分

Note

pattern是一个表示正则表达式的字符串。

>>> df = spark.createDataFrame([('ab12cd',)], ['s',])

>>> df.select(split(df.s, '[0-9]+').alias('s')).collect()

[Row(s=[u'ab', u'cd'])]

New in version 1.5.

pyspark.sql.functions.sqrt(col)

计算指定浮点值的平方根。

New in version 1.3.

pyspark.sql.functions.stddev(col)

聚合函数：返回组中表达式的无偏样本标准差。

New in version 1.6.

pyspark.sql.functions.stddev\_pop(col)

聚合函数：返回组中表达式的总体标准偏差。

New in version 1.6.

pyspark.sql.functions.stddev\_samp(col)

聚合函数：返回组中表达式的无偏样本标准差。

New in version 1.6.

pyspark.sql.functions.struct(\*cols)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#struct)

创建一个新的struct列。

|  |  |
| --- | --- |
| Parameters: | cols – list of column names (string) or list of Column expressions |

>>> df.select(struct('age', 'name').alias("struct")).collect()

[Row(struct=Row(age=2, name=u'Alice')), Row(struct=Row(age=5, name=u'Bob'))]

>>> df.select(struct([df.age, df.name]).alias("struct")).collect()

[Row(struct=Row(age=2, name=u'Alice')), Row(struct=Row(age=5, name=u'Bob'))]

New in version 1.4.

pyspark.sql.functions.substring(str, pos, len)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#substring)

Substring从pos开始，当str为String类型时，长度为len，或者返回以字节开头的字节数组的slice，当str为二进制类型时，长度为len

>>> df = spark.createDataFrame([('abcd',)], ['s',])

>>> df.select(substring(df.s, 1, 2).alias('s')).collect()

[Row(s=u'ab')]

New in version 1.5.

pyspark.sql.functions.substring\_index(str, delim, count)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#substring_index)

在分隔符分隔符的出现之前返回字符串str中的子字符串。 如果count为正，则返回最终分隔符左边的所有内容（从左开始计数）。 如果count为负数，则返回最终分隔符的右侧（从右侧计数）。 substring\_index在搜索delim时执行区分大小写匹配。

>>> df = spark.createDataFrame([('a.b.c.d',)], ['s'])

>>> df.select(substring\_index(df.s, '.', 2).alias('s')).collect()

[Row(s=u'a.b')]

>>> df.select(substring\_index(df.s, '.', -3).alias('s')).collect()

[Row(s=u'b.c.d')]

New in version 1.5.

pyspark.sql.functions.sum(col)

聚合函数：返回表达式中所有值的总和。

New in version 1.3.

pyspark.sql.functions.sumDistinct(col)

聚合函数：返回表达式中不同值的总和。

New in version 1.3.

pyspark.sql.functions.tan(col)

计算给定值的切线。

New in version 1.4.

pyspark.sql.functions.tanh(col)

计算给定值的双曲正切值。

New in version 1.4.

pyspark.sql.functions.toDegrees(col)

Note

在2.1中弃用，改为使用度。

New in version 1.4.

pyspark.sql.functions.toRadians(col)

Note

在2.1中弃用，而是使用弧度。

New in version 1.4.

pyspark.sql.functions.to\_date(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#to_date)

将pyspark.sql.types.StringType或pyspark.sql.types.TimestampType的列转换为pyspark.sql.types.DateType。

>>> df = spark.createDataFrame([('1997-02-28 10:30:00',)], ['t'])

>>> df.select(to\_date(df.t).alias('date')).collect()

[Row(date=datetime.date(1997, 2, 28))]

New in version 1.5.

pyspark.sql.functions.to\_json(col, options={})[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#to_json)

将包含[[StructType]]的列转换为JSON字符串。 在不支持的类型的情况下抛出异常。

|  |  |
| --- | --- |
| Parameters: | col – name of column containing the struct  options – options to control converting. accepts the same options as the json datasource |

>>> from pyspark.sql import Row

>>> from pyspark.sql.types import \*

>>> data = [(1, Row(name='Alice', age=2))]

>>> df = spark.createDataFrame(data, ("key", "value"))

>>> df.select(to\_json(df.value).alias("json")).collect()

[Row(json=u'{"age":2,"name":"Alice"}')]

New in version 2.1.

pyspark.sql.functions.to\_utc\_timestamp(timestamp, tz)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#to_utc_timestamp)

给定对应于给定时区中某一特定时间的时间戳，返回与UTC中相同时间相对应的另一个时间戳。

>>> df = spark.createDataFrame([('1997-02-28 10:30:00',)], ['t'])

>>> df.select(to\_utc\_timestamp(df.t, "PST").alias('t')).collect()

[Row(t=datetime.datetime(1997, 2, 28, 18, 30))]

New in version 1.5.

pyspark.sql.functions.translate(srcCol, matching, replace)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#translate)

一个函数通过匹配中的一个字符来翻译srcCol中的任何字符。 替换中的字符与匹配中的字符相对应。 当字符串中的任何字符与匹配中的字符匹配时，会发生翻译。

>>> spark.createDataFrame([('translate',)], ['a']).select(translate('a', "rnlt", "123") \

... .alias('r')).collect()

[Row(r=u'1a2s3ae')]

New in version 1.5.

pyspark.sql.functions.trim(col)

Trim the spaces from both ends for the specified string column.

New in version 1.5.

pyspark.sql.functions.trunc(date, format)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#trunc)

Returns date truncated to the unit specified by the format.

|  |  |
| --- | --- |
| Parameters: | format – ‘year’, ‘YYYY’, ‘yy’ or ‘month’, ‘mon’, ‘mm’ |

>>> df = spark.createDataFrame([('1997-02-28',)], ['d'])

>>> df.select(trunc(df.d, 'year').alias('year')).collect()

[Row(year=datetime.date(1997, 1, 1))]

>>> df.select(trunc(df.d, 'mon').alias('month')).collect()

[Row(month=datetime.date(1997, 2, 1))]

New in version 1.5.

pyspark.sql.functions.udf(f, returnType=StringType)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#udf)

创建一个表示用户定义函数（UDF）的列表达式。

注意

用户定义的函数必须是确定性的。 由于优化，可以消除重复的调用，或者该函数甚至可以被调用比查询中存在的次数更多的时间。

|  |  |
| --- | --- |
| Parameters: | f – python function  returnType – a [pyspark.sql.types.DataType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.DataType) object |

>>> from pyspark.sql.types import IntegerType

>>> slen = udf(lambda s: len(s), IntegerType())

>>> df.select(slen(df.name).alias('slen')).collect()

[Row(slen=5), Row(slen=3)]

New in version 1.3.

pyspark.sql.functions.unbase64(col)

对BASE64编码的字符串列进行解码，并将其作为二进制列返回。

1.5版新功能

pyspark.sql.functions.unhex(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#unhex)

十六进制反向 将每对字符解释为十六进制数，并转换为数字的字节表示。

>>> spark.createDataFrame([('414243',)], ['a']).select(unhex('a')).collect()

[Row(unhex(a)=bytearray(b'ABC'))]

New in version 1.5.

pyspark.sql.functions.unix\_timestamp(timestamp=None, format='yyyy-MM-dd HH:mm:ss')[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#unix_timestamp)

使用默认时区和默认语言环境，将给定模式（'yyyy-MM-dd HH：mm：ss'）的时间字符串转换为Unix时间戳（以秒为单位），如果失败则返回null。

如果timestamp为None，则返回当前时间戳。

New in version 1.5.

pyspark.sql.functions.upper(col)

将字符串列转换为大写。

New in version 1.5.

pyspark.sql.functions.var\_pop(col)

聚合函数：返回组中值的总体方差。

New in version 1.6.

pyspark.sql.functions.var\_samp(col)

聚合函数：返回组中值的无偏差值。

New in version 1.6.

pyspark.sql.functions.variance(col)

聚合函数：返回组中值的总体方差。

New in version 1.6.

pyspark.sql.functions.weekofyear(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#weekofyear)

将给定日期的周数提取为整数。

>>> df = spark.createDataFrame([('2015-04-08',)], ['a'])

>>> df.select(weekofyear(df.a).alias('week')).collect()

[Row(week=15)]

New in version 1.5.

pyspark.sql.functions.when(condition, value)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#when)

评估条件列表并返回多个可能的结果表达式之一。 如果不调用Column.otherwise（），则返回无匹配条件的None。

|  |  |
| --- | --- |
| Parameters: | condition – a boolean Column expression.  value – a literal value, or a Column expression. |

>>> df.select(when(df['age'] == 2, 3).otherwise(4).alias("age")).collect()

[Row(age=3), Row(age=4)]

>>> df.select(when(df.age == 2, df.age + 1).alias("age")).collect()

[Row(age=3), Row(age=None)]

New in version 1.4.

pyspark.sql.functions.window(timeColumn, windowDuration, slideDuration=None, startTime=None)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#window)

给定一个时间戳指定列，将行卷入一个或多个时间窗口。窗口开始是包容性的，但是窗口结束是排他的，例如。 12:05将在窗口[12：05,12：10]，但不在[12：00,12：05]。 Windows可以支持微秒精度。不支持按月的顺序的Windows。

时间列必须是pyspark.sql.types.TimestampType。

持续时间作为字符串提供，例如'1秒'，'1天12小时'，'2分钟'。有效间隔字符串为“周”，“日”，“小时”，“分钟”，“秒”，“毫秒”，“微秒”。如果没有提供slideDuration，窗口将翻滚窗口。

startTime是相对于启动窗口间隔的1970-01-01 00:00:00 UTC的偏移量。例如，为了让小时翻转的窗户从小时开始15分钟，例如12：15-13：15,13：15-14：15 ...提供

>>> df = spark.createDataFrame([("2016-03-11 09:00:07", 1)]).toDF("date", "val")

>>> w = df.groupBy(window("date", "5 seconds")).agg(sum("val").alias("sum"))

>>> w.select(w.window.start.cast("string").alias("start"),

... w.window.end.cast("string").alias("end"), "sum").collect()

[Row(start=u'2016-03-11 09:00:05', end=u'2016-03-11 09:00:10', sum=1)]

New in version 2.0.

pyspark.sql.functions.year(col)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/functions.html#year)

Extract the year of a given date as integer.

>>> df = spark.createDataFrame([('2015-04-08',)], ['a'])

>>> df.select(year('a').alias('year')).collect()

[Row(year=2015)]

New in version 1.5.

pyspark.sql.streaming module

class pyspark.sql.streaming.StreamingQuery(jsq)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQuery)

当新数据到达时，在后台连续执行的查询的句柄。 所有这些方法都是线程安全的。

注意

试验

2.0版新功能

awaitTermination(timeout=None)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQuery.awaitTermination)

等待通过query.stop（）或异常终止此查询。 如果查询已经以异常终止，那么异常将被抛出。 如果设置了超时，则会在超时秒内返回查询是否已终止。

如果查询已经终止，那么对此方法的所有后续调用将立即返回（如果查询被stop（）终止），或者立即抛出异常（如果查询已被异常终止）。

抛出StreamingQueryException，如果此查询已终止异常

2.0版新功能

exception()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQuery.exception)

|  |  |
| --- | --- |
| Returns: | 如果查询被异常终止，则为StreamingQueryException，或者为None。 |

New in version 2.1.

explain(extended=False)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQuery.explain)

将（逻辑和物理）计划打印到控制台进行调试。

|  |  |
| --- | --- |
| Parameters: | extended – boolean, default False. If False, prints only the physical plan. |

>>> sq = sdf.writeStream.format('memory').queryName('query\_explain').start()

>>> sq.processAllAvailable() # Wait a bit to generate the runtime plans.

>>> sq.explain()

== Physical Plan ==

...

>>> sq.explain(True)

== Parsed Logical Plan ==

...

== Analyzed Logical Plan ==

...

== Optimized Logical Plan ==

...

== Physical Plan ==

...

>>> sq.stop()

New in version 2.1.

id[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQuery.id)

返回从检查点数据重新启动的该查询的唯一ID。 也就是说，当第一次启动查询时，会生成此ID，并且每次从检查点数据重新启动时都会相同。 在Spark群集中只能有一个具有相同ID的查询活动。 另请参见runId。

New in version 2.0.

isActive[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQuery.isActive)

此流式查询当前是否处于活动状态。

2.0版新功能

lastProgress[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQuery.lastProgress)

返回此流式查询的最新StreamingQueryProgress更新，如果没有进度更新则返回None：return：a map

2.1版新功能

name[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQuery.name)

返回用户指定的查询名称，如果未指定则返回null。 该名称可以在org.apache.spark.sql.streaming.DataStreamWriter中指定为dataframe.writeStream.queryName（“query”）。start（）。 如果设置此名称，在所有活动查询中必须是唯一的。

2.0版新功能

processAllAvailable()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQuery.processAllAvailable)

阻塞，直到源中的所有可用数据已被处理并提交到接收器。 此方法用于测试。

注意

在持续到达数据的情况下，该方法可能会永久阻止。 另外，只有在调用之前已将数据同步附加到流源的数据之前，才能保证此方法。 （即getOffset必须立即反映加法）。

2.0版新功能

recentProgress[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQuery.recentProgress)

返回此查询的最新[[StreamingQueryProgress]]更新的数组。 每个流保留的进度更新数由Spark会话配置spark.sql.streaming.numRecentProgressUpdates配置。

2.1版新功能

runId[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQuery.runId)

返回重新启动时不会持续的此查询的唯一ID。 也就是说，启动的每个查询（或从检查点重新启动）将具有不同的runId。

2.1版新功能

status[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQuery.status)

返回查询的当前状态。

New in version 2.1.

stop()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQuery.stop)

停止此流式查询。

New in version 2.0.

class pyspark.sql.streaming.StreamingQueryManager(jsqm)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQueryManager)

一个管理所有StreamingQuery StreamingQueries的类。

Note

Experimental

New in version 2.0.

active[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQueryManager.active)

返回与此SQLContext关联的活动查询的列表

>>> sq = sdf.writeStream.format('memory').queryName('this\_query').start()

>>> sqm = spark.streams

>>> # get the list of active streaming queries

>>> [q.name for q in sqm.active]

[u'this\_query']

>>> sq.stop()

New in version 2.0.

awaitAnyTermination(timeout=None)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQueryManager.awaitAnyTermination)

等待直到关联的SQLContext的任何查询从创建上下文或从resetTerminated（）被调用后终止。如果任何查询被异常终止，那么将抛出异常。如果设置了超时，则会在超时秒内返回查询是否已终止。

如果查询已经终止，则对awaitAnyTermination（）的后续调用将立即返回（如果查询被query.stop（）终止）），或者立即抛出异常（如果查询被异常终止）。使用resetTerminated（）清除过去的终止并等待新的终止。

在resetTermination（）被调用的情况下多个查询已经终止的情况下，如果任何查询已被异常终止，那么awaitAnyTermination（）将抛出任何异常。为了正确记录多个查询中的异常，用户需要在任何异常终止后停止所有异常，然后检查每个查询的query.exception（）。

抛出StreamingQueryException，如果此查询已终止异常

2.0版新功能

get(id)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQueryManager.get)

如果此名称的活动查询不存在，则返回此SQLContext中的活动查询，或者抛出异常。

>>> sq = sdf.writeStream.format('memory').queryName('this\_query').start()

>>> sq.name

u'this\_query'

>>> sq = spark.streams.get(sq.id)

>>> sq.isActive

True

>>> sq = sqlContext.streams.get(sq.id)

>>> sq.isActive

True

>>> sq.stop()

New in version 2.0.

resetTerminated()[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#StreamingQueryManager.resetTerminated)

忘记关于过去终止的查询，以便再次使用awaitAnyTermination（）等待新的终止。

>>> spark.streams.resetTerminated()

New in version 2.0.

class pyspark.sql.streaming.DataStreamReader(spark)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamReader)

用于从外部存储系统（例如文件系统，键值存储等）加载流式DataFrame的接口。 使用spark.readStream（）来访问它。

注意

实验。

New in version 2.0.

csv(path, schema=None, sep=None, encoding=None, quote=None, escape=None, comment=None, header=None, inferSchema=None, ignoreLeadingWhiteSpace=None, ignoreTrailingWhiteSpace=None, nullValue=None, nanValue=None, positiveInf=None, negativeInf=None, dateFormat=None, timestampFormat=None, maxColumns=None, maxCharsPerColumn=None, maxMalformedLogPerPartition=None, mode=None)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamReader.csv)

加载CSV文件流，并将结果作为DataFrame返回。

如果inferSchema被启用，此功能将通过输入一次来确定输入模式。 为避免遍历整个数据一次，请禁用inferSchema选项或使用模式显式指定模式。

注意

实验。

|  |  |
| --- | --- |
| Parameters: | path – string, or list of strings, for input path(s).  schema – 输入模式的可选pyspark.sql.types.StructType。  sep – 将单个字符设置为每个字段和值的分隔符。 如果设置为None，它将使用默认值，。  encoding –按照给定的编码类型对CSV文件进行解码。如果设置为无，则使用默认值UTF-8。  quote –设置用于转义引用值的单个字符，其中分隔符可以作为值的一部分。 如果设置为None，则使用默认值“。如果要关闭报价，则需要设置一个空字符串。  escape – 将用于转义引号的单个字符设置为已引用的值。 如果设置为None，它将使用默认值\。  comment – 设置用于跳过以此字符开头的行的单个字符。 默认情况下（无），它被禁用。  header – 使用第一行作为列的名称。 如果设置为None，它将使用默认值false。  inferSchema –从数据自动推断输入模式。 它需要一个额外的数据传递。 如果设置为None，它将使用默认值false。  ignoreLeadingWhiteSpace – 定义是否应该跳过正在读取的值的引导空格。 如果设置为None，它将使用默认值false。  ignoreTrailingWhiteSpace –定义是否应跳过正在读取的值的尾随空格。 如果设置为None，它将使用默认值false。  nullValue – 设置一个空值的字符串表示形式。 如果设置为None，它将使用默认值空字符串。 自2.0.1起，此null值参数适用于所有支持的类型，包括字符串类型。  nanValue –设置非数字值的字符串表示形式。 如果设置为None，则使用默认值NaN。  positiveInf – 设置正无穷大值的字符串表示。 如果设置为None，它将使用默认值Inf。  negativeInf – 设置负无穷大值的字符串表示。 如果设置为None，它将使用默认值Inf。  dateFormat –设置表示日期格式的字符串。 自定义日期格式遵循java.text.SimpleDateFormat的格式。 这适用于日期类型。 如果设置为None，则使用默认值yyyy-MM-dd。  timestampFormat –设置指示时间戳格式的字符串。 自定义日期格式遵循java.text.SimpleDateFormat的格式。 这适用于时间戳类型。 如果设置为None，则使用默认值yyyy-MM-dd'T'HH：mm：ss.SSSZZ。  maxColumns – 定义了一条记录可以拥有多少列的硬限制。 如果设置为None，它将使用默认值20480。  maxCharsPerColumn – 定义读取任何给定值允许的最大字符数。 如果设置为None，则使用默认值，-1表示无限长度。  mode – 允许在解析期间处理损坏记录的模式。 如果没有  设置，它使用默认值PERMISSIVE。  PERMISSIVE : 将其他字段设置为null，当它遇到损坏的记录。  当用户设置模式时，它为额外的字段设置为空。  DROPMALFORMED : 忽略整个损坏的记录。  FAILFAST：遇到损坏的记录时抛出异常。 |

>>> csv\_sdf = spark.readStream.csv(tempfile.mkdtemp(), schema = sdf\_schema)

>>> csv\_sdf.isStreaming

True

>>> csv\_sdf.schema == sdf\_schema

True

New in version 2.0.

format(source)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamReader.format)

指定输入数据源格式。

Note

Experimental.

|  |  |
| --- | --- |
| Parameters: | source – string, name of the data source, e.g. ‘json’, ‘parquet’. |

>>> s = spark.readStream.format("text")

New in version 2.0.

json(path, schema=None, primitivesAsString=None, prefersDecimal=None, allowComments=None, allowUnquotedFieldNames=None, allowSingleQuotes=None, allowNumericLeadingZero=None, allowBackslashEscapingAnyCharacter=None, mode=None, columnNameOfCorruptRecord=None, dateFormat=None, timestampFormat=None)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamReader.json)

加载JSON文件流（JSON Lines文本格式或换行符分隔的JSON）并返回：class`DataFrame`。

如果未指定模式参数，则此函数会经过输入一次以确定输入模式。

注意

实验。

|  |  |
| --- | --- |
| Parameters: | path – string represents path to the JSON dataset, or RDD of Strings storing JSON objects.  schema – an optional [pyspark.sql.types.StructType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.StructType) for the input schema.  primitivesAsString – 将所有原始值推断为字符串类型。 如果设置为None，它将使用默认值false。  prefersDecimal – 将所有浮点值推定为十进制类型。 如果值不符合十进制，则将它们推定为双精度。 如果设置为None，它将使用默认值false。  allowComments – 忽略JSON记录中的Java / C ++风格注释。 如果设置为None，它将使用默认值false。  allowUnquotedFieldNames –允许无引号的JSON字段名称。 如果设置为None，它将使用默认值false。  allowSingleQuotes –允许单引号除双引号之外。 如果设置为None，它将使用默认值true.  allowNumericLeadingZero – 允许数字前导零（例如00012）。 如果设置为None，它将使用默认值false。  allowBackslashEscapingAnyCharacter – 允许使用反斜杠引用机制接受所有字符的引用。 如果设置为None，它将使用默认值false。  mode –  允许在解析期间处理损坏记录的模式。 如果没有  设置，它使用默认值PERMISSIVE。  PERMISSIVE :将其他字段设置为null，当它遇到损坏的记录，并将格式错误的字符串放入由columnNameOfCorruptRecord配置的新字段。 当用户设置模式时，它为额外的字段设置为空。  DROPMALFORMED : 忽略整个损坏的记录。  FAILFAST：遇到损坏的记录时抛出异常。  columnNameOfCorruptRecord – 允许重命名具有由PERMISSIVE模式创建的畸形字符串的新字段。 这将覆盖spark.sql.columnNameOfCorruptRecord。 如果设置为None，它将使用spark.sql.columnNameOfCorruptRecord中指定的值。  dateFormat –设置表示日期格式的字符串。 自定义日期格式遵循java.text.SimpleDateFormat的格式。 这适用于日期类型。 如果设置为None，则使用默认值yyyy-MM-dd。  timestampFormat –设置指示时间戳格式的字符串。 自定义日期格式遵循java.text.SimpleDateFormat的格式。 这适用于时间戳类型。 如果设置为None，则使用默认值yyyy-MM-dd'T'HH：mm：ss.SSSZZ。 |

>>> json\_sdf = spark.readStream.json(tempfile.mkdtemp(), schema = sdf\_schema)

>>> json\_sdf.isStreaming

True

>>> json\_sdf.schema == sdf\_schema

True

New in version 2.0.

load(path=None, format=None, schema=None, \*\*options)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamReader.load)

从数据源加载数据流，并将其作为：class`DataFrame返回。

Note

Experimental.

|  |  |
| --- | --- |
| Parameters: | path – optional string for file-system backed data sources.  format – 可选字符串，用于数据源的格式。Default to ‘parquet’.  schema – optional [pyspark.sql.types.StructType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.StructType) for the input schema.  options – all other string options |

>>> json\_sdf = spark.readStream.format("json") \

... .schema(sdf\_schema) \

... .load(tempfile.mkdtemp())

>>> json\_sdf.isStreaming

True

>>> json\_sdf.schema == sdf\_schema

True

New in version 2.0.

option(key, value)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamReader.option)

为底层数据源添加输入选项。

Note

Experimental.

>>> s = spark.readStream.option("x", 1)

New in version 2.0.

options(\*\*options)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamReader.options)

添加底层数据源的输入选项。

Note

Experimental.

>>> s = spark.readStream.options(x="1", y=2)

New in version 2.0.

parquet(path)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamReader.parquet)

加载Parquet文件流，将结果作为DataFrame返回。

您可以设置以下Parquet特定选项来阅读Parquet文件：

mergeSchema：设置是否合并从所有Parquet部件文件中收集的模式。 这将覆盖spark.sql.parquet.mergeSchema。 默认值在spark.sql.parquet.mergeSchema中指定。

Note

Experimental.

>>> parquet\_sdf = spark.readStream.schema(sdf\_schema).parquet(tempfile.mkdtemp())

>>> parquet\_sdf.isStreaming

True

>>> parquet\_sdf.schema == sdf\_schema

True

New in version 2.0.

schema(schema)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamReader.schema)

指定输入模式。

一些数据源（例如JSON）可以从数据自动推断输入模式。 通过在此处指定模式，底层数据源可以跳过模式推理步骤，从而加快数据加载速度。

Note

Experimental.

|  |  |
| --- | --- |
| Parameters: | schema – a [pyspark.sql.types.StructType](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.types.StructType) object |

>>> s = spark.readStream.schema(sdf\_schema)

New in version 2.0.

text(path)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamReader.text)

加载文本文件流并返回一个DataFrame，其架构以名为“value”的字符串列开始，如果有的话，则返回分区列。

文本文件中的每一行都是生成的DataFrame中的一行。

注意

实验。

|  |  |
| --- | --- |
| Parameters: | paths – string, or list of strings, for input path(s). |

>>> text\_sdf = spark.readStream.text(tempfile.mkdtemp())

>>> text\_sdf.isStreaming

True

>>> "value" in str(text\_sdf.schema)

True

New in version 2.0.

class pyspark.sql.streaming.DataStreamWriter(df)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamWriter)

用于将流DataFrame写入外部存储系统（例如文件系统，键值存储等）的接口。 使用DataFrame.writeStream（）来访问它。

注意

实验。

New in version 2.0.

format(source)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamWriter.format)

指定底层输出数据源。

Note

Experimental.

|  |  |
| --- | --- |
| Parameters: | source – string, name of the data source, which for now can be ‘parquet’. |

>>> writer = sdf.writeStream.format('json')

New in version 2.0.

option(key, value)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamWriter.option)

为底层数据源添加一个输出选项。

Note

Experimental.

New in version 2.0.

options(\*\*options)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamWriter.options)

添加底层数据源的输出选项。

Note

Experimental.

New in version 2.0.

outputMode(outputMode)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamWriter.outputMode)

指定数据流DataFrame / Dataset的数据如何写入流接收器。

Options include:

append:Only the new rows in the streaming DataFrame/Dataset will be written to

the sink

complete:每次这些更新时，流DataFrame / Dataset中的所有行将被写入宿主

update:只有在流式DataFrame / Dataset中更新的行才会在每次有更新时被写入宿。 如果查询不包含聚合，它将等同于附加模式。

Note

Experimental.

>>> writer = sdf.writeStream.outputMode('append')

New in version 2.0.

partitionBy(\*cols)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamWriter.partitionBy)

按照文件系统上给定列的输出进行分区。

如果指定，输出将放置在文件系统上，类似于Hive的分区方案。

Note

Experimental.

|  |  |
| --- | --- |
| Parameters: | cols – name of columns |

New in version 2.0.

queryName(queryName)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamWriter.queryName)

指定可以用start（）启动的StreamingQuery的名称。 该名称在关联的SparkSession中的所有当前活动查询中必须是唯一的。

Note

Experimental.

|  |  |
| --- | --- |
| Parameters: | queryName – unique name for the query |

>>> writer = sdf.writeStream.queryName('streaming\_query')

New in version 2.0.

start(path=None, format=None, outputMode=None, partitionBy=None, queryName=None, \*\*options)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamWriter.start)

将DataFrame的内容流传输到数据源。

数据源由格式和一组选项指定。 如果未指定格式，将使用由spark.sql.sources.default配置的默认数据源。

Note

Experimental.

|  |  |
| --- | --- |
| Parameters: | path – the path in a Hadoop supported file system  format – the format used to save  outputMode – 指定流数据帧/数据集的数据如何写入流宿。  append:只有流DataFrame / Dataset中的新行将被写入接收器  complete:每次这些更新时，流DataFrame / Dataset中的所有行将被写入宿主  update:只有在流式DataFrame / Dataset中更新的行才会在每次有更新时被写入宿。 如果查询不包含聚合，它将等同于附加模式。  partitionBy –分区列的名称  queryName –查询的唯一名称  options –所有其他字符串选项。 您可能希望为大多数流提供checkpointLocation，但是它不是内存流所必需的。 |

>>> sq = sdf.writeStream.format('memory').queryName('this\_query').start()

>>> sq.isActive

True

>>> sq.name

u'this\_query'

>>> sq.stop()

>>> sq.isActive

False

>>> sq = sdf.writeStream.trigger(processingTime='5 seconds').start(

... queryName='that\_query', outputMode="append", format='memory')

>>> sq.name

u'that\_query'

>>> sq.isActive

True

>>> sq.stop()

New in version 2.0.

trigger(\*args, \*\*kwargs)[[source]](http://spark.apache.org/docs/latest/api/python/_modules/pyspark/sql/streaming.html#DataStreamWriter.trigger)

设置流查询的触发器。 如果没有设置，它将尽可能快地运行查询，这相当于将触发器设置为processingTime ='0秒'。

Note

Experimental.

|  |  |
| --- | --- |
| Parameters: | processingTime – a processing time interval as a string, e.g. ‘5 seconds’, ‘1 minute’. |

>>> # trigger the query for execution every 5 seconds

>>> writer = sdf.writeStream.trigger(processingTime='5 seconds')

New in version 2.0.