第 3 章

Spark SQL结构化数据处理模块

学习目标：

- 了解 Spark SQL 的简介，能够说出 Spark SQL 的特点。
- 熟悉 Spark SQL 架构，能够说明 Catalyst 内部组件的运行流程。
- 熟悉 DataFrame 的基本概念，能够说明 DataFrame 与 RDD 在结构上的区别。
- 掌握 DataFrame 的创建，能够通过读取文件创建 DataFrame。
- 掌握 DataFrame 的常用操作，能够使用 DSL 风格和 SQL 风格操作 DataFrame。
- 掌握 DataFrame 的函数操作，能够使用标量函数和聚合函数操作 DataFrame。
- 掌握 RDD 与 DataFrame 的转换，能够通过反射机制和编程方式将 RDD 转换成 DataFrame。
- 掌握 Spark SQL 操作数据源，能够使用 Spark SQL 操作 MySQL 和 Hive。

针对那些不熟悉 Spark 常用 API，但希望利用 Spark 强大数据分析能力的用户，Spark 提供了一种结构化数据处理模块 Spark SQL，Spark SQL 模块使用户可以利用 SQL 语句处理结构化数据。本章针对 Spark SQL 的基本原理和使用方式进行详细讲解。

3.1 Spark SQL 基础知识

Spark SQL 是 Spark 用来处理结构化数据的一个模块，它提供了一个名为 DataFrame 的数据模型，即带有元数据信息的 RDD。基于 Python 语言使用 Spark SQL 时，用户可以通过 SQL 和 DataFrame API 两种方式实现对结构化数据的处理。无论用户选择哪种方式，它们都是基于同样的执行引擎，可以方便地在不同的方式之间进行切换。接下来，本节对 Spark SQL 的基础知识进行介绍。

3.1.1 Spark SQL 简介

Spark SQL 的前身是 Shark，Shark 最初是由加州大学伯克利分校的实验室开发的 Spark 生态系统的组件之一，它运行在 Spark 系统上，并且重新利用了 Hive 的工作机制，并继承了 Hive 的各个组件。Shark 主要的改变是将 SQL 语句的转换方式从 MapReduce 作业替换成了 Spark 作业，从而提高了计算效率。

然而，由于 Shark 过于依赖 Hive，所以在版本迭代时很难添加新的优化策略，从而限制了 Spark 的发展，因此，在后续的迭代更新中，Shark 的维护停止了，转向 Spark SQL 的

开发。

Spark SQL 具有 3 个特点,具体内容如下。

(1) 支持多种数据源。Spark SQL 可以从各种数据源中读取数据,包括 JSON、Hive、MySQL 等,使用户可以轻松处理不同数据源的数据。

(2) 支持标准连接。Spark SQL 提供了行业标准的 JDBC 和 ODBC 连接方式,使用户可以通过外部连接方式执行 SQL 查询,不再局限于在 Spark 程序内使用 SQL 语句进行查询。

(3) 支持无缝集成。Spark SQL 提供了 Python、Scala 和 Java 等编程语言的 API,使 Spark SQL 能够与 Spark 程序无缝集成,可以将结构化数据作为 Spark 中的 RDD 进行查询。这种紧密的集成方式使用户可以方便地在 Spark 框架中进行结构化数据的查询与分析。

总体来说,Spark SQL 支持多种数据源的查询和加载,并且兼容 Hive,可以使用 JDBC 和 ODBC 的连接方式来执行 SQL 语句,它为 Spark 框架在结构化数据分析方面提供重要的技术支持。

3.1.2 Spark SQL 架构

Spark SQL 的一个重要特点就是能够统一处理数据表和 RDD,使用户可以轻松地使用 SQL 语句进行外部查询,同时进行更加复杂的数据分析。接下来,通过图 3-1 来了解 Spark SQL 底层架构。

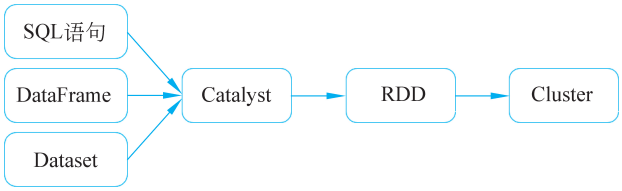

图 3-1 Spark SQL 底层架构

从图 3-1 可以看出,用户提交 SQL 语句、DataFrame 和 Dataset 后,会经过一个优化器(Catalyst),将 SQL 语句、DataFrame 和 Dataset 的执行逻辑转换为 RDD,然后提交给 Spark 集群(Cluster)处理。Spark SQL 的计算效率主要由 Catalyst 决定,也就是说 Spark SQL 执行逻辑的生成和优化工作全部交给 Spark SQL 的 Catalyst 管理。

Catalyst 是一个可扩展的查询优化框架,它基于 Scala 函数式编程,使用 Spark SQL 时,Catalyst 能够为后续的版本迭代更新轻松地添加新的优化技术和功能,特别是针对大数据生产环境中遇到的问题,如针对半结构化数据和高级数据分析。另外,Spark 作为开源项目,用户可以针对项目需求自行扩展 Catalyst 的功能。

Catalyst 内部主要包括 Parser(分析)组件、Analyzer(解析)组件、Optimizer(优化)组件、Planner(计划)组件和 Query Execution(执行)组件。接下来,通过图 3-2 来介绍 Catalyst 内部各组件的关系。

图 3-2 展示的是 Catalyst 的内部组件的关系,这些组件的运行流程如下。

(1) 当用户提交 SQL 语句、DataFrame 或 Dataset 时,它们会经过 Parser 组件进行分析。Parser 组件分析相关的执行语句,判断其是否符合规范,一旦分析完成,会创建

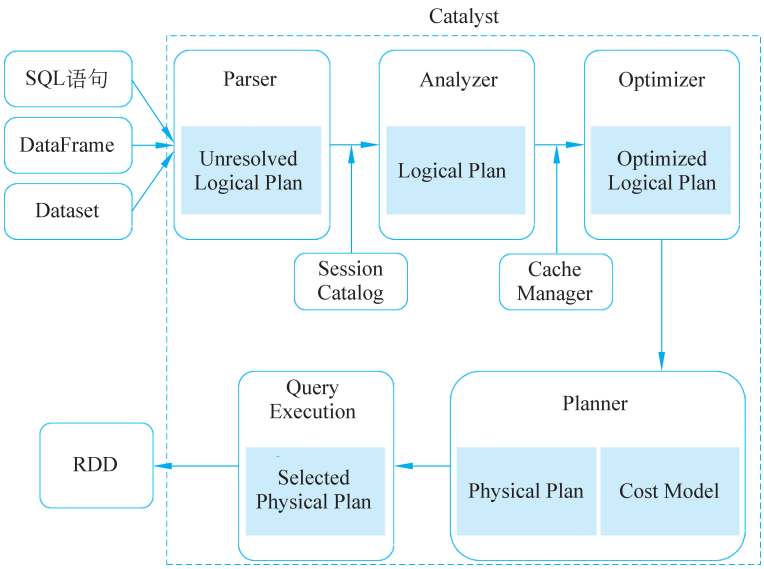

图 3-2　Catalyst 内部各组件的关系

SparkSession，并将包括表名、列名和数据类型等元数据保存在会话目录（Session Catalog）中发送给 Analyzer 组件，此时的执行语句为未解析的逻辑计划（Unresolved Logical Plan）。其中会话目录用于管理与元数据相关的信息。

（2）Analyzer 组件根据会话目录中的信息，将未解析的逻辑计划解析为逻辑计划（Logical Plan）。同时，Analyzer 组件还会验证执行语句中的表名、列名和数据类型是否存在于元数据中。如果所有的验证都通过，那么逻辑计划将被保存在缓存管理器（Cache Manager）中，并发送给 Optimizer 组件。

（3）Optimizer 组件接收到逻辑计划后进行优化处理，得到优化后的逻辑计划（Optimized Logical Plan）。例如，在计算表达式 x+(1+2)时，Optimizer 组件会将其优化为 x+3，如果没有经过优化，每个结果都需要执行一次 1+2 的操作，然后再与 x 相加，通过优化，就无须重复执行 1+2 的操作。优化后的逻辑计划会发送给 Planner 组件。

（4）Planner 组件将优化后的逻辑计划转换为多个物理计划（Physical Plan），通过成本模型（Cost Model）进行资源消耗估算，在多个物理计划中得到选择后的物理计划（Selected Physical Plan）并将其发送给 Query Execution 组件。

（5）Query Execution 组件根据选择后的物理计划生成具体的执行逻辑，并将其转化为 RDD。

3.2　DataFrame 基础知识

3.2.1　DataFrame 简介

在 Spark 中，DataFrame 是一种以 RDD 为基础的分布式数据集，因此 DataFrame 可以执行绝大多数 RDD 的功能。在实际开发中，可以方便地进行 RDD 和 DataFrame 之间的转换。

DataFrame 的结构类似于传统数据库的二维表格,并且可以由多种数据源创建,如结构化文件、外部数据库、Hive 表等。下面,通过图 3-3 来了解 DataFrame 与 RDD 在结构上的区别。

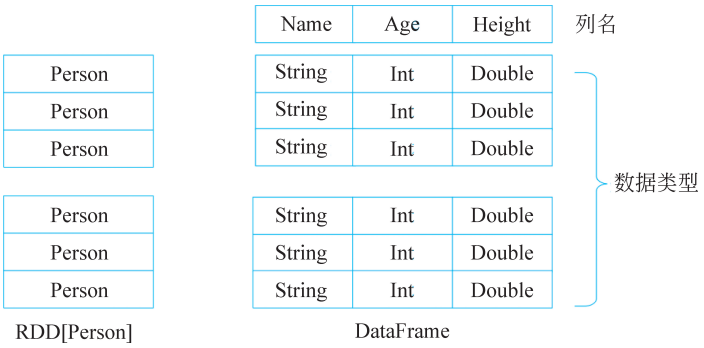

图 3-3 DataFrame 与 RDD 的区别

在图 3-3 中,左侧为 RDD[Person]数据集,右侧为 DataFrame 数据集。DataFrame 可以看作分布式 Row 对象的集合,每个 Row 表示一行数据。与 RDD 不同的是,DataFrame 还包含了元数据信息,即每列的名称和数据类型,如 Name、Age 和 Height 为列名,String、Int 和 Double 为数据类型。这使得 Spark SQL 可以获取更多的数据结构信息,并对数据源和 DataFrame 上的操作进行精细化的优化,最终提高计算效率,同时,DataFrame 与 Hive 类似,DataFrame 也支持嵌套数据类型,如 Struct、Array 和 Map。

RDD[Person]虽然以 Person 为类型参数,但是 Spark SQL 无法获取 RDD[Person]内部的结构,导致在转换数据形式时效率相对较低。

总的来说,DataFrame 提高了 Spark SQL 的执行效率、减少数据读取时间以及优化执行计划。引入 DataFrame 后,处理数据就更加简单了,可以直接用 SQL 或 DataFrame API 处理数据,极大提升了用户的易用性。通过 DataFrame API 或 SQL 处理数据时,Catalyst 会自动优化查询计划,即使用户编写的程序或 SQL 语句在逻辑上不是最优的,Spark 仍能够高效地执行这些查询。

3.2.2 DataFrame 的创建

在 Spark 2.0 之后,Spark 引入了 SparkSession 接口更方便地创建 DataFrame。创建 DataFrame 时需要创建 SparkSession 对象,该对象的创建分为两种方式,一种是通过代码 SparkSession.builder.master().appName().getOrCreate()来创建。另一种是 PySpark 中会默认创建一个名为 spark 的 SparkSession 对象。一旦 SparkSession 对象创建完成后,就可以通过其提供的 read 属性获取一个 DataFrameReader 对象,并利用该对象调用一系列方法从各种类型的文件中读取数据创建 DataFrame。

接下来,基于 YARN 集群的运行模式启动 PySpark。在虚拟机 Hadoop1 的目录/export/servers/sparkOnYarn/spark-3.3.0-bin-hadoop3 中执行如下命令。

```
$ bin/pyspark --master yarn
```

上述命令执行完成后的效果如图 3-4 所示。

从图 3-4 可以看出,PySpark 默认创建了一个名为 spark 的 SparkSession 对象。

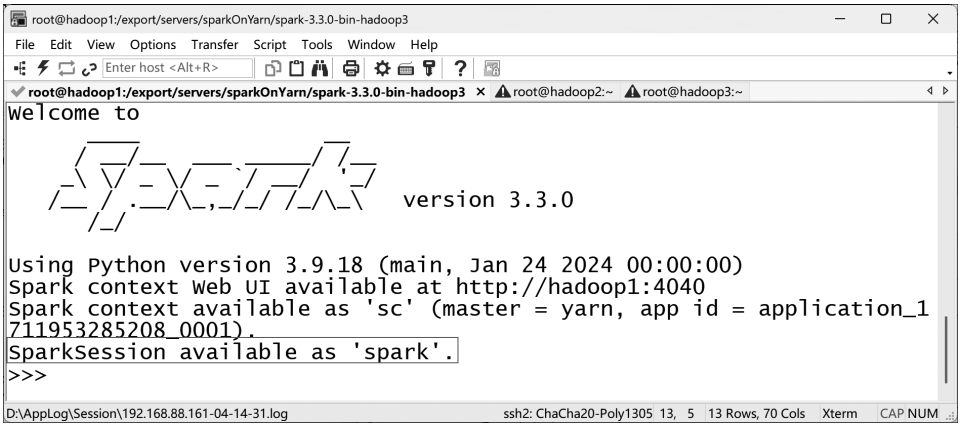

图 3-4　启动 PySpark

常见的读取数据创建 DataFrame 的方法如表 3-1 所示。

表 3-1　常见的读取数据创建 DataFrame 的方法

方　法	语 法 格 式	说　明
text()	SparkSession.read.text(path)	从指定目录 path 读取文本文件，创建 DataFrame
csv()	SparkSession.read.csv(path)	从指定目录 path 读取 CSV 文件，创建 DataFrame
json()	SparkSession.read.json(path)	从指定目录 path 读取 JSON 文件，创建 DataFrame
parquet()	SparkSession.read.parquet(path)	从指定目录 path 读取 parquet 文件，创建 DataFrame
toDF()	RDD.toDF([col,col,…])	用于将一个 RDD 转换为 DataFrame，并且可以指定列名 col。默认情况下，列名的格式为_1、_2 等
createDataFrame()	SparkSession.createDataFrame(data, schema)	通过读取自定义数据 data 创建 DataFrame

在表 3-1 中，参数 data 用于指定 DataFrame 的数据，其值的类型可以是数组、List 集合或者 RDD。参数 schema 为可选用于指定 DataFrame 的元信息，包括列名、数据类型等。如果没有指定参数 schema，那么使用默认的列名，其格式为_1、_2 等。而数据类型则通过数据自行推断。

接下来，通过读取 JSON 文件演示如何创建 DataFrame，具体步骤如下。

1. 数据准备

克隆一个虚拟机 Hadoop1 的会话，在虚拟机 Hadoop1 的/export/data 目录下执行 vi person.json 命令创建 JSON 文件 person.json，具体内容如文件 3-1 所示。

文件 3-1　person.json

```
{"age":20, "id":1, "name":"zhangsan"}
{"age":18, "id":2, "name":"lisi"}
{"age":21, "id":3, "name":"wangwu"}
{"age":23, "id":4, "name":"zhaoliu"}
{"age":25, "id":5, "name":"tianqi"}
{"age":19, "id":6, "name":"xiaoba"}
```

数据文件创建完成后,在/export/data 目录执行 hdfs dfs -put person.json /命令将 JSON 文件 person.json 上传到 HDFS 的根目录。

2. 读取文件创建 DataFrame

通过读取 JSON 文件 person.json 创建 DataFrame,具体代码如下。

```
>>> personDF = spark.read.json("/person.json")
```

上述代码中,使用 json()方法读取 HDFS 根目录的 JSON 文件 person.json 创建名为 personDF 的 DataFrame。

DataFrame 创建完成后,可以通过 printSchema()方法输出 personDF 的元数据信息,具体代码如下。

```
>>> personDF.printSchema()
```

上述代码运行完成后的效果如图 3-5 所示。

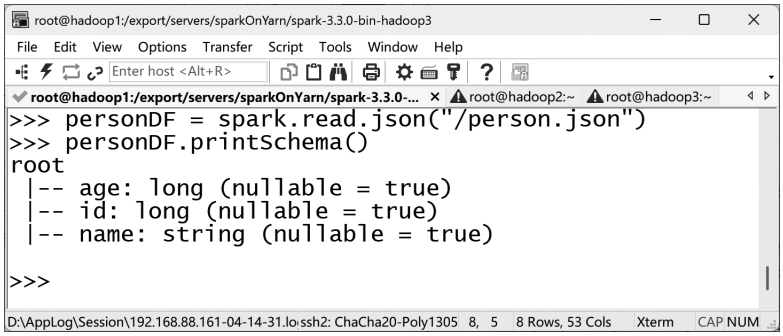

图 3-5 输出 personDF 的元数据信息

从图 3-5 可以看出,JSON 文件 person.json 中的键会作为 DataFrame 的列名,而列的数据类型会根据键对应的值自行推断。此外,默认情况下,DataFrame 中每个列的值可以为空,即 nullable = true。例如,根据 JSON 文件 person.json 中键 name 的值推断出列 name 的数据类型为 string。

使用 show()方法查看当前 DataFrame 的内容,具体代码如下。

```
>>> personDF.show()
```

上述代码运行完成后的效果如图 3-6 所示。

从图 3-6 可以看出,DataFrame 的内容为二维表格的形式,其中列与 JSON 文件 person.json 中的键有关,而列的数据为 JSON 文件 person.json 中键对应的值。

3.2.3 DataFrame 的常用操作

多样性是人类社会发展的基石,是文明进步的源泉。在一个多样化的社会中,每个人都能找到属于自己的位置,贡献独特的智慧和力量。尊重和包容多样性,不仅是对每个人基本权利的尊重,也是实现社会公平正义的必要条件。DataFrame 提供了两种语法风格,分别是

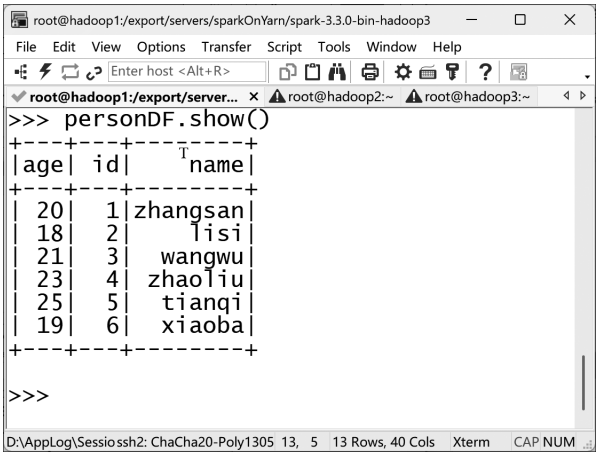

图 3-6　查看 DataFrame 的内容

DSL(Dynamic Script Language,领域特定语言)风格和 SQL 风格,前者通过 DataFrame API 的方式操作 DataFrame,后者通过 SQL 语句的方式操作 DataFrame。接下来,针对 DSL 风格和 SQL 风格分别讲解 DataFrame 的具体操作方式。

1. DSL 风格

DataFrame 提供了一种 DSL 风格去管理结构化数据的方式,可以在 Scala、Java、Python 和 R 语言中使用 DSL。下面,以 Python 语言使用 DSL 为例,讲解 DataFrame 的常用操作。

(1) printSchema()方法:查看 DataFrame 的元数据信息。

(2) show()方法:查看 DataFrame 中的内容。

(3) select()方法:选择 DataFrame 中指定列。

下面基于 3.2.2 节创建的 DataFrame 选择并查看其 name 列的数据,具体代码如下。

```
>>> personDF.select(personDF["name"]).show()
```

上述代码运行完成后的效果如图 3-7 所示。

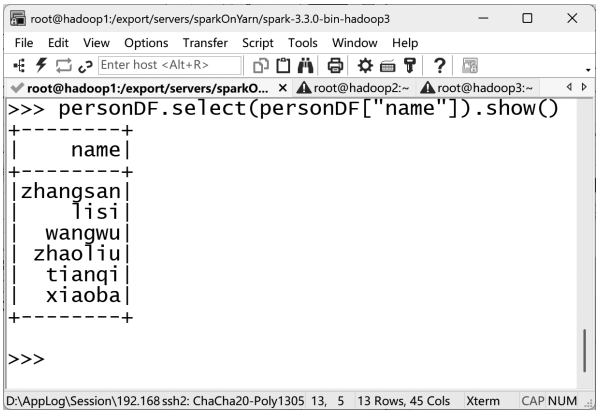

图 3-7　select()方法的使用

从图 3-7 可以看出,DataFrame 中 name 列的数据包括 zhangsan、lisi、wangwu、zhaoliu、tianqi、xiaoba。

(4) filter()方法:实现条件查询,筛选出想要的结果。

下面演示如何筛选 DataFrame 中 age 列大于或等于 20 的数据,具体代码如下。

```
>>> personDF.filter(personDF["age"] >= 20).show()
```

上述代码运行完成后的效果如图 3-8 所示。

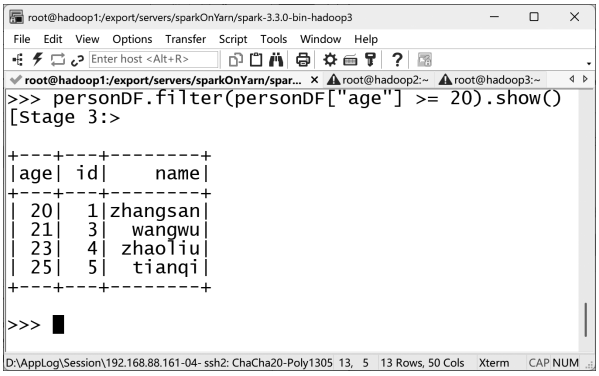

图 3-8 filter()方法的使用

从图 3-8 可以看出,DataFrame 中包含 4 条 age 列大于或等于 20 的数据。

(5) groupBy()方法:根据 DataFrame 的指定列进行分组,分组完成后可通过 count()方法对每个组内的元素进行计数操作。

下面演示如何将 DataFrame 中 age 列进行分组并统计每个组内元素的个数,具体代码如下。

```
>>> personDF.groupBy("age").count().show()
```

上述代码运行完成后的效果如图 3-9 所示。

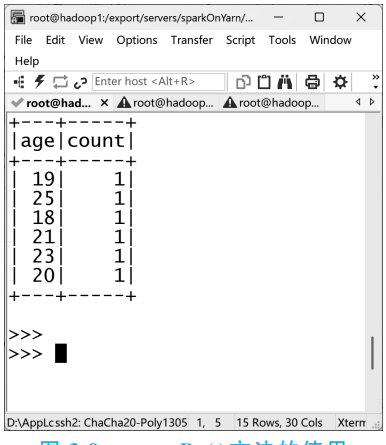

图 3-9 groupBy()方法的使用

从图 3-9 可以看出，根据 DataFrame 中 age 列分为 6 组，每组内元素的个数都为 1。

(6) sort()方法：根据指定列进行排序操作，默认是升序排序，若指定为降序排序，需要使用 desc()方法指定排序规则为降序排序。

下面演示如何将 DataFrame 中的 age 列进行降序排序，具体代码如下。

```
>>> personDF.sort(personDF["age"].desc()).show()
```

上述代码运行完成后的效果如图 3-10 所示。

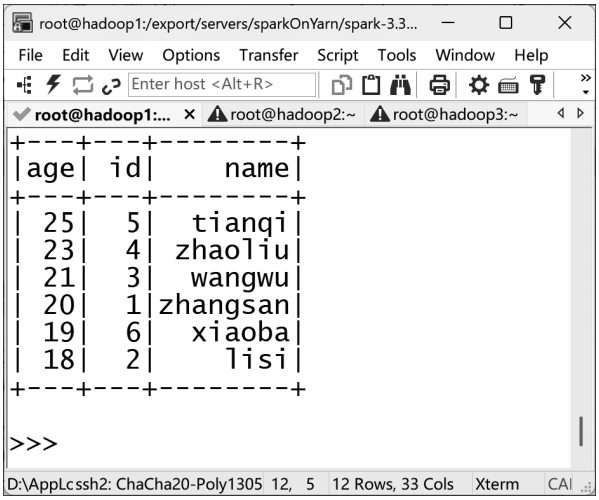

图 3-10　sort()方法的使用

从图 3-10 可以看出，DataFrame 中的数据根据 age 列进行降序排序。

2. SQL 风格

DataFrame 的强大之处就是可以将它看作一个关系型数据表，然后可以在 Spark 中直接使用 spark.sql()的方式执行 SQL 查询，结果将作为一个 DataFrame 返回。使用 SQL 风格操作的前提是需要使用 createOrReplaceTempView()方法将 DataFrame 创建成一个临时视图。接下来，创建一个 DataFrame 的临时视图 t_person，具体代码如下。

```
>>> personDF.createOrReplaceTempView("t_person")
```

上述代码中，通过 createOrReplaceTempView()方法创建 personDF 的临时视图 t_person。使用 createOrReplaceTempView()方法创建的临时视图的生命周期依赖于 SparkSession，即 SparkSession 存在则临时视图存在。当用户想要手动删除临时视图时，可以通过执行 spark.catalog.dropTempView("t_person")代码实现，其中 t_person 用于指定临时视图的名称。

下面，演示使用 SQL 风格方式操作 DataFrame。

(1) 查询临时视图 t_person 中 age 列的值最大的两行数据，具体代码如下。

```
>>> spark.sql("select * from t_person order by age desc limit 2").show()
```

上述代码中，通过 SQL 语句对临时视图 t_person 中 age 列进行降序排序，并获取排序

结果的前两条数据。

上述代码运行完成后的效果如图 3-11 所示。

从图 3-11 可以看出,成功筛选出临时视图 t_person 中 age 列的值最大的两行数据。

(2) 查询临时视图 t_person 中 age 列的值大于 20 的数据,具体代码如下。

```
>>> spark.sql("select * from t_person where age>20").show()
```

上述代码运行完成后的效果如图 3-12 所示。

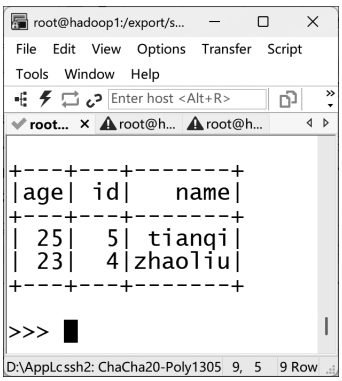

图 3-11 临时视图 t_person 中 age 列的值最大的两行数据

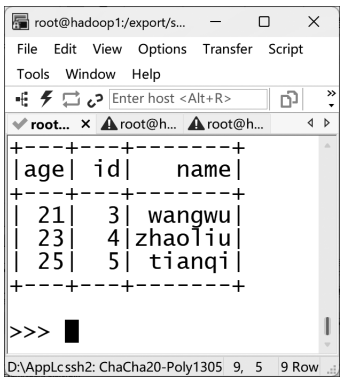

图 3-12 临时视图 t_person 中 age 列的值大于 20 的数据

从图 3-12 可以看出,成功筛选出 age 列的值大于 20 的数据。

DataFrame 操作方式简单,并且功能强大,熟悉 SQL 语法的用户都能够快速地掌握 DataFrame 的操作,本节只讲解了部分常用的操作方式,读者可通过查阅 Spark 官网详细学习 DataFrame 的操作方式。

3.2.4 DataFrame 的函数操作

Spark SQL 提供了一系列函数对 DataFrame 进行操作,能够实现对数据进行多样化的处理和分析,这些函数操作主要包括标量函数(Scalar Functions)操作和聚合函数(Aggregate Functions)操作,它们同样支持 DSL 风格和 SQL 风格操作 DataFrame,鉴于使用 SQL 风格操作 DataFrame 较为简单,本节使用 DSL 风格重点介绍这两种类型函数的操作。

1. 标量函数操作

标量函数操作是对于输入的每一行数据,函数会产生单个值作为输出。标量函数分为内置标量函数(Built-in Scalar Functions)操作和自定义标量函数(User-Defined Scalar Functions)操作,关于内置标量函数操作和自定义标量函数操作的介绍如下。

(1) 内置标量函数。

Spark SQL 提供了大量的内置标量函数供用户直接使用。下面介绍 Spark SQL 常用的内置标量函数,如表 3-2 所示。

表 3-2　Spark SQL 常用的内置标量函数

函　数	语 法 格 式	说　明
array_max	array_max(col)	用于对 DataFrame 中数组类型的列 col 进行操作，获取每个数组的最大值
array_min	array_min(col)	用于对 DataFrame 中数组类型的列 col 进行操作，获取每个数组的最小值
map_keys	map_keys(col)	用于对 DataFrame 中键值对类型的列 col 进行操作，获取每个键值对的键
map_values	map_values (col)	用于对 DataFrame 中键值对类型的列 col 进行操作，获取每个键值对的值
element_at	element_at(col,key)	用于对 DataFrame 中键值对类型的列 col 进行操作，根据指定的键 key 返回对应的值，如果键不存在，则返回 null
date_add	date_add(startDate, num_days)	用于在指定日期 startDate 上增加天数 num_days。startDate 可以是日期类型的列也可以是字符串类型的日期
datediff	datediff(endDate, startDate)	用于计算两个日期 startDate 和 endDate，之间的天数差异。startDate 和 endDate 可以是日期类型的列也可以是字符串类型的日期
substring	substring(str, pos, len)	用于对 DataFrame 中字符串类型的列进行操作，从字符串中截取部分字符串。其中参数 str 为初始字符串或列；参数 pos 为提取部分字符串的索引位置，从 1 开始；参数 len 指定截取部分字符串的长度。如初始字符串为 world，索引位置为 1，截取长度为 2，则截取后的字符串为 wo。如果索引位置超过初始字符串的长度，则截取后的字符串为空，如果截取长度超过索引位置之后字符串的长度，则将索引位置之后字符串全部截取

在表 3-2 中，内置标量函数对数组和键值对类型数据的操作在 Python 中指代的是列表和字典。以 PyCharm 为例，演示表 3-2 中常用内置标量函数的使用，具体内容如下。

① array_max 函数。在项目 Python_Test 中创建 Function 文件夹，并在该文件夹中创建名为 FunTest 的 Python 文件，通过 array_max 函数获取 DataFrame 中数组类型列的最大值，具体代码如文件 3-2 所示。

文件 3-2　FunTest.py

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
spark = SparkSession.builder.master("local[*]") \
    .appName("FunTest") \
    .getOrCreate()
data = spark.createDataFrame(
    [(
        [80, 88, 68],
        {"xiaohong":"B","xiaoming":"A","xiaoliang":"C"},
        "2023-10-15",
```

```
        "2023-10-16"
    )],
    ["数学分数","学生评级","考试时间","成绩公布时间"]
)
result = data.select("数学分数",array_max("数学分数"))
# 参数 truncate = false 用于指定显示 DataFrame 中完整的行内容
result.show(truncate=False)
# 释放资源
spark.stop()
```

在文件 3-2 中,第 6～14 行代码通过 createDataFrame()方法创建一个名为 data 的 DataFrame。createDataFrame()方法的第一个参数通过列表指定 data 中的数据,列表的每个元素将作为 data 的每行数据。当列表中元素的类型为元组时,元组中的每个元素将依次作为 data 中每个列的数据;第二个元素通过列表指定 data 中每个列的列名,列的数据类型将通过其存储的数据自行推断。

第 15 行代码通过 select()方法选择 data 中的“数学分数”列,并通过 array_max 函数获取“数学分数”列的最大值。

文件 3-2 的运行结果如图 3-13 所示。

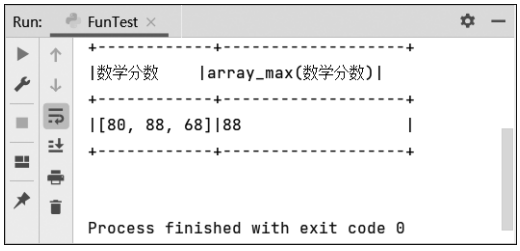

图 3-13 文件 3-2 的运行结果(1)

从图 3-13 可以看出,“数学分数”列的最大值为 88。

② array_min 函数。通过 array_min 函数获取 DataFrame 中数组类型列的最小值。这里将文件 3-2 中第 15 行代码修改为如下代码。

```
result = data.select("数学分数",array_min("数学分数"))
```

上述代码通过 select()方法选择 data 中的“数学分数”列,并通过 array_min 函数获取“数学分数”列的最小值。

文件 3-2 的运行结果如图 3-14 所示。

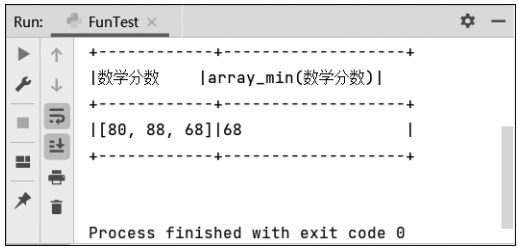

图 3-14 文件 3-2 的运行结果(2)

从图 3-14 可以看出，“数学分数”列的最小值为 68。

③ map_keys 函数。通过 map_keys 函数获取 DataFrame 中键值对类型列的键。这里将文件 3-2 中第 15 行代码修改为如下代码。

```
result = data.select("学生评级",map_keys("学生评级"))
```

上述代码通过 select()方法选择 data 中的“学生评级”列，并通过 map_keys 函数获取“学生评级”列中的键。

文件 3-2 的运行结果如图 3-15 所示。

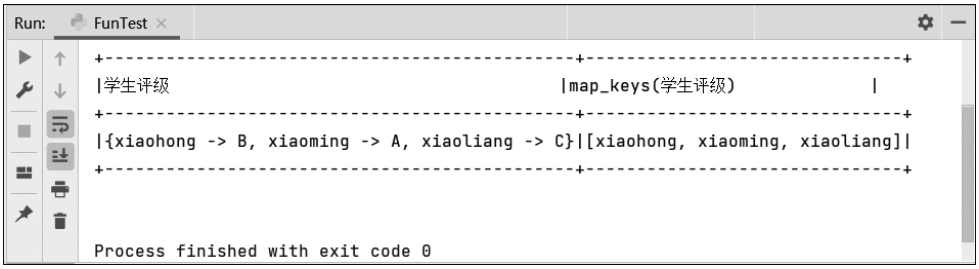

图 3-15　文件 3-2 的运行结果(3)

从图 3-15 可以看出，“数学评级 ”列中的键包括 xiaohong、xiaoming 和 xiaoliang。

④ map_values 函数。通过 map_values 函数获取 DataFrame 中键值对类型列的值。这里将文件 3-2 中第 15 行代码修改为如下代码。

```
result = data.select("学生评级",map_values("学生评级"))
```

上述代码通过 select()方法选择 data 中的“学生评级”列，并通过 map_values 函数获取“学生评级”列中的值。

文件 3-2 的运行结果如图 3-16 所示。

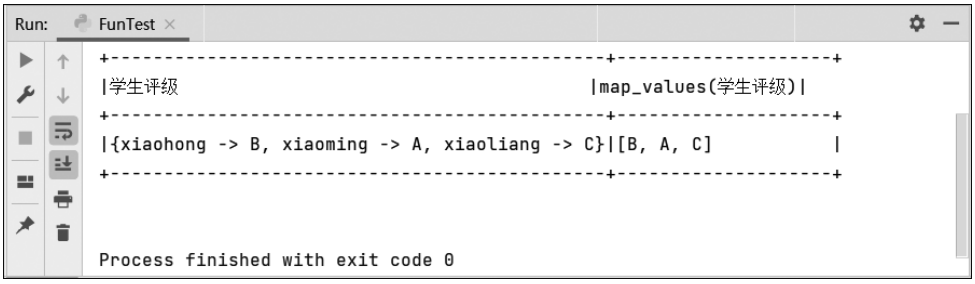

图 3-16　文件 3-2 的运行结果(4)

从图 3-16 可以看出，“数学评级”列中的值包括 B、A 和 C。

⑤ element_at 函数。通过 element_at 函数获取 DataFrame 中键值对类型列指定键对应的值。这里将文件 3-2 中第 15 行代码修改为如下代码。

```
result = data.select("学生评级",element_at("学生评级","xiaoming"))
```

上述代码通过 select()方法选择 data 中的“学生评级”列，并通过 element_at 函数获取

“学生评级”列中键为 xiaoming 的值。

文件 3-2 的运行结果如图 3-17 所示。

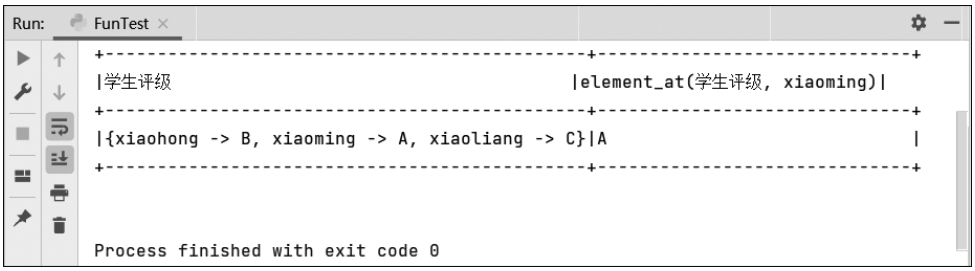

图 3-17 文件 3-2 的运行结果(5)

从图 3-17 可以看出,“学生评级”列中键为 xiaoming 的值为 A。

⑥ date_add 函数。通过 date_add 函数实现对 DataFrame 中字符串日期类型列增加指定的天数。这里将文件 3-2 中第 15 行代码修改为如下代码。

```
result = data.select("考试时间",date_add("考试时间",3))
```

上述代码通过 select()方法选择 data 中的“考试时间”列,并通过 date_add 函数将“考试时间”列中的日期增加 3 天。

文件 3-2 的运行结果如图 3-18 所示。

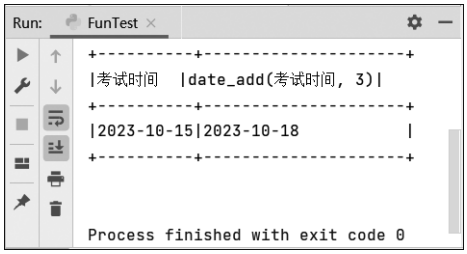

图 3-18 文件 3-2 的运行结果(6)

从图 3-18 可以看出,“考试时间”列中的日期增加 3 天的结果为 2023-10-18。

⑦ datediff 函数。通过 datediff 函数实现计算 DataFrame 中字符串日期类型列的时间间隔。这里将文件 3-2 中第 15 行代码修改为如下代码。

```
result = data.select("考试时间","成绩公布时间",datediff("成绩公布时间","考试时间"))
```

上述代码通过 select()方法选择 data 中的“考试时间”和“成绩公布时间”列,并通过 datediff 函数计算“考试时间”和“成绩公布时间”列的时间间隔。

文件 3-2 的运行结果如图 3-19 所示。

从图 3-19 可以看出,“考试时间”和“成绩公布时间”列的时间间隔为 1。

⑧ substring 函数。通过 substring 函数实现对 DataFrame 中字符串类型的列截取部分字符串。这里将文件 3-2 中第 15 行代码修改为如下代码。

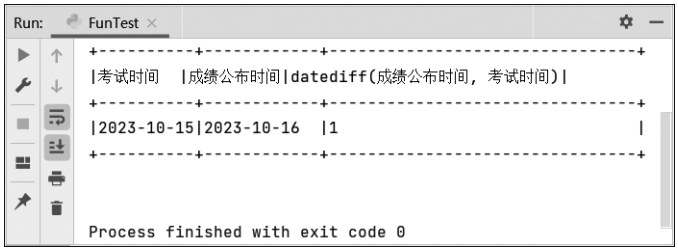

图 3-19　文件 3-2 的运行结果(7)

```
result = data.select("考试时间",substring("考试时间",0,4))
```

上述代码通过 select()方法选择 data 中的“考试时间”列，并通过 substring 函数从“考试时间”列中截取索引位置为 0，截取长度为 4 的字符串。

文件 3-2 的运行结果如图 3-20 所示。

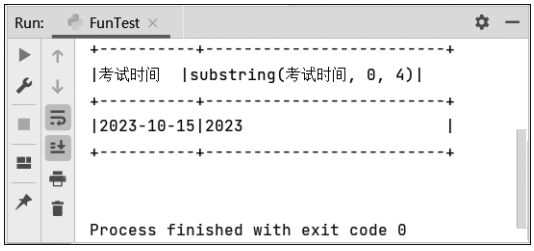

图 3-20　文件 3-2 的运行结果(8)

从图 3-20 可以看出，“考试时间”列中截取索引位置为 0，截取长度为 4 的字符串为 2023。

(2) 自定义标量函数。

自定义标量函数是指内置标量函数不足以处理指定需求时，用户可以自行定义的函数，它可以在程序中添加自定义的功能实现对 DataFrame 进行操作。

在 Spark SQL 中实现自定义标量函数分为定义函数和注册函数两部分操作，其中定义函数用于指定处理逻辑；注册函数用于将定义的函数注册到 SparkSession 中，使其成为 Spark SQL 中的标量函数，定义函数的语法格式如下。

```
def fun_name([参数列表]):
  函数体
  [return value]
```

上述语法格式的解释如下。

① def：定义函数的关键字。

② fun_name：用于指定函数的名称。

③ [参数列表]：负责接收传入函数中的参数，可以包含一个或多个参数，也可以为空。

④ 函数体：指定函数的处理逻辑。

⑤ [return value]：指定函数的返回值 value。如果函数没有返回值，可以省略。

注册函数针对使用 DSL 风格和 SQL 风格操作 DataFrame 具有不同的语法格式，具体

如下。

```
# 使用 DSL 风格操作 DataFrame 时的注册函数
udf_fun = udf(fun_name, returnType)
# 使用 SQL 风格操作 DataFrame 时的注册函数
spark.udf.register(name, fun_name, returnType)
```

上述语法格式中,使用 DSL 风格操作 DataFrame 时的注册函数中,udf()方法用于将定义的函数注册为自定义标量函数,该函数接收两个参数,fun_name 参数为定义的函数名,returnType 参数为自定义标量函数返回值的数据类型。

使用 SQL 风格操作 DataFrame 时的注册函数中,通过调用 SparkSession 对象的 udf()方法获取一个 UDFRegistration 对象,该对象的 register()方法用于将定义的函数注册为自定义标量函数,该方法接收 3 个参数,name 参数为自定义标量函数的函数名,fun_name 参数为定义的函数名,returnType 参数为自定义标量函数返回值的数据类型。

接下来,以 PyCharm 为例,演示自定义标量函数的使用。在项目 Python_Test 的 Function 文件夹中创建名为 UDFTest 的 Python 文件,实现将 DataFrame 中每个单词的第 3 个字母变为大写,具体代码如文件 3-3 所示。

文件 3-3 UDFTest.py

```
1  from pyspark.sql import SparkSession
2  from pyspark.sql.functions import udf
3  from pyspark.sql.types import StringType
4  spark = SparkSession.builder.master("local[*]") \
5      .appName("UDFTest") \
6      .getOrCreate()
7  data = [
8      (1, "hello"),
9      (2, "world"),
10     (3, "spark")
11 ]
12 schema = ["id", "value"]
13 df = spark.createDataFrame(data, schema)
14 def Up(word):
15     if len(word) >= 3:
16         return word[:2] + word[2].upper() + word[3:]
17     else:
18         return word
19 udf_up = udf(Up, StringType())
20 result = df.select("id", udf_up("value"))
21 result.show()
22 spark.stop()
```

在文件 3-3 中,第 14～18 行代码通过定义一个名为 Up 的函数用于将 DataFrame 中每个单词的第 3 个字母变为大写,接收参数 word 表示 DataFrame 中的每个单词。实现逻辑为通过 if-else 语句判断的单词长度是否大于或等于 3,如果满足条件则通过索引切片操作将单词的第 3 个字母通过 upper()方法转换为大写,并将其余字母拼接在一起。如果不满

足条件则返回原始单词。

第 19 行代码通过 udf()方法将名为 Up 的函数注册为自定义标量函数 udf_up，指定自定义标量函数返回值的数据类型为字符串类型。

第 20 行代码通过 select()方法选择 df 中的 id 列，并通过标量函数 udf_up 将 value 列中每个单词的第 3 个字母变为大写。

文件 3-3 的运行结果如图 3-21 所示。

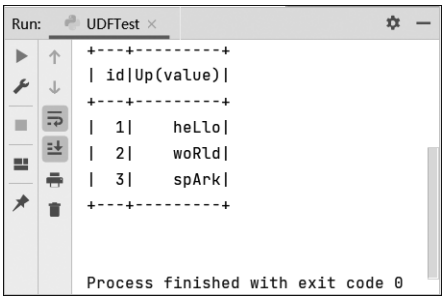

图 3-21　文件 3-3 的运行结果

从图 3-21 可以看出，value 列中每个单词的第 3 个字母已变为大写。

2. 聚合函数操作

聚合函数操作是对于一组数据进行计算并返回单个值的函数。聚合函数分为内置聚合函数(Built-in Aggregation Functions)操作和自定义聚合函数(User Defined Aggregate Functions)操作。关于内置聚合函数操作和自定义聚合函数操作的介绍如下。

(1) 内置聚合函数。

Spark SQL 提供了大量的内置聚合函数供用户直接使用。下面介绍 Spark SQL 常用的内置聚合函数，如表 3-3 所示。

表 3-3　Spark SQL 常用的内置聚合函数

函　数	语 法 格 式	相 关 说 明
count	count(col)	用于计算指定列 col 中非空值的数量
sum	sum(col)	计算指定列 col 中所有数值的总和
avg	avg(col)	计算指定列 col 中所有数值的平均值
max	max(col)	计算指定列 col 中所有数值的最大值
min	min(col)	计算指定列 col 中所有数值的最小值
var_samp	var_samp(col)	计算指定列 col 中样本的方差
stddev	stddev(col)	计算指定列 col 中样本的标准差

表 3-3 列举了 Spark SQL 常用的内置聚合函数。在使用这些内置聚合函数时可以配合 agg 函数进行嵌套使用，这是因为 agg 函数允许用户同时使用多个聚合函数对 DataFrame 中指定列进行不同的操作，实现一次性完成多种聚合需求。

接下来，以 PyCharm 为例，演示表 3-3 中常用内置聚合函数的使用。在项目 Python_Test 的 Function 文件夹中创建名为 AggTest 的 Python 文件，实现使用不同的内置聚合函

数对 DataFrame 中指定列进行操作,具体代码如文件 3-4 所示。

文件 3-4 AggTest.py

```
1  from pyspark.sql import SparkSession
2  from pyspark.sql.functions import *
3  spark = SparkSession.builder.master("local[*]") \
4      .appName("AggTest") \
5      .getOrCreate()
6  data = [(3,), (6,), (3,), (4,)]
7  df = spark.createDataFrame(data, ["value"])
8  result = df.agg(
9      count("value"),
10     sum("value"),
11     avg("value"),
12     max("value"),
13     min("value"),
14     var_samp("value"),
15     stddev("value")
16 )
17 result.show(truncate=False)
18 spark.stop()
```

在文件 3-4 中,第 8～16 行代码使用不同的内置聚合函数对 value 列的值进行计算。

文件 3-4 的运行结果如图 3-22 所示。

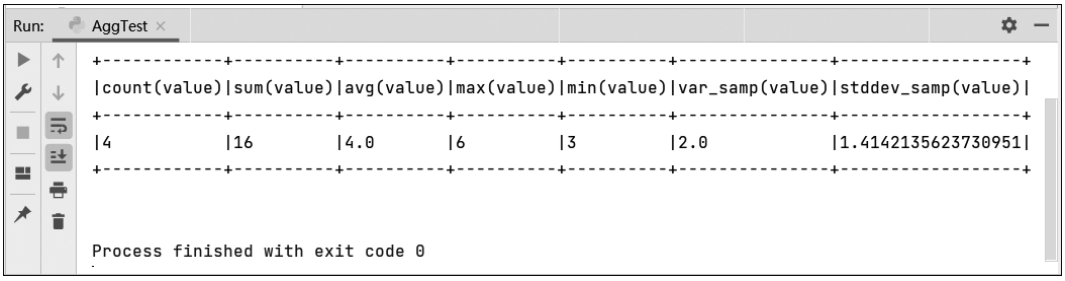

图 3-22 文件 3-4 的运行结果

从图 3-22 可以看出,value 列中非空值的数量为 4、所有数值的总和为 16、所有数值的平均值为 4.0、所有数值的最大值为 6、所有数值的最小值为 3、样本的方差为 2.0、样本的标准差为 1.4142135623730951。

(2) 自定义聚合函数。

自定义聚合函数操作是指内置聚合函数不足以处理指定需求时,用户可以自行定义的函数,它可以在程序中添加自定义的功能实现对 DataFrame 进行操作。

自定义聚合函数同样分为定义函数和注册函数两部分操作,其语法格式与自定义标量函数的语法格式相同,这里不再赘述。

接下来,以 PyCharm 为例,演示自定义聚合函数的使用。在项目 Python_Test 的 Function 文件夹中创建名为 UDAFTest 的 Python 文件,实现从指定列的字符串中提取数值并计算它们相加的结果,具体代码如文件 3-5 所示。

文件 3-5　UDAFTest.py

```
1  from pyspark.sql import SparkSession
2  from pyspark.sql.functions import udf
3  from pyspark.sql.types import IntegerType
4  spark = SparkSession.builder.master("local[*]") \
5      .appName("UDAFTest") \
6      .getOrCreate()
7  data = [("a1b3d2",)]
8  df = spark.createDataFrame(data, ["value"])
9  def str_num(data):
10     num = [int(x) for x in data if x.isdigit()]
11     if not num:
12         return 0
13     else:
14         return sum(num)
15 udaf_num = udf(str_num, IntegerType())
16 result = df.select(udaf_num("value"))
17 result.show(truncate=False)
18 spark.stop()
```

在文件 3-5 中，第 9～14 行代码定义一个名为 str_num 的函数用于从 DataFrame 中指定列的字符串中提取数值并计算它们相加的结果，接收参数 data 表示 DataFrame 中的字符串。实现逻辑为从字符串中提取数值，如果存在数值则返回数值的和，否则返回 0。其中 isdigit()方法用于判断字符串中是否存在数值，若存在则保留，然后交由 int()方法将其转换为整数类型。

第 15 行代码通过 udf()方法将名为 str_num 的函数注册为自定义聚合函数 udaf_num，指定自定义聚合函数返回值的数据类型为整数类型。

第 16 行代码在 select()方法中通过自定义聚合函数 udaf_num 从 value 列的字符串中提取数值并计算它们相加的结果。

文件 3-5 的运行结果如图 3-23 所示。

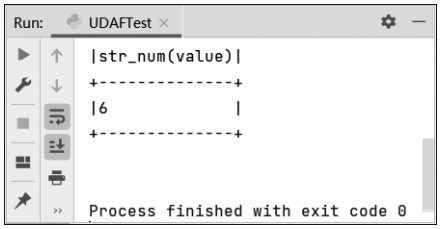

图 3-23　文件 3-5 的运行结果

从图 3-23 可以看出，value 列的字符串中数值相加的结果为 6。

多学一招：通过@udf()装饰器实现自定义标量函数或聚合函数

从 Spark 1.3.0 版本开始，Spark SQL 引入了@udf()装饰器来实现自定义标量函数或聚合函数，通过该装饰器实现自定义标量函数或聚合函数时，无须手动对定义的函数进行注册便可直接使用。这种方式简化了用户实现自定义标量函数或聚合函数的过程，提高了代

码的可读性和可维护性,不过通过@udf()装饰器实现自定义标量函数或聚合函数时,只能通过 DSL 风格操作 DataFrame,语法格式如下。

```
@udf(returnType)
def fun_name([参数列表]):
  函数体
  [return value]
```

上述语法格式中,returnType 参数表示自定义标量函数或聚合函数返回值的数据类型。

3.3 RDD 转换为 DataFrame

当 RDD 无法满足用户更高级别、更高效的数据分析时,可以将 RDD 转换为 DataFrame。Spark 提供了两种方法实现将 RDD 转换为 DataFrame,第一种方法是利用反射机制来推断包含特定类型对象的 Schema 元数据信息,这种方式适用于对已知数据结构的 RDD 转换;第二种方法通过编程方式定义一个 Schema,并将其应用在已知的 RDD 中。接下来,本节针对这两种转换方法进行讲解。

3.3.1 反射机制推断 Schema

当有一个数据文件时,人类可以轻松理解其中的字段,如编号、姓名和年龄的含义,但计算机无法像人一样直观地理解这些字段。在这种情况下,可以通过反射机制来自动推断包含特定类型对象的 Schema 元数据信息。这个 Schema 元数据信息可以帮助计算机更好地理解和处理数据文件中的字段。

通过反射机制推断 Schema 主要包含两个步骤,具体如下。

① 创建一个 ROW 类型的 RDD。

② 通过 toDF()方法根据 Row 对象中的列名来推断 Schema 元数据信息。

上述步骤对应的语法格式如下。

```
schema = data.map(lambda y: Row(fieldName=y[0], fieldName=y[1],
  fieldName=y[2], ...))
df = schema.toDF()
```

上述语法格式中,通过 map 算子中定义的匿名函数,将名为 data 的 RDD 中每个元素与 Row 对象进行映射操作,生成名为 schema 的 RDD。在 Row 对象中,fieldName 用于指定列名。映射操作完成后,通过 toDF()方法根据映射的列名来推断 Schema 元数据信息,将 schema 转换成名为 df 的 DataFrame。

接下来,以 PyCharm 为例,实现通过反射机制推断 Schema,具体操作步骤如下。

(1) 在本地计算机中准备文本数据文件,这里在本地计算机 D 盘根目录下创建文件 person.txt,数据内容如文件 3-6 所示。