What is the purpose of the PySpark transform() function and how can it be used in a practical scenario with an example?

Authored by

stats writer

June 24, 2024

The PySpark transform() function is used to apply a function to each element in a specified Spark Resilient Distributed Dataset (RDD). Its purpose is to transform the data within an RDD into a different format or structure, allowing for easier analysis and manipulation. This function is commonly used in big data processing and machine learning applications.

Practically, this function can be used in scenarios where data needs to be manipulated before further analysis. For example, let's say we have a dataset containing information about customer purchases. We want to calculate the total amount spent by each customer and return the result in a new RDD. We can use the transform() function to map each purchase record in the original RDD to the corresponding customer and then reduce the data by summing up the total amount spent by each customer. This transformed RDD can then be used for further analysis, such as identifying top-spending customers or creating personalized marketing strategies. Overall, the transform() function allows for efficient data processing and helps to extract valuable insights from large datasets.

PySpark provides two transform() functions one with DataFrame and another in pyspark.sql.functions.

In this article, I will explain the syntax of these two functions and explain with examples. First, let's create the DataFrame.

```
# Imports
from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder
.appName('SparkByExamples.com')
.getOrCreate()

# Prepare Data
simpleData = (("Java",4000,5),
("Python", 4600,10),
("Scala", 4100,15),
("Scala", 4500,15),
("PHP", 3000,20),
)
columns=

# Create DataFrame
df = spark.createDataFrame(data = simpleData, schema = columns)
df.printSchema()
```

df.show(truncate=False)

1. PySpark DataFrame.transform()

The pyspark.sql.DataFrame.transform() is used to chain the custom transformations and this function returns the new DataFrame after applying the specified transformations.

This function always returns the same number of rows that exists on the input PySpark DataFrame.

1.1 Syntax

Following is the syntax of the `pyspark.sql.DataFrame.transform()` function

```
# Syntax
DataFrame.transform(func: Callable, DataFrame], *args: Any, **kwargs: Any) → pyspark.sql.dataframe.DataFrame
```

The following are the parameters:

1.2 Create Custom Functions

In the below snippet, I have created the three custom transformations to be applied to the DataFrame. These transformations are nothing but Python functions that take the DataFrame apply some changes and return the new DataFrame.

```
# Custom transformation 1
from pyspark.sql.functions import upper
def to_upper_str_columns(df):
return df.withColumn("CourseName",upper(df.CourseName))
```

# Custom transformation 2
def reduce_price(df,reduceBy):
return df.withColumn("new_fee",df.fee - reduceBy)

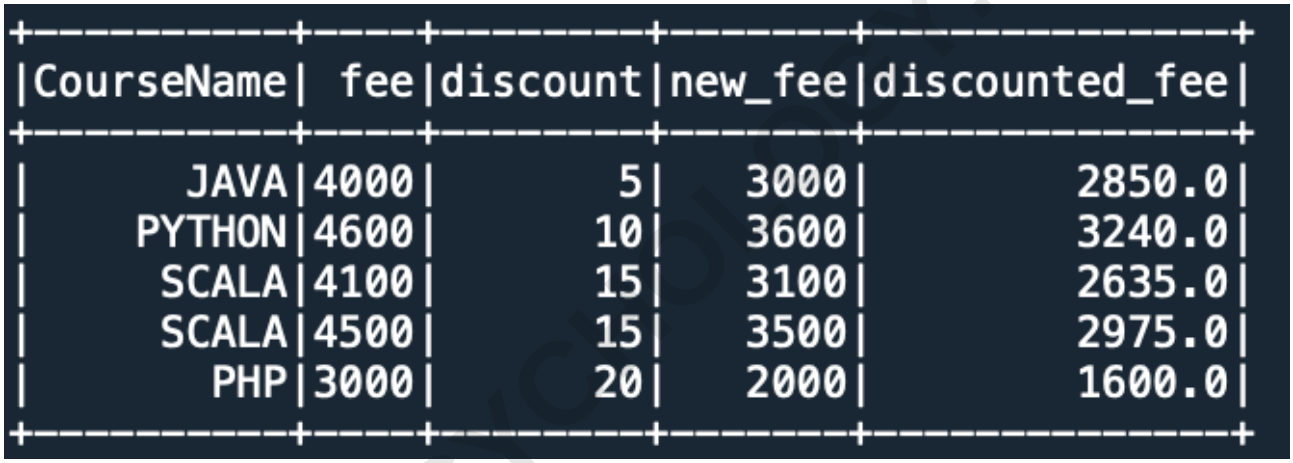
# Custom transformation 3
def apply_discount(df):
return df.withColumn("discounted_fee",
df.new_fee - (df.new_fee * df.discount) / 100)

1.3 PySpark Apply DataFrame.transform()

Now, let's chain these custom functions together and run them using PySpark DataFrame transform() function.

```
# PySpark transform() Usage
df2 = df.transform(to_upper_str_columns)
.transform(reduce_price,1000)
.transform(apply_discount)
```

Yields the below output.



CourseName	fee	discount	new_fee	discounted_fee
JAVA	4000	5	3000	2850.0
PYTHON	4600	10	3600	3240.0
SCALA	4100	15	3100	2635.0
SCALA	4500	15	3500	2975.0
PHP	3000	20	2000	1600.0

In case you wanted to select the columns either you can chain it with select() or create another custom function.

```
# custom function
def select_columns(df):
return df.select("CourseName","discounted_fee")
```

# Chain transformations
df2 = df.transform(to_upper_str_columns)
.transform(reduce_price,1000)
.transform(apply_discount)
.transform(select_columns)

I will leave this to you to run and explore the output.

2. PySpark sql.functions.transform()

The PySpark sql.functions.transform() is used to apply the transformation on a column of type Array. This function applies the specified transformation on every element of the array and returns an object of ArrayType.

2.1 Syntax

Following is the syntax of the `pyspark.sql.functions.transform()` function

```
# Syntax
pyspark.sql.functions.transform(col, f)
```

The following are the parameters:

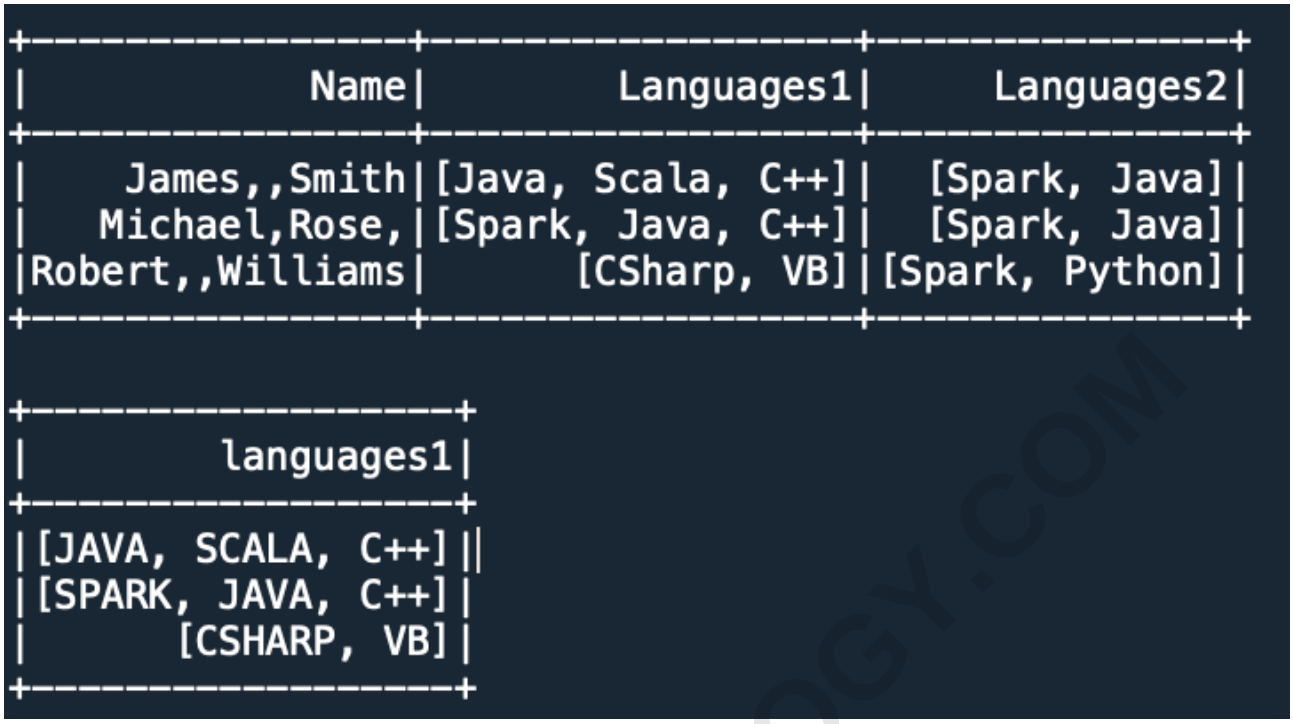
2.2 Example

Since our above DataFrame doesn't contain ArrayType, I will create a new simple array to explain.

```
# Create DataFrame with Array
data = ,),
("Michael,Rose,",,),
("Robert,,Williams",,)
]
df = spark.createDataFrame(data=data,schema=)
df.printSchema()
df.show()
```

# using transform() function
from pyspark.sql.functions import upper
from pyspark.sql.functions import transform
df.select(transform("Languages1", lambda x: upper(x)).alias("languages1"))
.show()

Yields below output.



3. Complete Example

Following is the complete example of PySpark transform() function

```
# Imports
from pyspark.sql import SparkSession
```

```
# Create SparkSession
spark = SparkSession.builder
.appName('SparkByExamples.com')
.getOrCreate()

# Prepare Data
simpleData = (("Java",4000,5),
("Python", 4600,10),
("Scala", 4100,15),
("Scala", 4500,15),
("PHP", 3000,20),
)
columns=

# Create DataFrame
```

```
df = spark.createDataFrame(data = simpleData, schema = columns)
df.printSchema()
df.show(truncate=False)

# Custom transformation 1
from pyspark.sql.functions import upper
def to_upper_str_columns(df):
return df.withColumn("CourseName",upper(df.CourseName))

# Custom transformation 2
def reduce_price(df,reduceBy):
return df.withColumn("new_fee",df.fee - reduceBy)

# Custom transformation 3
def apply_discount(df):
return df.withColumn("discounted_fee",
df.new_fee - (df.new_fee * df.discount) / 100)

# transform() usage
df2 = df.transform(to_upper_str_columns)
.transform(reduce_price,1000)
.transform(apply_discount)

df2.show()

# Create DataFrame with Array
data = ,),
("Michael,Rose,",,),
("Robert,,Williams",,)
]
df = spark.createDataFrame(data=data,schema=)
df.printSchema()
df.show()

# using transform() SQL function
from pyspark.sql.functions import upper
from pyspark.sql.functions import transform
df.select(transform("Languages1", lambda x: upper(x)).alias("languages1"))
.show()
```

4. Conclusion

In this article, you have learned the transform() function from pyspark.sql.DataFrame class and pyspark.sql.functions package.

Related Articles