

第 5 章

Spark SQL的DataFrame 操作大全

通过对前一章 Spark SQL 编程入门这样的基础内容的学习，我们逐渐地了解了作为 Spark 四大高级模块之一的 Spark SQL 以及利用 Spark SQL 编程需掌握的基本概念和基本步骤。

基本步骤总结起来就是：Spark 程序中利用 SparkSession 对象提供的读取相关数据源的方法读取来自不同数据源的结构化数据，转化为 DataFrame，当然也可以将现成 RDDs 转化为 DataFrame，在转化为 DataFrame 的过程中，需自识别或指定 DataFrame 的 Schema，之后可以直接通过 DataFrame 的 API 进行数据分析，当然也可以直接将 DataFrame 注册为 table，直接利用 Sparksession 提供的 sql 方法在已注册的表上进行 SQL 查询，DataFrame 在转化为临时视图时需根据实际情况选择是否转化为全局临时表

在接下来的 Spark SQL 编程进阶，我们开始深入对 DataFrame 丰富强大的 API 进行研究，进而帮助读者轻松高效地组合使用 DataFrame 所提供的 API 来实现业务需求。

5.1 由 JSON 文件生成所需的 DataFrame 对象

正如上一章提到的，Spark-SQL 可以以 RDD 对象、Parquet 文件、JSON 文件、Hive 表，以及通过 JDBC 连接到其他关系型数据库表作为数据源来生成 DataFrame 对象。本章将以 JSON 文件为数据源，读取其中数据生成 DataFrame 对象，进而在该 DataFrame 对象上通过各种实例操作讲解 DataFrame API 的使用，本小节将演示如何把 JSON 文件作为数据源，生成所需的 DataFrame 对象。

图 5-1 是作为数据源的 JSON 文件内容。

```

people.json
1 {"name":"Michael", "age":20,"sex":"男","institute":"信息学院","phone":18663930185}
2 {"name":"Andy", "age":21,"sex":"女","institute":"信息学院","phone":18663930186}
3 {"name":"Justin", "age":19,"sex":"男","institute":"信息学院","phone":18663930187}
4 {"name":"Aaron", "age":19,"sex":"男","institute":"信息学院","phone":18663930188}
5 {"name":"Abbott", "age":20,"sex":"男","institute":"信息学院","phone":18663930189}
6 {"name":"Abel", "age":19,"sex":"女","institute":"信息学院","phone":18663930180}
7 {"name":"Abner", "age":20,"sex":"男","institute":"材料学院","phone":18163930185}
8 {"name":"Abraham", "age":21,"sex":"女","institute":"材料学院","phone":18263930185}
9 {"name":"Adam", "age":19,"sex":"男","institute":"材料学院","phone":18363930185}
10 {"name":"Addison", "age":19,"sex":"男","institute":"材料学院","phone":18463930185}
11 {"name":"Barret", "age":18,"sex":"女","institute":"材料学院","phone":18563930185}
12 {"name":"Basil", "age":19,"sex":"女","institute":"化工学院","phone":18763930185}
13 {"name":"Beau", "age":20,"sex":"女","institute":"化工学院","phone":18863930185}
14 {"name":"Benedict", "age":20,"sex":"男","institute":"化工学院","phone":18963930185}
15 {"name":"Cedric", "age":19,"sex":"男","institute":"化工学院","phone":18603930185}
16 {"name":"Chad", "age":21,"sex":"男","institute":"化工学院","phone":18613930185}
17 {"name":"Chapman", "age":22,"sex":"女","institute":"化工学院","phone":18623930185}
18 {"name":"Clement", "age":19,"sex":"男","institute":"化工学院","phone":18643930185}
19 {"name":"Dempsey", "age":19,"sex":"女","institute":"医学院","phone":18665930185}
20 {"name":"Dennis", "age":20,"sex":"女","institute":"医学院","phone":18666930185}
21 {"name":"Derrick", "age":19,"sex":"男","institute":"医学院","phone":18673930185}
22 {"name":"Donald", "age":21,"sex":"男","institute":"医学院","phone":18683930185}
23 {"name":"Duke", "age":21,"sex":"男","institute":"医学院","phone":18693930185}
24 {"name":"Eugene", "age":19,"sex":"女","institute":"医学院","phone":18663930115}
25 {"name":"Everley", "age":20,"sex":"女","institute":"经管学院","phone":18663930125}
26 {"name":"Fabian", "age":20,"sex":"女","institute":"经管学院","phone":18663930135}
27 {"name":"Ford", "age":19,"sex":"男","institute":"经管学院","phone":18663930145}
28 {"name":"Gene", "age":20,"sex":"男","institute":"经管学院","phone":18663930155}
29 {"name":"Geoff", "age":20,"sex":"男","institute":"经管学院","phone":18663930165}
30 {"name":"Gilbert", "age":19,"sex":"男","institute":"经管学院","phone":18663930175}

```

图 5-1

JSON 文件内容简介:

正如大家所见, 该 JSON 数据源其实是一张典型的结构化学生表, 包括了 name、age、sex、institute(学院)以及 phone_num 等常见学生信息, 但值得注意的是, 该文件内容中并不含有像 SQL 文件那样介绍数据库表各个字段的表头(schema), 如图 5-2 所示, 而是如上文所述由 Spark 根据表字段类型及长度推断所得, 也正由此见证了 Spark 的强大。

```

1 create table tbl_student(
2
3 stuid varchar(20) not null,
4
5 name varchar(20)not null,
6
7 rdate date,
8
9 gender varchar(2),
10
11 institute varchar(20),
12
13 major varchar(20),
14
15 clazz varchar(10),
16
17 politics varchar(8),
18
19 nationality varchar(10),
20
21 phone varchar(20),
22
23 homeaddress varchar(100),
24
25 mail varchar(50),
26
27 phote blob,
28
29 parent varchar(100),
30
31 remark varchar(200) default '暂无',

```

图 5-2

在 spark-shell 中，读取 json 文件并转化为 DataFrame 对象：

```
import org.apache.spark.sql.Row
import org.apache.spark.sql.types._
import spark.implicits._
//读取Json数据源
val df = spark.read.json("file:///opt/people.json")
```

结果如图 5-3 所示。

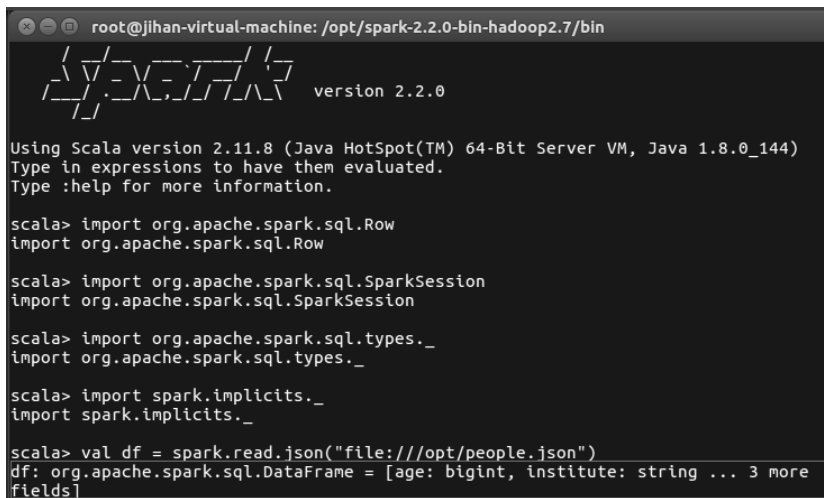


图 5-3

5.2 DataFrame 上的行动操作

通过第 3 章（RDD 编程基础）的学习，我们了解到，RDD 的操作分为两大类，转化操作和行动操作，其中转化操作实际上是逻辑分析过程的实现，但是由于惰性计算的原因，只有当行动操作出现时，才会触发真正的计算。

同样地，`DataFrame` 提供的 API 也可以采用此种分类方法，有实现逻辑的转化操作，如 `select`、`where`、`orderBy`、`groupBy` 等负责指定结果列、过滤、排序、分组的方法，和负责触发计算、回收结果的行动操作。需要注意的是，无论是直接使用 `sql()` 方法查询 `DataFrame` 注册后的表还是像本章通过在 `DataFrame` 对象上调用其提供的转化操作 API 组合出来类似的 `sql` 表达都会交由 Spark SQL 的解析、优化引擎——Catalyst 进行解析优化（Catalyst 详解见 9.7 节 Catalyst 简介），这样的底层自带优化功能的设计带给了 Spark SQL 模块使用者极大的便利，即使我们编写的 SQL 语句或者是在 `DataFrame` 对象上调用其提供的转化操作 API 组合出来类似的 `sql` 表达不够高效也可以不必担心了。

这一节，我们主要来学习 DataFrame 上引发计算，返回结果的行动操作。

1. show : 展示数据

以表格的形式在输出中展示 DF (DataFrame) 中的数据, 类似于 `select * from table_name` 的功能。

`show` 方法主要有四种调用方式, 如图 5-4 所示。

▶	<code>def show(numRows: Int, truncate: Boolean): Unit</code> Displays the Dataset in a tabular form.
▶	<code>def show(truncate: Boolean): Unit</code> Displays the top 20 rows of Dataset in a tabular form.
▶	<code>def show(): Unit</code> Displays the top 20 rows of Dataset in a tabular form.
▶	<code>def show(numRows: Int): Unit</code> Displays the Dataset in a tabular form.

图 5-4

如图所示, 在以上四个 `show()` 方法中, 关键的不同在于 `show()` 方法内的 `numRows` 和 `truncate` 参数设定。

NumRows: 即要为展示出的行数, 默认为 20 行。

Truncate: 只有两个取值 `true`、`false`, 表示一个字段是否最多显示 20 个字符, 默认为 `true`。

(1) show

只显示前 20 条记录。

示例:

```
df.show()
```

结果如图 5-5 所示。

```
root@jihon-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin
df: org.apache.spark.sql.DataFrame = [age: bigint, institute: string ... 3 more
fields]

scala> df.show()
+-----+-----+-----+-----+
|age|institute|name|phone|sex|
+-----+-----+-----+-----+
20|信息学院|Michael|18663930185|男|
21|信息学院|Andy|18663930186|女|
19|信息学院|Justin|18663930187|男|
19|信息学院|Aaron|18663930188|男|
20|信息学院|Abbott|18663930189|男|
19|信息学院|Abel|18663930180|女|
20|材料学院|Abner|18163930185|女|
21|材料学院|Abraham|18263930185|女|
19|材料学院|Adam|18363930185|男|
19|材料学院|Addison|18463930185|男|
18|材料学院|Barret|18563930185|女|
19|化工学院|Basil|18763930185|女|
20|化工学院|Beau|18863930185|女|
20|化工学院|Benedict|18963930185|男|
19|化工学院|Cedric|18603930185|男|
21|化工学院|Chad|18613930185|男|
22|化工学院|Chapman|18623930185|女|
19|化工学院|Clement|18643930185|男|
19|医学院|Dempsey|18665930185|女|
20|医学院|Dennis|18666930185|女|
+-----+-----+-----+-----+
only showing top 20 rows

scala>
```

图 5-5

可以看到在未设置 `numRows` 时，只输出了前 20 行。

(2) `show(numRows: Int)`

显示前 `numRows` 条记录。

示例：输出前 30 条数据。

```
df.show(30)
```

结果如图 5-6 所示。

```

scala> df.show(30)
+-----+-----+-----+-----+-----+
|age|institute|  name|  phone|sex|
+-----+-----+-----+-----+
|20| 信息学院| Michael|18663930185|男|
|21| 信息学院|  Andy|18663930186|女|
|19| 信息学院| Justin|18663930187|男|
|19| 信息学院|  Aaron|18663930188|男|
|20| 信息学院| Abbott|18663930189|男|
|19| 信息学院|  Abel|18663930180|女|
|20| 材料学院|  Abner|18163930185|男|
|21| 材料学院| Abraham|18263930185|女|
|19| 材料学院|  Adam|18363930185|男|
|19| 材料学院| Addison|18463930185|男|
|18| 材料学院| Barret|18563930185|女|
|19| 化工学院|  Basil|18763930185|女|
|20| 化工学院|  Beau|18863930185|女|
|20| 化工学院| Benedict|18963930185|男|
|19| 化工学院| Cedric|18603930185|男|
|21| 化工学院|  Chad|18613930185|男|
|22| 化工学院| Chapman|18623930185|女|
|19| 化工学院| Clement|18643930185|男|
|19| 医学院| Dempsey|18665930185|女|
|20| 医学院| Dennis|18666930185|女|
|19| 医学院| Derrick|18673930185|男|
|21| 医学院| Donald|18683930185|男|
|21| 医学院|  Duke|18693930185|男|
|19| 医学院| Eugene|18663930115|女|
|20| 经管学院| Everley|18663930125|女|
|20| 经管学院| Fabtan|18663930135|女|
|19| 经管学院|  Ford|18663930145|男|
|20| 经管学院|  Gene|18663930155|男|
|20| 经管学院| Geoff|18663930165|男|
|19| 经管学院| Gilbert|18663930175|男|
+-----+-----+-----+-----+
scala>

```

图 5-6

(3) `show(truncate: Boolean)`

是否最多只显示 20 个字符，默认为 `true`。

示例：

```
df.show(true)
df.show(false)
```

结果如图 5-7、图 5-8 所示。

```

root@jihan-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin

scala> df.show(true)
+-----+-----+-----+-----+-----+
|age|institute|name|phone|sex|
+-----+-----+-----+-----+
|20|信息学院|Michael|18663930185|男|
|21|信息学院|Andy|18663930186|女|
|19|信息学院|Justin|18663930187|男|
|19|信息学院|Aaron|18663930188|男|
|20|信息学院|Abbott|18663930189|男|
|19|信息学院|Abel|18663930180|女|
|20|材料学院|Abner|18163930185|男|
|21|材料学院|Abraham|18263930185|女|
|19|材料学院|Adam|18363930185|男|
|19|材料学院|Addison|18463930185|男|
|18|材料学院|Barret|18563930185|女|
|19|化工学院|Basil|18763930185|女|
|20|化工学院|Beau|18863930185|女|
|20|化工学院|Benedict|18963930185|男|
|19|化工学院|Cedric|18603930185|男|
|21|化工学院|Chad|18613930185|男|
|22|化工学院|Chapman|18623930185|女|
|19|化工学院|Clement|18643930185|男|
|19|医学院|Dempsey|18665930185|女|
|20|医学院|Dennis|18666930185|女|
+-----+-----+-----+-----+
only showing top 20 rows

```

图 5-7

```

root@jihan-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin

scala> df.show(false)
+-----+-----+-----+-----+-----+
|age|institute|name|phone|sex|
+-----+-----+-----+-----+
|20|信息学院|Michael|18663930185|男|
|21|信息学院|Andy|18663930186|女|
|19|信息学院|Justin|18663930187|男|
|19|信息学院|Aaron|18663930188|男|
|20|信息学院|Abbott|18663930189|男|
|19|信息学院|Abel|18663930180|女|
|20|材料学院|Abner|18163930185|男|
|21|材料学院|Abraham|18263930185|女|
|19|材料学院|Adam|18363930185|男|
|19|材料学院|Addison|18463930185|男|
|18|材料学院|Barret|18563930185|女|
|19|化工学院|Basil|18763930185|女|
|20|化工学院|Beau|18863930185|女|
|20|化工学院|Benedict|18963930185|男|
|19|化工学院|Cedric|18603930185|男|
|21|化工学院|Chad|18613930185|男|
|22|化工学院|Chapman|18623930185|女|
|19|化工学院|Clement|18643930185|男|
|19|医学院|Dempsey|18665930185|女|
|20|医学院|Dennis|18666930185|女|
+-----+-----+-----+-----+
only showing top 20 rows

```

图 5-8

(4) show(numRows: Int, truncate: Boolean)

综合前面的显示记录条数，以及对过长字符串的显示格式。

示例：

```
df.show(10, false)
```

结果如图 5-9 所示。

```
scala> df.show(10,false)
+---+-----+-----+-----+
|age|institute|name|phone|sex|
+---+-----+-----+-----+
|20|信息学院|Michael|18663930185|男|
|21|信息学院|Andy|18663930186|女|
|19|信息学院|Justin|18663930187|男|
|19|信息学院|Aaron|18663930188|男|
|20|信息学院|Abbott|18663930189|男|
|19|信息学院|Abel|18663930180|女|
|20|材料学院|Abner|18163930185|男|
|21|材料学院|Abraham|18263930185|女|
|19|材料学院|Adam|18363930185|男|
|19|材料学院|Addison|18463930185|男|
+---+-----+-----+-----+
only showing top 10 rows
```

图 5-9

2. collect : 获取所有数据到数组 (如图 5-10 所示)

```
def collect(): Array[T]
Returns an array that contains all rows in this Dataset.
```

图 5-10

不同于前面的 show 方法, 这里的 collect 方法会将 DF 中的所有数据都获取到, 并返回一个 Array 对象。

```
df.collect()
```

结果如图 5-11 所示, 数组中包含了 DF 的每一条记录, 每一条记录由一个 org.apache.spark.sql.Row 对象来表示, 来存储字段值。

```
scala> val dfArray = df.collect()
dfArray: Array[org.apache.spark.sql.Row] = Array([20,信息学院,Michael,18663930185,男], [21,信息学院,Andy,18663930186,女], [19,信息学院,Justin,18663930187,男], [19,信息学院,Aaron,18663930188,男], [20,信息学院,Abbott,18663930189,男], [19,信息学院,Abel,18663930180,女], [20,材料学院,Abner,18163930185,男], [21,材料学院,Abraham,18263930185,女], [19,材料学院,Adam,18363930185,男], [19,材料学院,Addison,18463930185,男], [18,材料学院,Barret,18563930185,女], [19,化工学院,Basil,18763930185,女], [20,化工学院,Beau,18863930185,女], [20,化工学院,Benedict,18963930185,男], [19,化工学院,Cedric,18603930185,男], [21,化工学院,Chad,18613930185,男], [22,化工学院,Chapman,18623930185,女], [19,化工学院,Clement,18643930185,男], [19,医学院,Dempsey,18665930185,女], [20,医学院,Dennis,18666930185,女], [19,医学院,Derrick,18673930185,男], [21,医学院,Donald,18683930185,男], [21,医学院,Duke,18693930185,男]...
```

图 5-11

3. collectAsList : 获取所有数据到 List

功能和 collect 类似, 只不过将返回结构变成了 List 对象, 不再是 Array。

使用方法如下:

```
df.collectAsList()
```

结果如图 5-12 所示。

```
scala> val dfList = df.collectAsList()
dfList: java.util.List[org.apache.spark.sql.Row] = [[20,信息学院,Michael,18663930185,男], [21,信息学院,Andy,18663930186,女], [19,信息学院,Justin,18663930187,男], [19,信息学院,Aaron,18663930188,男], [20,信息学院,Abbott,18663930189,男], [19,信息学院,Abel,18663930180,女], [20,材料学院,Abner,18163930185,男], [21,材料学院,Abraham,18263930185,女], [19,材料学院,Adam,18363930185,男], [19,材料学院,Addison,18463930185,男], [18,材料学院,Barret,18563930185,女], [19,化工学院,Basil,18763930185,女], [20,化工学院,Beau,18863930185,女], [20,化工学院,Benedict,18963930185,男], [19,化工学院,Cedric,18603930185,男], [21,化工学院,Chad,18613930185,男], [22,化工学院,Chapman,18623930185,女], [19,化工学院,Clement,18643930185,男], [19,医学院,Dempsey,18665930185,女], [20,医学院,Dennis,18666930185,女], [19,医学院,Derrick,18673930185,男], [21,医学院,Donald,18683930185,男], [21,医学院,Duke,18693930185,男]]
scala>
```

图 5-12

注意：

`collect()`和 `collectAsList()`方法，用来从 `DataFrame` 中获取整个数据集。

如果当你的程序将原始的 `DataFrame`（数据量很大）中的数据进行层层处理筛选，得到了包含着最终结果的 `DataFrame`（数据量小）并且希望从 `DataFrame` 以 `Array`、`List` 取出结果并进行下一步处理时，可以使用它。

因为这两个方法是将集群中的目标变量的所有数据取回到一个结点当中，所以当你的单台结点的内存不足以放下 `DataFrame` 中包含的数据时就会出错。因此，`collect()`、`collectAsList()` 不适用于特别大规模的数据集。

4. `describe(cols: String*)`：获取指定字段的统计信息

这个方法可以动态的传入一个或多个 `String` 类型的字段名，结果仍然为 `DataFrame` 对象，用于统计数值类型字段的统计值。

使用方法如下：

```
df.describe("age").show()
```

结果如图 5-13 所示。

```
scala> df.describe("age").show()
+-----+-----+
|summary|      age|
+-----+-----+
|  count|       30|
|   mean|19.733333333333334|
| stddev|0.9071871393197362|
|    min|        18|
|    max|        22|
+-----+-----+
scala>
```

图 5-13

在 `DataFrame` 下只需调用一个 `describe()` 子函数，即可轻松地获得以下信息：

- Count（记录条数）
- Mean（平均值）
- Stddev（样本标准差）

- Min (最小值)
- Max (最大值)

进而掌握大规模结构化数据集的某字段的统计特性。

5. first、head、take、takeAsList : 获取若干行记录

first、head、take、takeAsList 方法如图 5-14 所示。

▶	def first () : T Returns the first row.
▶	def head (n: Int): Array[T] Returns the first n rows.
▶	def take (n: Int): Array[T] Returns the first n rows in the Dataset.
▶	def takeAsList (n: Int): List[T] Returns the first n rows in the Dataset as a list.

图 5-14

first、head、take、takeAsList 用来获取部分记录，与 collect、collectAsList 获取全部记录相对应。

这里列出的四个方法比较类似，其中：

- (1) first 获取第一行记录。
- (2) head 获取第一行记录，head(n: Int)获取前 n 行记录。
- (3) take(n: Int)获取前 n 行数据。
- (4) takeAsList(n: Int)获取前 n 行数据，并以 List 的形式展现。

以 Row 或者 Array[Row]的形式返回一行或多行数据。first 和 head 功能相同。

take 和 takeAsList 方法会将获得到的数据返回到 Driver 端，所以在使用这两个方法时需要注意数据量，以免 Driver 发生 OutOfMemoryError。

first、head、take、takeAsList 方法演示如图 5-15~图 5-18 所示。

```
df.first()
```

```
scala> df.first()
res8: org.apache.spark.sql.Row = [20,信息学院,Michael,18663930185,男]
```

图 5-15

可见，first()返回的是一个表示一行的 Row 对象。

```
df.head()
```

```
df.head(10)
```

```
scala> df.head()
res9: org.apache.spark.sql.Row = [20,信息学院,Michael,18663930185,男]

scala> df.head(20)
res10: Array[org.apache.spark.sql.Row] = Array([20,信息学院,Michael,18663930185,男], [21,信息学院,Andy,18663930186,女], [19,信息学院,Justin,18663930187,男], [19,信息学院,Aaron,18663930188,男], [20,信息学院,Abbott,18663930189,男], [19,信息学院,Abel,18663930180,女], [20,材料学院,Abner,18163930185,男], [21,材料学院,Abraham,18263930185,女], [19,材料学院,Adam,18363930185,男], [19,材料学院,Addison,18463930185,男], [18,材料学院,Barret,18563930185,女], [19,化工学院,Basil,18763930185,女], [20,化工学院,Beau,18863930185,女], [20,化工学院,Benedict,18963930185,男], [19,化工学院,Cedric,18603930185,男], [21,化工学院,Chad,18613930185,男], [22,化工学院,Chapman,18623930185,女], [19,化工学院,Clement,18643930185,男], [19,医学院,Dempsey,18665930185,女], [20,医学院,Dennis,18666930185,女])
```

图 5-16

可见，`head()`返回的是一个表示一行的 `Row` 对象，而 `head(20)`返回的却是 `Array[Row]`。

```
df.take(20)
```

```
scala> df.take(20)
res11: Array[org.apache.spark.sql.Row] = Array([20,信息学院,Michael,18663930185,男], [21,信息学院,Andy,18663930186,女], [19,信息学院,Justin,18663930187,男], [19,信息学院,Aaron,18663930188,男], [20,信息学院,Abbott,18663930189,男], [19,信息学院,Abel,18663930180,女], [20,材料学院,Abner,18163930185,男], [21,材料学院,Abraham,18263930185,女], [19,材料学院,Adam,18363930185,男], [19,材料学院,Addison,18463930185,男], [18,材料学院,Barret,18563930185,女], [19,化工学院,Basil,18763930185,女], [20,化工学院,Beau,18863930185,女], [20,化工学院,Benedict,18963930185,男], [19,化工学院,Cedric,18603930185,男], [21,化工学院,Chad,18613930185,男], [22,化工学院,Chapman,18623930185,女], [19,化工学院,Clement,18643930185,男], [19,医学院,Dempsey,18665930185,女], [20,医学院,Dennis,18666930185,女])
```

图 5-17

`take(20)`返回的是 `Array[Row]`。

```
df.takeAsList(20)
```

```
scala> df.takeAsList(20)
res13: java.util.List[org.apache.spark.sql.Row] = [[20,信息学院,Michael,18663930185,男], [21,信息学院,Andy,18663930186,女], [19,信息学院,Justin,18663930187,男], [19,信息学院,Aaron,18663930188,男], [20,信息学院,Abbott,18663930189,男], [19,信息学院,Abel,18663930180,女], [20,材料学院,Abner,18163930185,男], [21,材料学院,Abraham,18263930185,女], [19,材料学院,Adam,18363930185,男], [19,材料学院,Addison,18463930185,男], [18,材料学院,Barret,18563930185,女], [19,化工学院,Basil,18763930185,女], [20,化工学院,Beau,18863930185,女], [20,化工学院,Benedict,18963930185,男], [19,化工学院,Cedric,18603930185,男], [21,化工学院,Chad,18613930185,男], [22,化工学院,Chapman,18623930185,女], [19,化工学院,Clement,18643930185,男], [19,医学院,Dempsey,18665930185,女], [20,医学院,Dennis,18666930185,女]]
```

图 5-18

与 `take(20)`返回 `Array[Row]`不同的是 `takeAsList(20)`返回的是 `List[Row]`。

5.3 DataFrame 上的转化操作

本节主要介绍 `DataFrame` 所提供用以形成 `SQL` 表达的转化操作，如 `select()`、`where()`、

orderBy()、groupBy()、join()等方法。以下方法皆为返回 DataFrame 对象的方法，所以可以连续调用。

5.3.1 where 条件相关

如图 5-19 所示，where 方法根据参数类型及数目不同进行了同名函数重载，可以看到第一个 where(conditionExpr: String) 输入更像一种传统 SQL 的 where 子句的条件整体描述，而 where(condition: Column)，该方法的输入则是要把 where 子句的对于每一个 column 的要求进行分别描述，且该种表述等效于 filter() 实现的筛选，但从最终效果上来讲，这两种方法并没有什么不同，只是解析语句时，第一种方法，需要对整个 where 子句进行解析，从而得到对于每一个 column 的要求。

<pre>def where(conditionExpr: String): Dataset[T] Filters rows using the given SQL expression.</pre>	
<pre>peopleDs.where("age > 15")</pre>	
Since	1.6.0
<pre>def where(condition: Column): Dataset[T] Filters rows using the given condition. This is an alias for filter.</pre>	
<pre>// The following are equivalent: peopleDs.filter(\$"age" > 15) peopleDs.where(\$"age" > 15)</pre>	
Since	1.6.0

图 5-19

(1) where(conditionExpr: String): SQL 语言中 where 关键字后的条件传入筛选条件表达式，可以用 and 和 or，得到 DataFrame 类型的返回结果。示例（见图 5-20~图 5-22）：

```
df.where("name = 'Michael' or age = 19").show()
```

```
scala> df.where("name = 'Michael' or age = 19").show()
+-----+-----+-----+-----+
|age|institute|  name|  phone|sex|
+-----+-----+-----+-----+
| 20| 信息学院|Michael|18663930185|男|
| 19| 信息学院|Justin|18663930187|男|
| 19| 信息学院|Aaron|18663930188|男|
| 19| 信息学院|Abel|18663930180|女|
| 19| 材料学院|Adam|18363930185|男|
| 19| 材料学院|Addison|18463930185|男|
| 19| 化工学院|Basil|18763930185|女|
| 19| 化工学院|Cedric|18603930185|男|
| 19| 化工学院|Clement|18643930185|男|
| 19| 医学院|Dempsey|18665930185|女|
| 19| 医学院|Derrick|18673930185|男|
| 19| 医学院|Eugene|18663930115|女|
| 19| 经管学院|Ford|18663930145|男|
| 19| 经管学院|Gilbert|18663930175|男|
+-----+-----+-----+-----+
```

图 5-20

```
df.where("name = 'Michael' and age = 20").show()
```

```
scala> df.where("name = 'Michael' and age = 20").show()
+---+-----+-----+-----+---+
|age|institute|  name|    phone|sex|
+---+-----+-----+-----+---+
| 20|   信息学院|Michael|18663930185|男|
+---+-----+-----+-----+---+
```

图 5-21

```
df.where($"age">20).show()
```

```
scala> df.where($"age">20).show()
+---+-----+-----+-----+---+
|age|institute|  name|    phone|sex|
+---+-----+-----+-----+---+
| 21|   信息学院|  Andy|18663930186|女|
| 21|   材料学院|Abraham|18263930185|女|
| 21|   化工学院|  Chad|18613930185|男|
| 22|   化工学院|Chapman|18623930185|女|
| 21|   医学院|Donald|18683930185|男|
| 21|   医学院|  Duke|18693930185|男|
+---+-----+-----+-----+---+
```

图 5-22

需要注意的是，在该示例中，使用 `"age"` 提取 `age` 列数据作比较时，用到了隐式转换，故需在程序中引入相应包（`import spark.implicits._`）。

（2）filter：根据字段进行筛选

如图 5-23 所示，`filter()` 同样具有两个同名重载函数 `filter(conditionExpr: String)`、`filter(condition: Column)`，其间区分差不多 `where()` 情况相同，即其两者效果等效，仅为了满足程序员的不同开发习惯。

<pre>def filter(conditionExpr: String): Dataset[T] Filters rows using the given SQL expression.</pre>	
<pre>peopleDs.filter("age > 15")</pre>	
Since	1.6.0
<pre>def filter(condition: Column): Dataset[T] Filters rows using the given condition.</pre>	
<pre>// The following are equivalent: peopleDs.filter(\$"age" > 15) peopleDs.where(\$"age" > 15)</pre>	
Since	1.6.0

图 5-23

传入筛选条件表达式，得到 `DataFrame` 类型的返回结果，和 `where` 使用条件相同。

示例（见图 5-24~图 5-25）：

```
df.filter("name = 'Michael' or age = 20").show()
```

```
scala> df.filter("name = 'Michael' or age = 20").show()
+-----+-----+-----+-----+
|age|institute|name|phone|sex|
+-----+-----+-----+-----+
|20|信息学院|Michael|18663930185|男|
|20|信息学院|Abbott|18663930189|男|
|20|材料学院|Abner|18163930185|男|
|20|化工学院|Beau|18863930185|女|
|20|化工学院|Benedict|18963930185|男|
|20|医学院|Dennis|18666930185|女|
|20|经管学院|Everley|18663930125|女|
|20|经管学院|Fabian|18663930135|女|
|20|经管学院|Gene|18663930155|男|
|20|经管学院|Geoff|18663930165|男|
+-----+-----+-----+-----+
```

图 5-24

可见，与 `df.where("name = 'Michael' or age = 20").show()` 等效。

```
df.filter($"age" > 20).show()
```

```
scala> df.filter($"age" > 20).show()
+-----+-----+-----+-----+
|age|institute|name|phone|sex|
+-----+-----+-----+-----+
|21|信息学院|Andy|18663930186|女|
|21|材料学院|Abraham|18263930185|女|
|21|化工学院|Chad|18613930185|男|
|22|化工学院|Chapman|18623930185|女|
|21|医学院|Donald|18683930185|男|
|21|医学院|Duke|18693930185|男|
+-----+-----+-----+-----+
scala>
```

图 5-25

5.3.2 查询指定列

(1) `select`: 获取指定字段值（见图 5-26）

```
def select(col: String, cols: String*): DataFrame
  Selects a set of columns. This is a variant of select that can only select existing columns
  using column names (i.e. cannot construct expressions).

  // The following two are equivalent:
  ds.select("colA", "colB")
  ds.select($"colA", $"colB")

  Annotations    @varargs()
  Since          2.0.0
```

图 5-26

根据传入的 `String` 类型字段名，获取指定字段的值，以 `DataFrame` 类型返回。

示例（见图 5-27）：

```
df.select("name", "age").show(false)
```

```

root@jihan-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin
scala> df.select("name","age").show()
+-----+
| name | age |
+-----+
| Michael | 20 |
| Andy | 21 |
| Justin | 19 |
| Aaron | 19 |
| Abbott | 20 |
| Abel | 19 |
| Abner | 20 |
| Abraham | 21 |
| Adam | 19 |
| Addison | 19 |
| Barret | 18 |
| Basil | 19 |
| Beau | 20 |
| Benedict | 20 |
| Cedric | 19 |
| Chad | 21 |
| Chapman | 22 |
| Clement | 19 |
| Dempsey | 19 |
| Dennis | 20 |
+-----+
only showing top 20 rows

```

图 5-27

还有一个重载的 select 方法，不是传入 String 类型参数，而是传入 Column 类型参数。可以实现 select id, id+1 from test 这种逻辑。

```

def select(cols: Column*): DataFrame
  Selects a set of column based expressions.

```

```

ds.select($"colA", $"colB" + 1)

```

Annotations @varargs()

Since 2.0.0

```

df.select(df("name"),df("institute"), df("age") +

```

(2) show(false) (见图 5-28)

```

root@jihan-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin
scala> df.select(df("name"),df("institute"),df("age")+1).show()
+-----+-----+-----+
| name | institute | (age + 1) |
+-----+-----+-----+
| Michael | 信息学院 | 21 |
| Andy | 信息学院 | 22 |
| Justin | 信息学院 | 20 |
| Aaron | 信息学院 | 20 |
| Abbott | 信息学院 | 21 |
| Abel | 信息学院 | 20 |
| Abner | 材料学院 | 21 |
| Abraham | 材料学院 | 22 |
| Adam | 材料学院 | 20 |
| Addison | 材料学院 | 20 |
| Barret | 材料学院 | 19 |
| Basil | 化工学院 | 20 |
| Beau | 化工学院 | 21 |
| Benedict | 化工学院 | 21 |
| Cedric | 化工学院 | 20 |
| Chad | 化工学院 | 22 |
| Chapman | 化工学院 | 23 |
| Clement | 化工学院 | 20 |
| Dempsey | 医学院 | 20 |
| Dennis | 医学院 | 21 |
+-----+-----+-----+
only showing top 20 rows

scala>

```

图 5-28

能得到 Column 类型的方法是 `apply` 以及 `col` 方法，一般用 `apply` 方法更简便。

(3) `selectExpr`: 可以对指定字段进行特殊处理（见图 5-29）

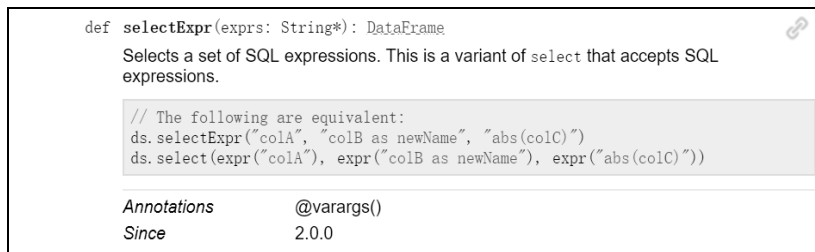


图 5-29

可以直接对指定字段调用 UDF 函数，或者指定别名等。传入 `String` 类型参数，得到 `DataFrame` 对象。

示例：查询 `id` 字段，`c3` 字段取别名 `time`，`c4` 字段四舍五入。（见图 5-30~图 5-31）

```
Df.selectExpr("name", "institute as 'xueyuan'", "round(age)").show()
```

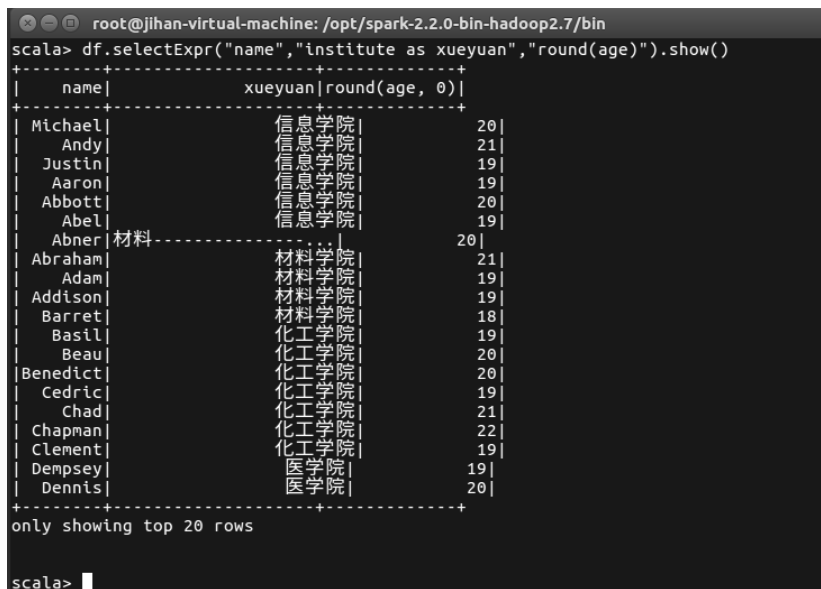


图 5-30

```
df.select(expr("name"),expr("institute as xueyuan"),expr("age")).show()
```

```

root@jihan-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin
scala> df.select(expr("name"),expr("institute as xueyuan"),expr("age")).show()
+-----+-----+-----+
| name | xueyuan | age |
+-----+-----+-----+
| Michael | 信息学院 | 20 |
| Andy | 信息学院 | 21 |
| Justin | 信息学院 | 19 |
| Aaron | 信息学院 | 19 |
| Abbott | 信息学院 | 20 |
| Abel | 信息学院 | 19 |
| Abner | 材料学院 | 20 |
| Abraham | 材料学院 | 21 |
| Adam | 材料学院 | 19 |
| Addison | 材料学院 | 19 |
| Barret | 材料学院 | 18 |
| Basil | 化工学院 | 19 |
| Beau | 化工学院 | 20 |
| Benedict | 化工学院 | 20 |
| Cedric | 化工学院 | 19 |
| Chad | 化工学院 | 21 |
| Chapman | 化工学院 | 22 |
| Clement | 化工学院 | 19 |
| Dempsey | 医学院 | 19 |
| Dennis | 医学院 | 20 |
+-----+-----+-----+
only showing top 20 rows

scala>

```

图 5-31

```

ds.selectExpr("colA", "colB as newName", "abs(colC)")
ds.select(expr("colA"), expr("colB as newName"), expr("abs(colC)"))

```

可以看出，以上两种方法是等效的，使用时根据程序员编程习惯而定。

(4) col: 获取指定字段（见图 5-32、图 5-33）

```

def col(colName: String): Column
    Selects column based on the column name and return it as a Column.

```

图 5-32

只能获取一个字段，返回对象为 Column 类型。

```
val nameCol = df.col("name")
```

```

scala> val nameCol = df.col("name")
nameCol: org.apache.spark.sql.Column = name

```

图 5-33

(5) apply: 获取指定列（见图 5-34）

```

def apply(colName: String): Column
    Selects column based on the column name and return it as a Column.

def col(colName: String): Column
    Selects column based on the column name and return it as a Column.

```

图 5-34

由图 5-34 可见，apply()与 col()参数类型、个数以及返回值类型均相同，只能获取某一列，返回对象为 Column 类型。

示例（见图 5-35）：


```
val ageCol1 = df.apply("age")
val ageCol2 = df("age")
```

```
scala> val ageCol1 = df.apply("age")
ageCol1: org.apache.spark.sql.Column = age

scala> val ageCol2 = df("age")
ageCol2: org.apache.spark.sql.Column = age
```

图 5-35

上述两种获取 column 的方法等效，返回的皆为对应 Column。

(6) drop: 去除指定字段，保留其他字段（见图 5-36）

```
def drop(colName: String): DataFrame
Returns a new Dataset with a column dropped.
```

```
def drop(col: Column): DataFrame
Returns a new Dataset with a column dropped.
```

图 5-36

以上两种 drop 重载函数，不同在于，前者输入参数是描述列名称的 String，而后者传入的是 Column 类型的列。

返回一个新的 DataFrame 对象，其中不包含去除的字段，一次只能去除一个字段。

示例（见图 5-37~图 5-38）：

```
df.drop("id")
df.drop(df("id"))
```

```
root@jjhan-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin
fields]
scala> df2.show()
+-----+-----+-----+-----+-----+
|age|institute|name|phone|sex|
+-----+-----+-----+-----+
|20|信息学院|Michael|18663930185|男|
|21|信息学院|Andy|18663930186|女|
|19|信息学院|Justin|18663930187|男|
|19|信息学院|Aaron|18663930188|男|
|20|信息学院|Abbott|18663930189|男|
|19|信息学院|Abel|18663930180|女|
|20|材料学院|Abner|18163930185|男|
|21|材料学院|Abraham|18263930185|女|
|19|材料学院|Adam|18363930185|男|
|19|材料学院|Addison|18463930185|男|
|18|材料学院|Barret|18563930185|女|
|19|化工学院|Basil|18763930185|女|
|20|化工学院|Beau|18863930185|女|
|20|化工学院|Benedict|18963930185|男|
|19|化工学院|Cedric|18603930185|男|
|21|化工学院|Chad|18613930185|男|
|22|化工学院|Chapman|18623930185|女|
|19|化工学院|Clement|18643930185|男|
|19|医学院|Dempsey|18665930185|女|
|20|医学院|Dennis|18666930185|女|
+-----+-----+-----+-----+
only showing top 20 rows
```

图 5-37

```

root@jihon-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin
scala> df2.drop("name").show()
+----+-----+-----+----+
|age|      institute|    phone|sex|
+----+-----+-----+----+
|20|      信息学院|18663930185|男|
|21|      信息学院|18663930186|女|
|19|      信息学院|18663930187|男|
|19|      信息学院|18663930188|男|
|20|      信息学院|18663930189|男|
|19|      信息学院|18663930180|女|
|20| 材料-----|18163930185|男|
|21|      材料学院|18263930185|女|
|19|      材料学院|18363930185|男|
|19|      材料学院|18463930185|男|
|18|      材料学院|18563930185|女|
|19|      化工学院|18763930185|女|
|20|      化工学院|18863930185|女|
|20|      化工学院|18963930185|男|
|19|      化工学院|18603930185|男|
|21|      化工学院|18613930185|男|
|22|      化工学院|18623930185|女|
|19|      化工学院|18643930185|男|
|19|      医学院|18665930185|女|
|20|      医学院|18666930185|女|
+----+-----+-----+----+
only showing top 20 rows

```

图 5-38

drop() 传入 Column 类型的列，如图 5-39 所示。

```

root@jihon-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin
scala> df2.drop(df2("name")).show()
+----+-----+-----+----+
|age|      institute|    phone|sex|
+----+-----+-----+----+
|20|      信息学院|18663930185|男|
|21|      信息学院|18663930186|女|
|19|      信息学院|18663930187|男|
|19|      信息学院|18663930188|男|
|20|      信息学院|18663930189|男|
|19|      信息学院|18663930180|女|
|20| 材料-----|18163930185|男|
|21|      材料学院|18263930185|女|
|19|      材料学院|18363930185|男|
|19|      材料学院|18463930185|男|
|18|      材料学院|18563930185|女|
|19|      化工学院|18763930185|女|
|20|      化工学院|18863930185|女|
|20|      化工学院|18963930185|男|
|19|      化工学院|18603930185|男|
|21|      化工学院|18613930185|男|
|22|      化工学院|18623930185|女|
|19|      化工学院|18643930185|男|
|19|      医学院|18665930185|女|
|20|      医学院|18666930185|女|
+----+-----+-----+----+
only showing top 20 rows

```

图 5-39

5.3.3 思维开拓：Column 的巧妙应用

经过上一小节的学习，可以看出，col()、apply()、drop()均是对 DataFrame 的某一列进行操作，以下通过一个生活中常见的小应用来看看 Column 操作的灵活巧妙之处。

1. 实例描述

国家体质健康网最近已经统计得到了一张记录着全国学生身高体重以及相关体质信息的数据表（近千万条数据），然而如此居大的数据量，已远远超出某台计算机的计算、存储能力，

因此，我们采用在 Spark 集群之上，采用 Spark SQL 快速处理结构化数据集，进而利用数百台 Spark 处理节点在十分钟之内快速准确地得到目标结果详细统计表结构如图 5-40 所示。

```
scala> phyReport.select("name","height","weight","phone").show()
+-----+-----+-----+-----+
| name|height|weight|  phone|
+-----+-----+-----+-----+
| Michael| 180| 140|18663930185|
| Andy| 180| 200|18663930186|
| Justin| 170| 140|18663930187|
| Aaron| 170| 140|18663930188|
| Abbott| 170| 90|18663930189|
| Abel| 170| 90|18663930180|
| Abner| 170| 140|18163930185|
| Abraham| 170| 91|18263930185|
| Adam| 170| 120|18363930185|
| Addison| 170| 110|18463930185|
| Barret| 170| 80|18563930185|
| Basil| 170| 160|18763930185|
| Beau| 170| 170|18863930185|
| Benedict| 170| 120|18963930185|
| Cedric| 170| 130|18603930185|
+-----+-----+-----+-----+
```

图 5-40

已知信息是每个人的身高、体重，如何较为科学地并且简单地评价一个人的身体状况呢？

身高体重指数（BMI）是体脂指标，它是基于身高体重进行计算，可以反映出体重是否偏轻、正常、偏重、肥胖，它有助于评估体脂增加可能诱发疾病的风险。

因此，本示例的需求，即是将在录学生的身高体重做除法，进而根据所得值判断每一位学生体重是否偏轻、正常、偏重、肥胖，进而查询其联系电话，每个月定时向体质偏轻、偏胖、肥胖的用户发送短信，给出提醒建议，进而促进全国大学生的身体健康。

2. 模拟实现

(1) 首先从原始数据表（phyReport）取出学生体重身高关键信息，得到 phyInfo 表，如图 5-41、图 5-42 所示。

```
scala> val phyInfo = phyReport.select("name","height","weight","phone")
phyInfo: org.apache.spark.sql.DataFrame = [name: string, height: bigint ... 2 more fields]
```

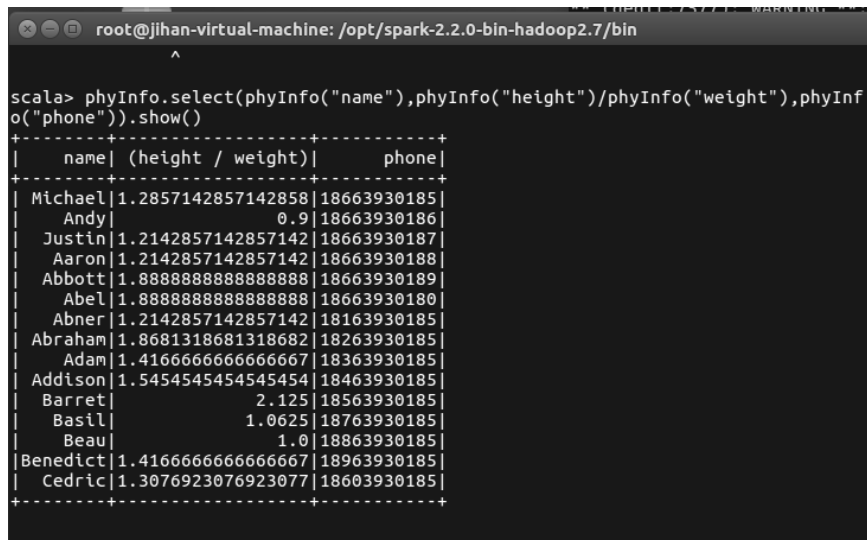
图 5-41

```
scala> phyInfo.show()
+-----+-----+-----+-----+
| name|height|weight|  phone|
+-----+-----+-----+-----+
| Michael| 180| 140|18663930185|
| Andy| 180| 200|18663930186|
| Justin| 170| 140|18663930187|
| Aaron| 170| 140|18663930188|
| Abbott| 170| 90|18663930189|
| Abel| 170| 90|18663930180|
| Abner| 170| 140|18163930185|
| Abraham| 170| 91|18263930185|
| Adam| 170| 120|18363930185|
| Addison| 170| 110|18463930185|
| Barret| 170| 80|18563930185|
| Basil| 170| 160|18763930185|
| Beau| 170| 170|18863930185|
| Benedict| 170| 120|18963930185|
| Cedric| 170| 130|18603930185|
+-----+-----+-----+-----+
```

图 5-42

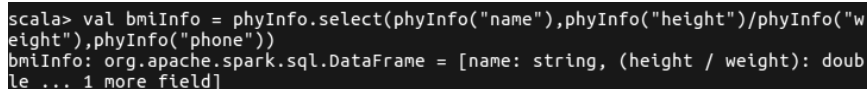
(2) 对 phyInfo 表的两列 height、weight 做除法, 得到所有人体质指数, 并抽取相关联系信息列, 得到 (bmiInfo) 身高体重指数信息表, 如图 5-43、图 5-44 所示。

```
val BMIInfo =
phyInfo.select(phyInfo("name"), (phyInfo("height")/phyInfo("weight")).as("bmi"),
phyInfo("phone")).show()
```



```
scala> phyInfo.select(phyInfo("name"),phyInfo("height")/phyInfo("weight"),phyInfo("phone")).show()
+-----+-----+-----+
| name| (height / weight)| phone|
+-----+-----+-----+
| Michael|1.2857142857142858|18663930185|
| Andy|0.9|18663930186|
| Justin|1.2142857142857142|18663930187|
| Aaron|1.2142857142857142|18663930188|
| Abbott|1.8888888888888888|18663930189|
| Abel|1.8888888888888888|18663930180|
| Abner|1.2142857142857142|18163930185|
| Abraham|1.8681318681318682|18263930185|
| Adam|1.4166666666666667|18363930185|
| Addison|1.5454545454545454|18463930185|
| Barret|2.125|18563930185|
| Basil|1.0625|18763930185|
| Beau|1.0|18863930185|
| Benedict|1.4166666666666667|18963930185|
| Cedric|1.3076923076923077|18603930185|
+-----+-----+-----+
```

图 5-43

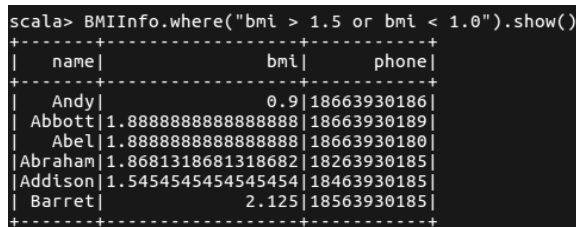


```
scala> val bmiInfo = phyInfo.select(phyInfo("name"),phyInfo("height")/phyInfo("weight"),phyInfo("phone"))
bmiInfo: org.apache.spark.sql.DataFrame = [name: string, (height / weight): double ... 1 more field]
```

图 5-44

(3) 对身高体重指标表中不合格学生进行筛选, 如图 5-45 所示。

```
BMIInfo.where("bmi > 1.5 or bmi < 1.0").show()
```



```
scala> BMIInfo.where("bmi > 1.5 or bmi < 1.0").show()
+-----+-----+-----+
| name| bmi| phone|
+-----+-----+-----+
| Andy|0.9|18663930186|
| Abbott|1.8888888888888888|18663930189|
| Abel|1.8888888888888888|18663930180|
| Abraham|1.8681318681318682|18263930185|
| Addison|1.5454545454545454|18463930185|
| Barret|2.125|18563930185|
+-----+-----+-----+
```

图 5-45

(4) 系统每月根据电话号码对该部分学生进行身体健康建议推送短信 (见图 5-46、图 5-47)。

总结: `col` 还可以进行许多其他运算, 详细内容可见官方文档。

<http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.Column>

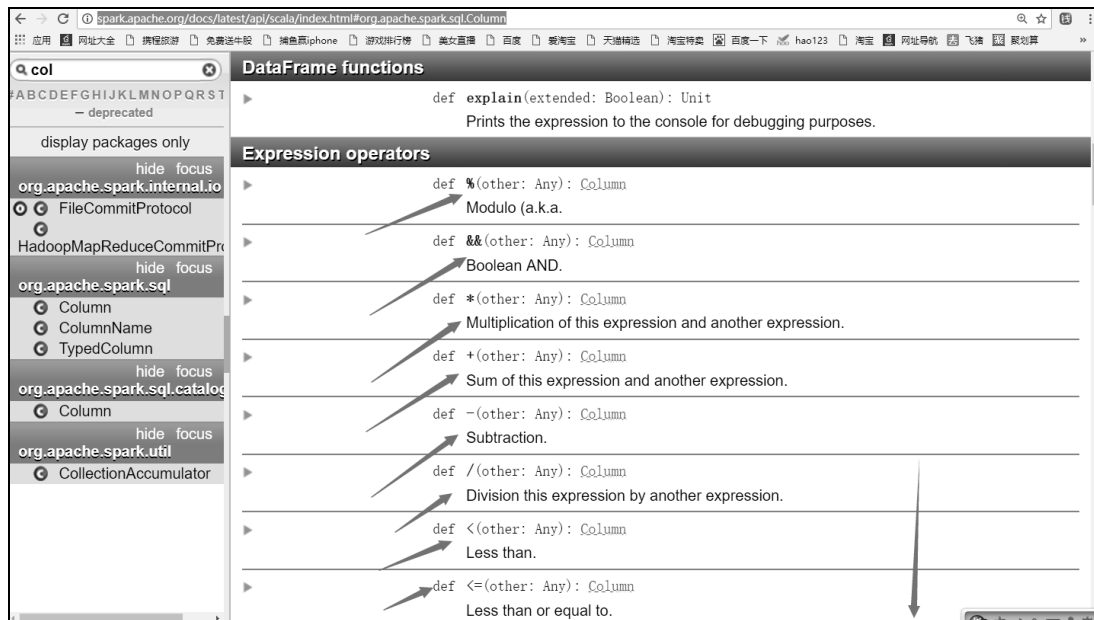


图 5-46

5.3.4 limit 操作

`limit` 方法获取指定 `DataFrame` 的前 `n` 行记录, 得到一个新的 `DataFrame` 对象, 如图 5-47 所示。

```
def limit(n: Int): Dataset[T]
Returns a new Dataset by taking the first n rows.
```

图 5-47

和 `take` 与 `head` 不同的是, `limit` 方法不是 Action 操作, 因为 `take`、`head` 均获得的均为 `Array`(数组), 而 `limit` 返回的一个新的转化生成的 `DataFrame` 对象 (见图 5-48)。

```
df.limit(20).show()
```

```
root@jihan-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin

scala> df.limit(20).show()
+-----+-----+-----+-----+-----+
|age|      institute|   name|   phone|sex|
+-----+-----+-----+-----+-----+
|20|      信息学院| Michael|18663930185|男|
|21|      信息学院|   Andy|18663930186|女|
|19|      信息学院|  Justin|18663930187|男|
|19|      信息学院|   Aaron|18663930188|男|
|20|      信息学院| Abbott|18663930189|女|
|19|      信息学院|   Abel|18663930180|女|
|20| 材料-----+-----+-----+-----+-----+
|21|      材料学院| Abraham|18263930185|女|
|19|      材料学院|   Adam|18363930185|男|
|19|      材料学院| Addison|18463930185|男|
|18|      材料学院| Barret|18563930185|女|
|19|      化工学院|   Basil|18763930185|女|
|20|      化工学院|   Beau|18863930185|女|
|20|      化工学院| Benedict|18963930185|男|
|19|      化工学院| Cedric|18603930185|男|
|21|      化工学院|   Chad|18613930185|男|
|22|      化工学院| Chapman|18623930185|女|
|19|      化工学院| Clement|18643930185|男|
|19|      医学院| Dempsey|18665930185|女|
|20|      医学院|  Dennis|18666930185|女|
+-----+-----+-----+-----+-----+
```

图 5-48

5.3.5 排序操作：order by 和 sort

order by 和 sort 方法如图 5-49 所示。

def orderBy (sortExprs: Column*): Dataset[T] Returns a new Dataset sorted by the given expressions.	
def orderBy (sortCol: String, sortCols: String*): Dataset[T] Returns a new Dataset sorted by the given expressions.	
def sort (sortExprs: Column*): Dataset[T] Returns a new Dataset sorted by the given expressions. For example: <div>ds.sort(\$"col1", \$"col2".desc)</div>	
Annotations	@varargs()
Since	2.0.0
def sort (sortCol: String, sortCols: String*): Dataset[T] Returns a new Dataset sorted by the specified column, all in ascending order. <div>// The following 3 are equivalent ds.sort("sortcol") ds.sort(\$"sortcol") ds.sort(\$"sortcol".asc)</div>	

图 5-49

(1) orderBy 和 sort: 按指定字段排序，默认为升序
按指定字段排序。在 Column 后面加.desc 表示降序排序,加.asc 表示升序排序。sort 和

orderBy 使用方法相同，可参看图 5-50、图 5-51。

```
df.orderBy(df("age").asc).show(false)
```

```

root@jihon-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin
scala> df.orderBy(df("age").asc).show(false)
+-----+-----+-----+-----+
|age|institute|name|phone|sex|
+-----+-----+-----+-----+
|18|材料学院|Barret|18563930185|女|
|19|信息学院|Justin|18663930187|男|
|19|材料学院|Addison|18463930185|男|
|19|信息学院|Aaron|18663930188|男|
|19|信息学院|Abel|18663930180|女|
|19|医学院|Derrick|18673930185|男|
|19|医学院|Eugene|18663930115|女|
|19|经管学院|Ford|18663930145|男|
|19|经管学院|Gilbert|18663930175|男|
|19|化工学院|Basil|18763930185|女|
|19|化工学院|Cedric|18603930185|男|
|19|化工学院|Clement|18643930185|男|
|19|医学院|Dempsey|18665930185|女|
|19|材料学院|Adam|18363930185|男|
|20|信息学院|Michael|18663930185|男|
|20|信息学院|Abbott|18663930189|男|
|20|材料学院|Abner|18163930185|男|
|20|医学院|Dennis|18666930185|女|
|20|化工学院|Beau|18863930185|女|
|20|经管学院|Everley|18663930125|女|
+-----+-----+-----+-----+
only showing top 20 rows

```

图 5-50

```
df.orderBy(df("age").desc).show(false)
```

```

root@jihon-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin
scala> df.orderBy(df("age").desc).show(false)
+-----+-----+-----+-----+
|age|institute|name|phone|sex|
+-----+-----+-----+-----+
|22|化工学院|Chapman|18623930185|女|
|21|医学院|Duke|18693930185|男|
|21|信息学院|Andy|18663930186|女|
|21|材料学院|Abraham|18263930185|女|
|21|化工学院|Chad|18613930185|男|
|21|医学院|Donald|18683930185|男|
|20|医学院|Dennis|18666930185|女|
|20|经管学院|Everley|18663930125|女|
|20|经管学院|Fabian|18663930135|女|
|20|材料学院|Abner|18163930185|男|
|20|信息学院|Michael|18663930185|男|
|20|化工学院|Benedict|18963930185|男|
|20|经管学院|Gene|18663930155|男|
|20|经管学院|Geoff|18663930165|男|
|20|信息学院|Abbott|18663930189|女|
|20|化工学院|Beau|18863930185|女|
|19|信息学院|Justin|18663930187|男|
|19|信息学院|Aaron|18663930188|男|
|19|信息学院|Abel|18663930180|女|
|19|材料学院|Adam|18363930185|男|
+-----+-----+-----+-----+
only showing top 20 rows

```

图 5-51

orderBy() 也支持对多个字段进行排序，如图 5-52 所示。

```
df.orderBy(df("age").asc, df("name")).show(false)
```

```

root@jihan-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin
scala> df.orderBy(df("age").asc,df("name")).show(false)
+---+
|age|institute|name|phone|sex|
+---+
|18|材料学院|Barret|18563930185|女|
|19|信息学院|Aaron|18663930188|男|
|19|信息学院|Abel|18663930180|女|
|19|材料学院|Adam|18363930185|男|
|19|材料学院|Addison|18463930185|男|
|19|化工学院|Basil|18763930185|女|
|19|化工学院|Cedric|18603930185|男|
|19|化工学院|Clement|18643930185|男|
|19|医学院|Dempsey|18665930185|女|
|19|医学院|Derrick|18673930185|男|
|19|医学院|Eugene|18663930115|女|
|19|经管学院|Ford|18663930145|男|
|19|经管学院|Gilbert|18663930175|男|
|19|信息学院|Justin|18663930187|男|
|20|信息学院|Abbott|18663930189|男|
|20|材料学院|Abner|18163930185|男|
|20|化工学院|Beau|18863930185|女|
|20|化工学院|Benedict|18963930185|男|
|20|医学院|Dennis|18666930185|女|
|20|经管学院|Everley|18663930125|女|
+---+
only showing top 20 rows

```

图 5-52

sort() 方法和 orderBy() 方法用法一致，效果等效，如图 5-53 所示。

```

root@jihan-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin
scala> df.sort(df("age").asc,df("name")).show(false)
+---+
|age|institute|name|phone|sex|
+---+
|18|材料学院|Barret|18563930185|女|
|19|信息学院|Aaron|18663930188|男|
|19|信息学院|Abel|18663930180|女|
|19|材料学院|Adam|18363930185|男|
|19|材料学院|Addison|18463930185|男|
|19|化工学院|Basil|18763930185|女|
|19|化工学院|Cedric|18603930185|男|
|19|化工学院|Clement|18643930185|男|
|19|医学院|Dempsey|18665930185|女|
|19|医学院|Derrick|18673930185|男|
|19|医学院|Eugene|18663930115|女|
|19|经管学院|Ford|18663930145|男|
|19|经管学院|Gilbert|18663930175|男|
|19|信息学院|Justin|18663930187|男|
|20|信息学院|Abbott|18663930189|男|
|20|材料学院|Abner|18163930185|男|
|20|化工学院|Beau|18863930185|女|
|20|化工学院|Benedict|18963930185|男|
|20|医学院|Dennis|18666930185|女|
|20|经管学院|Everley|18663930125|女|
+---+
only showing top 20 rows

```

图 5-53

(2) sortWithinPartitions (见图 5-54)

<pre>def sortWithinPartitions(sortExprs: Column*): Dataset[T]</pre> <p>Returns a new Dataset with each partition sorted by the given expressions.</p>
<pre>def sortWithinPartitions(sortCol: String, sortCols: String*): Dataset[T]</pre> <p>Returns a new Dataset with each partition sorted by the given expressions.</p>

图 5-54

和上面的 sort 方法功能类似，区别在于 sortWithinPartitions 方法返回的是排好序的每一个 Partition 的 DataFrame 对象。

5.3.6 group by 操作

group by 方法如图 5-55 所示。


<pre>def groupBy(coll: String, cols: String*): RelationalGroupedDataset</pre> <p>Groups the Dataset using the specified columns, so that we can run aggregation on them.</p>	
<pre>def groupBy(cols: Column*): RelationalGroupedDataset</pre> <p>Groups the Dataset using the specified columns, so we can run aggregation on them.</p>	

图 5-55

(1) groupBy: 根据字段进行 group by 操作

groupBy 方法有两种调用方式，可以传入 String 类型的字段名，也可传入 Column 类型的对象。

使用方法如下：

```
Df.groupBy("institute")
df.groupBy(df("institute"))
```

(2) RelationalGroupedDataset 对象

groupBy()方法得到的是 RelationalGroupedDataset 类型对象，在 RelationalGroupedDataset 的 API 中提供了 group by 之后的操作，比如：

- max(colNames: String*)方法，获取分组中指定字段或者所有的数字类型字段的最大值，只能作用于数字型字段。
- min(colNames: String*)方法，获取分组中指定字段或者所有的数字类型字段的最小值，只能作用于数字型字段。
- mean(colNames: String*)方法，获取分组中指定字段或者所有的数字类型字段的平均值，只能作用于数字型字段。
- sum(colNames: String*)方法，获取分组中指定字段或者所有的数字类型字段的和值，只能作用于数字型字段。
- Count()方法，获取分组中的元素个数。

运行结果说明如下。

统计所有学生中不同年龄数目，如图 5-56 所示。

```
df.groupBy("age").count().show()
```

```
scala> df.groupBy("age").count().show()
+---+-----+
|age|count|
+---+-----+
| 19|    13|
| 22|     1|
| 18|     1|
| 21|     5|
| 20|    10|
+---+-----+
```

图 5-56

统计各学院学生的学生平均年龄，如图 5-57 所示。

```
df.groupBy("institute").mean("age").show()
```

```
scala> df.groupBy("institute").mean("age").show()
+-----+-----+
|institute|avg(age)|
+-----+-----+
|医学院|19.833333333333332|
|材料|20.0|
|材料学院|19.25|
|化工学院|20.0|
|信息学院|19.666666666666668|
|经管学院|19.666666666666668|
+-----+-----+
```

图 5-57

统计各学院男女学生的学生平均年龄，如图 5-58 所示。

```
df.groupBy("institute","sex").mean("age").show()
```

```
scala> df.groupBy("institute","sex").mean("age").show()
+-----+-----+
|institute|sex|avg(age)|
+-----+-----+
|医学院|男|20.333333333333332|
|化工学院|男|19.75|
|材料学院|女|19.5|
|化工学院|女|20.333333333333332|
|经管学院|男|19.5|
|材料学院|男|19.0|
|信息学院|男|19.5|
|材料|男|20.0|
|医学院|女|19.333333333333332|
|经管学院|女|20.0|
|信息学院|女|20.0|
+-----+-----+
```

图 5-58

5.3.7 distinct、dropDuplicates 去重操作

distinct、dropDuplicates 方法如图 5-59 所示。

```
def distinct(): Dataset[T]
```

Returns a new Dataset that contains only the unique rows from this Dataset.

```
def dropDuplicates(): Dataset[T]
```

Returns a new Dataset that contains only the unique rows from this Dataset.

图 5-59

(1) **distinct**: 返回一个不包含重复记录的 DataFrame

返回当前 DataFrame 中不重复的 Row 记录。该方法和接下来的 `dropDuplicates()` 方法不传入指定字段时的结果相同。

示例:

在去重操作 `distinct()` 之前有重复记录, 如图 5-60 所示。

```
scala> df.show(5)
+---+-----+-----+-----+
|age|institute| name|      phone|sex|
+---+-----+-----+-----+
| 20| 信息学院|Michael|18663930185|男|
| 20| 信息学院|Michael|18663930185|男|
| 21| 信息学院|  Andy|18663930186|女|
| 19| 信息学院|  Justin|18663930187|男|
| 19| 信息学院|  Aaron|18663930188|男|
+---+-----+-----+-----+
only showing top 5 rows
```

图 5-60

`df.distinct()` 去重之后, 删除了重复行, 如图 5-61 所示。

```
scala> df.distinct().show(40)
+---+-----+-----+-----+
|age|institute| name|      phone|sex|
+---+-----+-----+-----+
| 20| 化工学院|  Beau|18863930185|女|
| 19| 医学院| Eugene|18663930115|女|
| 20| 信息学院| Michael|18663930185|男|
| 22| 化工学院| Chapman|18623930185|女|
| 19| 医学院| Dempsey|18665930185|女|
| 19| 医学院| Derrick|18673930185|男|
| 20| 经管学院| Geoff|18663930165|男|
| 18| 材料学院| Barret|18563930185|女|
| 19| 信息学院| Abel|18663930180|女|
| 20| 医学院| Dennis|18666930185|女|
| 19| 化工学院| Cedric|18603930185|男|
| 21| 信息学院| Andy|18663930186|女|
| 21| 材料学院| Abraham|18263930185|女|
| 19| 经管学院| Gilbert|18663930175|男|
| 20| 信息学院| Abbott|18663930189|男|
| 19| 信息学院| Justin|18663930187|男|
| 20| 经管学院| Gene|18663930155|男|
| 19| 材料学院| Addison|18463930185|男|
| 19| 化工学院| Basil|18763930185|女|
| 19| 材料学院| Adam|18363930185|男|
| 21| 医学院| Donald|18683930185|男|
| 19| 化工学院| Clement|18643930185|男|
| 19| 经管学院| Ford|18663930145|男|
| 20| 化工学院| Benedict|18963930185|男|
| 19| 信息学院| Aaron|18663930188|男|
| 20| 经管学院| Fabian|18663930135|女|
| 21| 医学院| Duke|18693930185|男|
| 20| 经管学院| Everley|18663930125|女|
| 21| 化工学院| Chad|18613930185|男|
+---+-----+-----+-----+
```

图 5-61

(2) **dropDuplicates**: 根据指定字段 (可多个字段组合) 去重

根据指定字段去重。由图 5-62 可以看到，也可同时输入多个列名（cols）进行组合去重。

```
def dropDuplicates(coll: String, cols: String*): Dataset[T]
Returns a new Dataset with duplicate rows removed, considering only the subset of
columns.
```

图 5-62

示例：

(1) 组合 age、institute 双字段进行去重，如图 5-63 所示。

```
scala> df.dropDuplicates("age","institute").show()
+-----+-----+-----+-----+-----+
|age|institute|  name|  phone|sex|
+-----+-----+-----+-----+-----+
| 21| 化工学院|  Chad|18613930185| 男|
| 18| 材料学院| Barret|18563930185| 女|
| 21| 信息学院|  Andy|18663930186| 女|
| 21| 材料学院| Abraham|18263930185| 女|
| 20| 信息学院| Michael|18663930185| 男|
| 21| 医学院| Donald|18683930185| 男|
| 20| 医学院| Dennis|18666930185| 女|
| 20| 化工学院| Beau|18863930185| 女|
| 20| 经管学院| Everley|18663930125| 女|
| 19| 化工学院| Basil|18763930185| 女|
| 22| 化工学院| Chapman|18623930185| 女|
| 19| 材料学院| Adam|18363930185| 男|
| 19| 经管学院| Ford|18663930145| 男|
| 19| 信息学院| Justin|18663930187| 男|
| 19| 医学院| Dempsey|18665930185| 女|
+-----+-----+-----+-----+-----+
```

图 5-63

(2) 使用 age 单字段去重，如图 5-64 所示。

```
scala> df.dropDuplicates("age").show()
+-----+-----+-----+-----+-----+
|age|institute|  name|  phone|sex|
+-----+-----+-----+-----+-----+
| 19| 信息学院| Justin|18663930187| 男|
| 22| 化工学院| Chapman|18623930185| 女|
| 18| 材料学院| Barret|18563930185| 女|
| 21| 信息学院|  Andy|18663930186| 女|
| 20| 信息学院| Michael|18663930185| 男|
+-----+-----+-----+-----+-----+
```

图 5-64

5.3.8 聚合操作

聚合操作是指 agg 方法，如图 5-65 所示。

```
def agg(aggExpr: (String, String), aggExprs: (String, String)*): DataFrame
  (Scala-specific) Aggregates on the entire Dataset without groups.
```

```
// ds.agg(...) is a shorthand for ds.groupBy().agg(...)
ds.agg("age" -> "max", "salary" -> "avg")
ds.groupBy().agg("age" -> "max", "salary" -> "avg")
```

Since 2.0.0

图 5-65

聚合操作调用的是 `agg` 方法，该方法输入的是对于聚合操作的表达 (`aggExpr`)，可同时对多个列进行聚合操作 (`aggExprs`)。一般与 `groupBy` 方法配合使用。

以下示例是其中最简单直观的一种用法，对 `age` 字段求平均值，对 `phone` 字段求最大值（见图 5-66）。

```
df.agg("age" -> "mean", "phone" -> "max").show()
```

```
scala> df.agg("age"->"mean","phone"->"max").show()
+-----+-----+
|          avg(age) | max(phone) |
+-----+-----+
|19.733333333333334|18963930185|
+-----+-----+
```

图 5-56

配合 `groupBy` 方法使用，如图 5-67 所示。

```
df.groupBy("institute").agg("age" -> "mean", "phone" -> "max").show()
```

```
scala> df.groupBy("institute").agg("age"->"mean","phone"->"max").show()
+-----+-----+-----+
|institute|          avg(age) | max(phone) |
+-----+-----+-----+
| 医学院 |19.833333333333332|18693930185|
|材料学院|          19.25 |18563930185|
|化工学院|          20.0 |18963930185|
|信息学院|19.714285714285715|18663930189|
|经管学院|19.666666666666668|18663930175|
+-----+-----+-----+
```

图 5-67

5.3.9 union 合并操作

`union` 方法如图 5-68 所示。

```
def union(other: Dataset[T]): Dataset[T]
  Returns a new Dataset containing union of rows in this Dataset and
  another Dataset.
```

图 5-68

`union` 方法对两个字段一致的 `DataFrame` 进行组合，返回是组合生成的新 `DataFrame`。类似于 SQL 中的 UNION 操作。

示例（见图 5-69）：

```
df.limit(5).union(df.limit(5))
```

```
scala> df.limit(5).union(df.limit(5)).show()
+-----+-----+-----+-----+-----+
|age|institute|  name|  phone|sex|
+-----+-----+-----+-----+-----+
| 20| 信息学院|Michael|18663930185|男|
| 20| 信息学院|Michael|18663930185|男|
| 21| 信息学院|  Andy|18663930186|女|
| 19| 信息学院| Justin|18663930187|男|
| 19| 信息学院| Aaron|18663930188|男|
| 20| 信息学院|Michael|18663930185|男|
| 20| 信息学院|Michael|18663930185|男|
| 21| 信息学院|  Andy|18663930186|女|
| 19| 信息学院| Justin|18663930187|男|
| 19| 信息学院| Aaron|18663930188|男|
+-----+-----+-----+-----+-----+
```

图 5-69

结果说明：

将两个相同的 df 前五条数据构成的 DataFrame 合并转化成新的 DataFrame。

5.3.10 join 操作

重点来了。在 SQL 语言中用得很多的就是 join 操作，DataFrame 中同样也提供了 join 的功能。

接下来隆重介绍 join 方法。在 DataFrame 中提供了以下五个重载的 join 方法，如图 5-70 所示。

- ①

```
def join(right: Dataset[_], usingColumn: String): DataFrame
    Inner equi-join with another DataFrame using the given column.
```
- ②

```
def join(right: Dataset[_], usingColumns: Seq[String]): DataFrame
    Inner equi-join with another DataFrame using the given columns.
```
- ③

```
def join(right: Dataset[_], usingColumns: Seq[String], joinType:
    String): DataFrame
    Equi-join with another DataFrame using the given columns.
```
- ④

```
def join(right: Dataset[_], joinExprs: Column): DataFrame
    Inner join with another DataFrame, using the given join expression.
```
- ⑤

```
def join(right: Dataset[_], joinExprs: Column, joinType: String):
    DataFrame
    Join with another DataFrame, using the given join expression.
```

图 5-70

(1) join 五大重载函数说明

观察图 5-70 中的五个 join() 函数，发现其主要区别在于输入参数的个数与类型不同。

其中，① ② ④ join() 方法皆为内连接（inner join），因为这两个 join() 方法并没有调节

join 类型的 joinType 的参数输入，因此是默认的内连接，而③ ⑤方法皆有 joinType: String 该参数，因此可从 inner、cross、outer、ull、full_outer、left、left_outer、right、right_outer、left_semi、left_anti 选择任何一种连接类型进行 join 操作。

观察① ②join() 函数，这两者主要区别在于第二个输入参数分别为 usingColumn: String、usingColumns: Seq[String]，前者是表示一个字段的 String，后者是可以表示多个字段的 Seq（序列），即当我们在两个 DataFrame 对象进行连接操作时，不仅可以基于一个字段，也可以用多个字段进行匹配连接。

观察④ ⑤join 方法，可看到出第二个输入参数不再是象征着字段的 usingColumn: String、usingColumns: Seq[String]，而是 joinExprs:Column 这种表示两个参与 join 运算的连接字段的表述（expression），如图 5-71 所示。

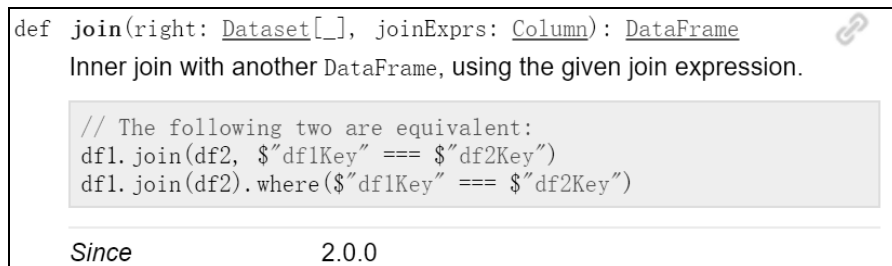


图 5-71

关于 inner、outer、full、left、left_outer、right、right_outer 等连接方式，本书不做详细解释，若对于内连接、外连接、全连接概念不清晰，请自行上网查阅“SQL 中的 join 类型”相关内容。

（2）根据特定字段进行 join 操作

下面这种 join 类似于 a join b using column1 的形式，需要两个 DataFrame 中有相同的一个列名。

示例：

```
joinDf1.join(joinDf2, "name").show(5)
```

JoinDf1 结构展示如图 5-72 所示。

```
scala> joinDf1.show(5)
+---+-----+-----+-----+---+
|age|institute|  name|    phone|sex|
+---+-----+-----+-----+---+
| 20|   信息学院|Michael|18663930185|男|
| 20|   信息学院|Michael|18663930185|男|
| 21|   信息学院|  Andy|18663930186|女|
| 19|   信息学院|Justin|18663930187|男|
| 19|   信息学院|Aaron|18663930188|男|
+---+-----+-----+-----+---+
only showing top 5 rows
```

图 5-72

JoinDf2 结构展示如图 5-73 所示。

```
scala> joinDf2.show(5)
+-----+-----+
|height|  name|weight|
+-----+-----+
|   180|Michael|   140|
|   180|  Andy|   200|
|   170| Justin|   140|
|   170| Aaron|   140|
|   170| Abbott|    90|
+-----+-----+
only showing top 5 rows
```

图 5-73

InnerJoin (单字段):

```
joinDf1.join(joinDf2, "name").show(5)
```

如图 5-74 所示。

```
scala> joinDf1.join(joinDf2, "name").show(5)
+-----+-----+-----+-----+-----+-----+
|  name|age|institute|  phone|sex|height|weight|
+-----+-----+-----+-----+-----+-----+
|Michael| 20| 信息学院|18663930185|男|  180|  140|
|Michael| 20| 信息学院|18663930185|男|  180|  140|
|  Andy| 21| 信息学院|18663930186|女|  180|  200|
| Justin| 19| 信息学院|18663930187|男|  170|  140|
| Aaron| 19| 信息学院|18663930188|男|  170|  140|
+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

图 5-74

joinDf1 和 joinDf2 根据 name 字段进行 join 操作, 结果如上, name 字段只显示一次。

(3) 根据多个字段进行 join 操作

除了上面这种 using 一个字段的情况外, 当两个 DataFrame 进行 join 操作需要两个或多个字段标识匹配一条记录时, 还可以 using 多个字段, 多个字段通过 Seq 类型的序列传入:

```
joinDf1.join(joinDf2, Seq("id", "name"))
```

(4) 指定 join 类型

两个 DataFrame 的 join 操作有 inner、outer、left_outer、right_outer、leftsemiinner、cross 等类型。在上面的根据多个字段 join 的情况下, 可以写第三个 String 类型参数, 指定 join 的类型, 如下所示:

```
joinDf1.join(joinDf2, Seq("id", "name"), "right_outer")
```

(5) 使用 Column 类型来 join

如果不采用① ② ③join() 方法通过直接传入列名或多个列名组成的序列的指定连接条件的方式, 也可以使用④ ⑤join() 方法如下直接分别指定两个 DataFrame 连接的 Column 的灵活表达形式, 如图 5-75 所示。

```
joinDf1.join(joinDf2, joinDf1("id") === joinDf2("t1_id")).show()
```



```
scala> joinDf1.join(joinDf2, joinDf1("name") === joinDf2("name")).show()
+-----+-----+-----+-----+-----+-----+-----+
|age|institute|name|phone|sex|height|name|weight|
+-----+-----+-----+-----+-----+-----+-----+
|20|信息学院|Michael|18663930185|男|180|Michael|140|
|20|信息学院|Michael|18663930185|男|180|Michael|140|
|21|信息学院|Andy|18663930186|女|180|Andy|200|
|19|信息学院|Justin|18663930187|男|170|Justin|140|
|19|信息学院|Aaron|18663930188|男|170|Aaron|140|
|20|信息学院|Abbott|18663930189|男|170|Abbott|90|
|19|信息学院|Abel|18663930180|女|170|Abel|90|
|21|材料学院|Abraham|18263930185|女|170|Abraham|91|
|19|材料学院|Adam|18363930185|男|170|Adam|120|
|19|材料学院|Addison|18463930185|男|170|Addison|110|
|18|材料学院|Barret|18563930185|女|170|Barret|80|
|19|化工学院|Basil|18763930185|女|170|Basil|160|
|20|化工学院|Beau|18863930185|女|170|Beau|170|
|20|化工学院|Benedict|18963930185|男|170|Benedict|120|
|19|化工学院|Cedric|18603930185|男|170|Cedric|130|
+-----+-----+-----+-----+-----+-----+-----+
```

图 5-75

(6) 在指定 join 字段同时指定 join 类型 (见图 5-76)

```
joinDf1.join(joinDf2, joinDf1("id") === joinDf2("t1_id"), "cross").show()
```

```
root@jihan-virtual-machine: /opt/spark-2.2.0-bin-hadoop2.7/bin
scala> joinDf1.join(joinDf2, joinDf1("name") === joinDf2("name"), "cross").show()
+-----+-----+-----+-----+-----+-----+-----+
|age|institute|name|phone|sex|height|name|weight|
+-----+-----+-----+-----+-----+-----+-----+
|20|信息学院|Michael|18663930185|男|180|Michael|140|
|20|信息学院|Michael|18663930185|男|180|Michael|140|
|21|信息学院|Andy|18663930186|女|180|Andy|200|
|19|信息学院|Justin|18663930187|男|170|Justin|140|
|19|信息学院|Aaron|18663930188|男|170|Aaron|140|
|20|信息学院|Abbott|18663930189|男|170|Abbott|90|
|19|信息学院|Abel|18663930180|女|170|Abel|90|
|21|材料学院|Abraham|18263930185|女|170|Abraham|91|
|19|材料学院|Adam|18363930185|男|170|Adam|120|
|19|材料学院|Addison|18463930185|男|170|Addison|110|
|18|材料学院|Barret|18563930185|女|170|Barret|80|
|19|化工学院|Basil|18763930185|女|170|Basil|160|
|20|化工学院|Beau|18863930185|女|170|Beau|170|
|20|化工学院|Benedict|18963930185|男|170|Benedict|120|
|19|化工学院|Cedric|18603930185|男|170|Cedric|130|
+-----+-----+-----+-----+-----+-----+-----+
```

图 5-76

5.3.11 获取指定字段统计信息

stat 方法可以用于计算指定字段或指定字段之间的统计信息, 比如方差、协方差、某字段出现频繁的元素集合等。这个方法返回一个 DataFrameStatFunctions 类型对象, 如图 5-77 所示。

```
def stat: DataFrameStatFunctions
  Returns a DataFrameStatFunctions for working statistic functions support.
```

图 5-77

DataFrame 调用 Stat 方法后获得 DataFrameStatFunctions 类型对象, 该对象下有子调用接

口 `freqItems`、`corr`、`cov` 等，`freqItems` 可用于分别计算某一列或几列中出现频繁的值集合，`corr` 可用于计算列与列之间的相关性，而 `cov` 可用于计算两列的协方差。`freqItems`、`corr`、`cov` 子接口如图 5-78 所示。

<pre>def freqItems(cols: Array[String]): DataFrame Finding frequent items for columns, possibly with false positives.</pre>
<pre>def freqItems(cols: Array[String], support: Double): DataFrame Finding frequent items for columns, possibly with false positives.</pre>
<pre>def corr(col1: String, col2: String): Double Calculates the Pearson Correlation Coefficient of two columns of a DataFrame.</pre>
<pre>def corr(col1: String, col2: String, method: String): Double Calculates the correlation of two columns of a DataFrame.</pre>
<pre>def cov(col1: String, col2: String): Double Calculate the sample covariance of two numerical columns of a DataFrame.</pre>

图 5-78

其中，两个 `freqItem()` 方法内的需输入表示列的 `Array` 可以换成对应的包含列的 `Seq`，即 `cols: Array[String]`、`cols: Seq[String]` 两种表现形式皆可。

`freqItems` 演示：

下面代码演示根据 `age`、`height` 字段，统计该字段值出现频率在 30% 以上的内容，如图 5-79 所示。

```
df.stat.freqItems(Array("age", "height"), 0.3).show()
df.stat.freqItems(Seq("age", "height"), 0.3).show()
```

```
scala> df.stat.freqItems(Array("age", "height"), 0.3).show()
+-----+-----+
|age_freqItems|height_freqItems|
+-----+-----+
| [20, 19, 21]| [180, 170]|
+-----+-----+

scala> df.stat.freqItems(Seq("age", "height"), 0.3).show()
+-----+-----+
|age_freqItems|height_freqItems|
+-----+-----+
| [20, 19, 21]| [180, 170]|
+-----+-----+
```

图 5-79

```
freqItems(cols: Seq[String]/Array[String], support: Double)
```

如上所述，可见 `cols` 即可用 `Seq[String]`/`Array[String]` 两种方式其中任何一种表示，而通过第二个参数 `support: Double` 可以控制在某一列中筛选出在某频率之上的值的集合。

`corr` 求两列相关性演示（见图 5-80）：

下面代码演示求取 height、weight 相关性。

```
df.stat.corr("height", "weight")
```

```
scala> df.stat.corr("height", "weight")
res20: Double = 0.49566271999285366
```

图 5-80

可见，返回值为 Double 类型的值，可根据该值查询相关系数显著性检验表，确认两列之间的相关程度

cov 求两列协方差演示如下，这段代码演示求取 height、weight 协方差（见图 5-81）。

```
df.stat.cov("height", "weight")
```

```
scala> df.stat.cov("height", "weight")
res21: Double = 68.42857142857143
```

图 5-81

可见，返回值也为 Double 类型的值，可参考确认两列之间的关系。

5.3.12 获取两个 DataFrame 中共有的记录

获取两个 DataFrame 中共有的记录方法如图 5-82 所示。

```
def intersect(other: Dataset[T]): Dataset[T]
Returns a new Dataset containing rows only in both this Dataset and another Dataset.
```

图 5-82

intersect 方法可以计算出两个 DataFrame 中相同的记录,返回值也为 DataFrame，如图 5-83 所示。

```
Df.intersect(df.limit(1)).show(false)
```

```
scala> df.intersect(df.limit(1)).show(false)
+---+-----+-----+-----+-----+-----+-----+
|age|institute|name  |phone      |sex|height|name  |weight|
+---+-----+-----+-----+-----+-----+-----+
|20 |信息学院  |Michael|18663930185|男 |180   |Michael|140   |
+---+-----+-----+-----+-----+-----+-----+
```

图 5-83

5.3.13 获取一个 DataFrame 中有另一个 DataFrame 中没有的记录

获取一个 DataFrame 中有，另一个 DataFrame 中没有的记录，方法如图 5-84 所示。

```
def except(other: Dataset[T]): Dataset[T]
Returns a new Dataset containing rows in this Dataset but not in another Dataset.
```

图 5-84

演示实例（见图 5-85）：

```
df.except(df.limit(2)).show(false)
```

```
scala> df.show(5)
+-----+-----+-----+-----+-----+-----+-----+
|age|institute|name|phone|sex|height|name|weight|
+-----+-----+-----+-----+-----+-----+-----+
|20|信息学院|Michael|18663930185|男|180|Michael|140|
|20|信息学院|Michael|18663930185|男|180|Michael|140|
|21|信息学院|Andy|18663930186|女|180|Andy|200|
|19|信息学院|Justin|18663930187|男|170|Justin|140|
|19|信息学院|Aaron|18663930188|男|170|Aaron|140|
+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

scala> df.except(df.limit(2)).show(5)
+-----+-----+-----+-----+-----+-----+-----+
|age|institute|name|phone|sex|height|name|weight|
+-----+-----+-----+-----+-----+-----+-----+
|19|化工学院|Basil|18763930185|女|170|Basil|160|
|19|材料学院|Addison|18463930185|男|170|Addison|110|
|20|信息学院|Abbott|18663930189|男|170|Abbott|90|
|19|信息学院|Justin|18663930187|男|170|Justin|140|
|19|信息学院|Abel|18663930180|女|170|Abel|90|
+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

图 5-85

如图 5-85 所示，经过 except 操作后，返回的 DataFrame 中已删除了之前的前两条记录。

5.3.14 操作字段名

(1) withColumnRenamed: 重命名 DataFrame 中的指定字段名（见图 5-86）

```
def withColumnRenamed(existingName: String, newName: String): DataFrame
Returns a new Dataset with a column renamed.
```

图 5-86

如果指定的字段名不存在，不进行任何操作。下面示例中将 df 中的 name 字段重命名为 englishName，如图 5-87 所示。

```
df.withColumnRenamed("name", "englishName")
```

```
scala> df.withColumnRenamed("name", "englishName").show()
+-----+-----+-----+-----+-----+-----+-----+
|age|institute|englishName|phone|sex|height|englishName|weight|
+-----+-----+-----+-----+-----+-----+-----+
|20|信息学院|Michael|18663930185|男|180|Michael|140|
|20|信息学院|Michael|18663930185|男|180|Michael|140|
|21|信息学院|Andy|18663930186|女|180|Andy|200|
|19|信息学院|Justin|18663930187|男|170|Justin|140|
|19|信息学院|Aaron|18663930188|男|170|Aaron|140|
|20|信息学院|Abbott|18663930189|男|170|Abbott|90|
|19|信息学院|Abel|18663930180|女|170|Abel|90|
|21|材料学院|Abraham|18263930185|女|170|Abraham|91|
|19|材料学院|Adam|18363930185|男|170|Adam|120|
|19|材料学院|Addison|18463930185|男|170|Addison|110|
|18|材料学院|Barret|18563930185|女|170|Barret|80|
|19|化工学院|Basil|18763930185|女|170|Basil|160|
|20|化工学院|Beau|18863930185|女|170|Beau|170|
|20|化工学院|Benedict|18963930185|男|170|Benedict|120|
|19|化工学院|Cedric|18603930185|男|170|Cedric|130|
+-----+-----+-----+-----+-----+-----+-----+
```

图 5-87

(2) **withColumn**: 往当前 **DataFrame** 中新增一列,该列可来源于本身 **DataFrame** 对象,不可来自其他非已 **DataFrame** 对象 (见图 5-88)

```
def withColumn(colName: String, col: Column): DataFrame
    Returns a new Dataset by adding a column or replacing the existing column that has the
    same name.
```

图 5-88

withColumn(colName: String, col: Column)方法根据指定 **colName** 往 **DataFrame** 中新增一列, 如果 **colName** 已存在, 则会覆盖当前列。

以下代码往 **df** 中新增一个名为 “age*2” 的列, 如图 5-89 所示。

```
df.withColumn("age*2",df("age")*2).show(5)
```

```
scala> df.withColumn("age*2",df("age")*2).show(5)
+-----+-----+-----+-----+-----+
|age|institute|name|phone|sex|age*2|
+-----+-----+-----+-----+-----+
|20|信息学院|Michael|18663930185|男|40|
|21|信息学院|Andy|18663930186|女|42|
|19|信息学院|Justin|18663930187|男|38|
|19|信息学院|Aaron|18663930188|男|38|
|20|信息学院|Abbott|18663930189|男|40|
+-----+-----+-----+-----+-----+
```

图 5-89

5.3.15 处理空值列

使用 **na** 方法对具有空值列的行数据进行处理,例如删除缺失某一列值的行或用指定值(缺失值)替换空值列的值, 如图 5-90 所示。

```
def na: DataFrameNaFunctions
    Returns a DataFrameNaFunctions for working with missing data.
```

图 5-90

需要注意的是, 在 **DataFrame** 对象上使用 **na** 方法后返回的是对应的 **DataFrameNaFunction** 对象, 进而调用对应的 **drop**、**fill** 方法来处理指定列为空值的行。

(1) **drop**: 删除指定列为空值的行 (见图 5-91)

```
def drop(): DataFrame
    Returns a new DataFrame that drops rows containing any null or NaN values.
```

```
def drop(cols: Array[String]): DataFrame
    Returns a new DataFrame that drops rows containing any null or NaN values in the specified columns.
```

图 5-91

第一个 **drop()** 无参方法, 只要行数据中有空值列 (一个或多个空值列) 便进行删除, 而第二个 **drop(cols: Array[String])** 重载方法, 可通过将指定列的列名组成的数组传入 **drop** 方法, 用于当特定的一个或几个列同时为空值时才删除的情况。

drop 方法前的 DataFrame 对象 (df) 内容展示 (见图 5-92):

```
df.show(5)
```

```
scala> df.show(5)
+---+-----+-----+-----+
| age|institute| name|      phone| sex|
+---+-----+-----+-----+
| null|      null| null|      null| null|
| null| 信息学院| Andy|18663930186| 女|
| 19| 信息学院| Justin|18663930187| 男|
| 19| 信息学院| Aaron|18663930188| 男|
| 20| 信息学院| Abbott|18663930189| 男|
+---+-----+-----+-----+
only showing top 5 rows
```

图 5-92

```
df.na.drop().show(5)
```

可以看到只要具有空值列的行数据就会被删除, 如图 5-93 所示。

```
scala> df.na.drop().show(5)
+---+-----+-----+-----+
| age|institute| name|      phone| sex|
+---+-----+-----+-----+
| 19| 信息学院| Justin|18663930187| 男|
| 19| 信息学院| Aaron|18663930188| 男|
| 20| 信息学院| Abbott|18663930189| 男|
| 19| 信息学院| Abel|18663930180| 女|
| 20| 材料学院| Abner|18163930185| 男|
+---+-----+-----+-----+
only showing top 5 rows
```

图 5-93

```
df.na.drop(Array("sex")).show(5)
```

指定 sex 列为空值的行数据被删除, 如图 5-94 所示。

```
scala> df.na.drop(Array("sex")).show(5)
+---+-----+-----+-----+
| age|institute| name|      phone| sex|
+---+-----+-----+-----+
| null| 信息学院| Andy|18663930186| 女|
| 19| 信息学院| Justin|18663930187| 男|
| 19| 信息学院| Aaron|18663930188| 男|
| 20| 信息学院| Abbott|18663930189| 男|
| 19| 信息学院| Abel|18663930180| 女|
+---+-----+-----+-----+
only showing top 5 rows
```

图 5-94

(2) fill: 使用指定的值替换指定空值列的值 (见图 5-95)

```
def fill(valueMap: Map[String, Any]): DataFrame
(Scala-specific) Returns a new DataFrame that replaces null values.
```

图 5-95

通过传入指定空值列列名以及该空值列替换值组成的 Map 对象传入 fill 方法来替换指定空

值列的值。

fill 方法前 DataFrame 对象 (df) 内容展示 (见图 5-96):

```
df.show(5)
```

```
scala> df.show(5)
+-----+-----+-----+-----+-----+
| age|institute|  name|    phone| sex|
+-----+-----+-----+-----+
| null|    null|  null|    null| null|
| null|    null| Andy|18663930186| 女|
| 19| 信息学院|Justin|18663930187| 男|
| 19| 信息学院| Aaron|18663930188| 男|
| 20| 信息学院|Abbott|18663930189| 男|
+-----+-----+-----+-----+
only showing top 5 rows
```

图 5-96

```
df.na.fill(Map(("age",12),("institute","cailiao"))).show(5)
```

将指定值替换指定空值列的空值, 如图 5-97 所示。

```
scala> df.na.fill(Map(("age",12),("institute","cailiao"))).show(5)
+-----+-----+-----+-----+-----+
|age|institute|  name|    phone| sex|
+-----+-----+-----+-----+
| 12|  cailiao|  null|    null| null|
| 12|  cailiao| Andy|18663930186| 女|
| 19| 信息学院|Justin|18663930187| 男|
| 19| 信息学院| Aaron|18663930188| 男|
| 20| 信息学院|Abbott|18663930189| 男|
+-----+-----+-----+-----+
only showing top 5 rows
```

图 5-97