SL-20 6.2 结构化数据DataFrame

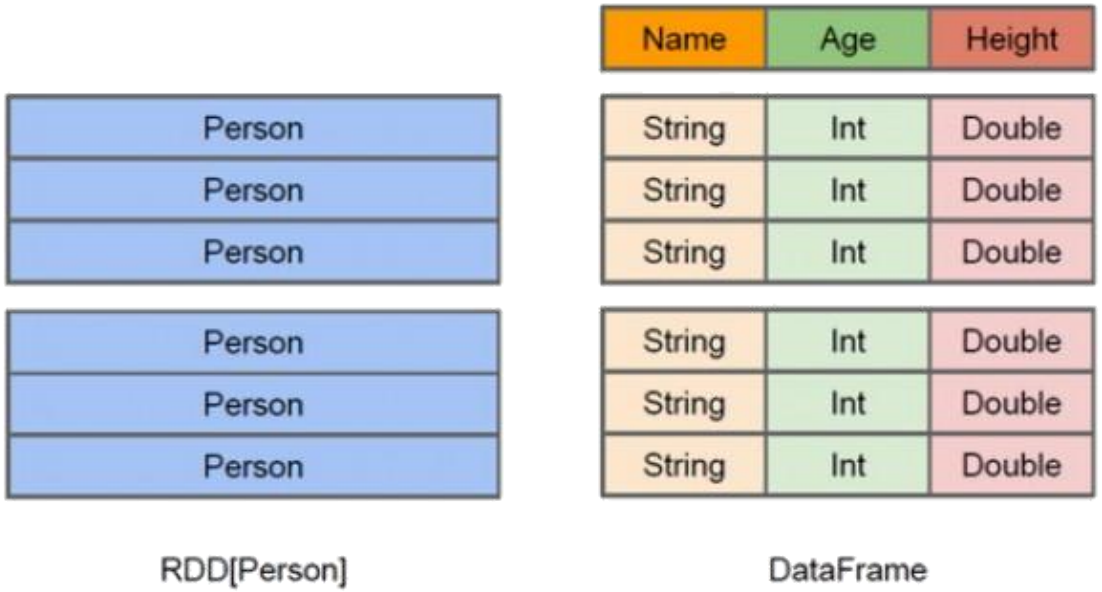
6.2.1 DataFrame概述

6.2.2 DataFrame的优点

SL-21 6.2.1 DataFrame概述



- DataFrame的推出，让Spark具备了处理大规模结构化数据的能力，不仅比原有的RDD转化方式更加简单易用，而且获得了更高的计算性能
- Spark能够轻松实现从MySQL到DataFrame的转化，并且支持SQL查询



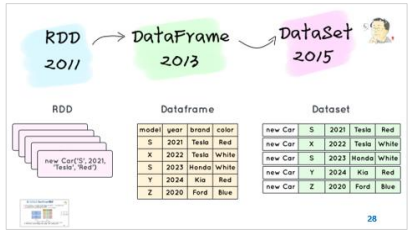


图6-5 DataFrame与RDD的区别

- RDD是分布式的 Java对象的集合，但是，对象内部结构对于RDD而言却是不可知的
- DataFrame是一种以RDD为基础的分布式数据集，提供了详细的结构信息

SL-22 6.2.2 DataFrame的优点



使用DataFrame作为数据抽象，可以带来很多好处，

比如，可以在Spark组件间获得更好的性能和更优的空间效率。

DataFrame的突出优点是表达能力强、简洁、易组合、风格一致。

下面用一个实例来展示DataFrame强大的表达能力和组合能力。

SL-23 6.2.2 DataFrame的优点



给定一组键值对("spark",2)、("hadoop",6)、("hadoop",4)、("spark",6),
键值对的key表示图书名称，value表示某天图书销量，现在需要计算每个键对应的平均值，也就是计算每种图书的每天平均销量。当使用RDD编程时，程序如下：

```
# coding:utf8
from pyspark import SparkConf, SparkContext
if __name__ == '__main__':
    conf = SparkConf().setAppName("Simple App")
    sc = SparkContext(conf=conf)
    bookRDD = sc.parallelize([("spark", 2), ("hadoop", 6), ("hadoop", 4), ("spark", 6)])
    saleRDD = bookRDD.map(lambda x: (x[0], (x[1], 1))) ₩
        .reduceByKey(lambda x, y : (x[0] + y[0], x[1] + y[1])) ₩
        .map(lambda x : (x[0], x[1][0] / x[1][1]))
    saleRDD.foreach(print)
    sc.stop()
```

SL-24 6.2.2 DataFrame的优点



可以看出，这段代码难度较高，可读性也较差。这段代码是在告诉Spark如何计算出查询结果。这个过程对于Spark而言是完全不透明的，因为代码没有告诉Spark最终目的是什么。相反，如果是使用DataFrame API来表达相同的查询，就会简单很多。使用DataFrame API编写的程序如下：

```
# coding:utf8
from pyspark.sql import SparkSession
from pyspark.sql.functions import avg
if __name__ == '__main__':
    spark = SparkSession₩
        .builder₩
        .master("local[*]")₩
        .appName("Simple Application")₩
        .getOrCreate()
    bookDF = spark ₩
    .createDataFrame([("spark", 2), ("hadoop", 6), ("hadoop", 4), ("spark", 6)]).toDF("book", "amount")
    avgDF = bookDF.groupBy("book").agg(avg("amount"))
    avgDF.show()
    spark.stop()
```

SL-25 6.2.2 DataFrame的优点



程序的执行结果如下：

```
+------+-----------+
|  book|avg(amount)|
+------+-----------+
| spark|        4.0|
|hadoop|        5.0|
+------+-----------+
```

可以看出，采用DataFrame编程以后，代码的表达能力和简洁程度都提高了很多，因为我们用高层的DSL（Domain Specified Language）算子和API告诉Spark去做什么。

实际上，我们已经用这些算子组合出了自己的查询。因为可以解析这条查询并理解我们的最终目的，所以，Spark能对操作进行优化或重排，从而实现更高的执行效率。

Spark明确知道我们要做什么：按书名对图书进行分组，聚合每种图书的销量，然后计算出每种图书的每天平均销量。

整个计算过程已经被我们用高层的算子组合成了仅仅一条查询，可见这种API的表达能力是很强的。



File Edit View Navigate Code Refactor Run Tools VCS Window Help

ch06SparkSQL 〉 p138_Book_Sale_RDD.py

p138_Book_Sale_RDD

p138_Book_Sale_RDD.py

```
1  # coding:utf8
2  from pyspark import SparkConf, SparkContext
3  if __name__ == '__main__':
4      conf = SparkConf().setAppName("Simple App")
5      sc = SparkContext(conf=conf)
6      saleList=[("spark", 2), ("hadoop", 6), ("hadoop", 4), ("spark", 6)]
7      bookRDD = sc.parallelize(saleList)
8      saleRDD = bookRDD.map(lambda x: (x[0], (x[1], 1))) \
9          .reduceByKey(lambda x, y : (x[0] + y[0], x[1] + y[1])) \
10         .map(lambda x : (x[0], x[1][0] / x[1][1]))
11     saleRDD.foreach(print)
12     sc.stop()
```

if __name__ == '__main__'

```
/home/hadoop/anaconda3/envs/nanyang/bin/python /home/hadoop/PycharmProjects/ch0
('spark', 4.0)
('hadoop', 5.0)

Process finished with exit code 0
```

Version Control | Run | TODO | Problems | Terminal | Python Packages | Python Console | Services

6:1 LF UTF-8 4 spaces Python 3.8 (nanyang)



File Edit View Navigate Code Refactor Run Tools VCS Window Help

ch06SparkSQL 〉 p138_Book_Sale_DataFrame.py

p138_Book_Sale_DataFrame

p138_Book_Sale_RDD.py × p138_Book_Sale_DataFrame.py ×

```
3  from pyspark.sql.functions import avg
4  if __name__ == '__main__':
5      spark = SparkSession\
6          .builder\
7          .master("local[*]")\
8          .appName("Simple Application")\
9          .getOrCreate()
10     saleList = [("spark", 2), ("hadoop", 6), ("hadoop", 4), ("spark", 6)]
11     bookDF = spark.createDataFrame(saleList).toDF("book", "amount")
12     avgDF = bookDF.groupBy("book").agg(avg("amount"))
13     avgDF.show()
14     spark.stop()
```

if __name__ == '__main__'

Run: p138_Book_Sale_DataFrame ×

```
/home/hadoop/anaconda3/envs/nanyang/bin/python /home/hadoop/PycharmProjects/ch06SparkSQL/p1
+------+-----------+
|  book|avg(amount)|
+------+-----------+
| spark|        4.0|
|hadoop|        5.0|
+------+-----------+
```

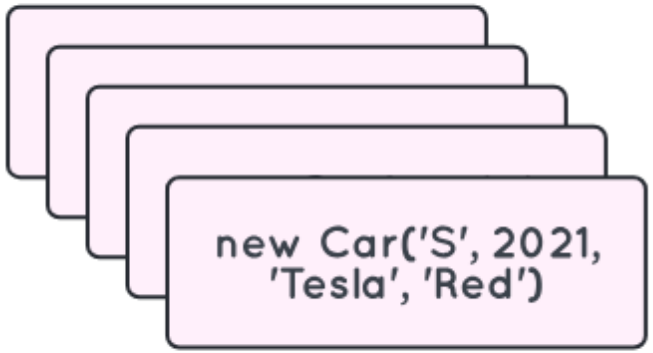
Version Control | Run | TODO | Problems | Terminal | Python Packages | Python Console | Services

6:18 LF UTF-8 4 spaces Python 3.8 (nanyang)

RDD 2011 → DataFrame 2013 → DataSet 2015



RDD



Dataframe

model	year	brand	color
S	2021	Tesla	Red
X	2022	Tesla	White
S	2023	Honda	White
Y	2024	Kia	Red
Z	2020	Ford	Blue

Dataset

new Car	S	2021	Tesla	Red
new Car	X	2022	Tesla	White
new Car	S	2023	Honda	White
new Car	Y	2024	Kia	Red
new Car	Z	2020	Ford	Blue

