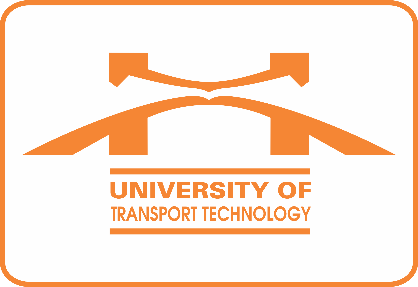
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ GIAO THÔNG VẬN TẢI**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BÀI TẬP NHÓM**

***Đề tài:***

**TÌM HIỂU VỀ RDD CẶP KHÓA/GIÁ TRỊ**

**MÔN HỌC: BIG DATA**

**Ngành: Công nghệ thông tin**

**Giảng viên hướng dẫn:** Đỗ Bảo Sơn

**Lớp:** 73DCTT23

**Sinh viên thực hiện:** 1. Trương Ngọc Duy

2. Nguyễn Huyền Nhung

3. Bùi Thu Phương

4. Đỗ Thị Phương

5. Nguyễn Như Xuân

**HÀ NỘI, tháng 3 / 2025**

Nội dung

[1. RDD cặp là gì? 3](#_Toc192273131)

[2. Khởi tạo RDD cặp 3](#_Toc192273132)

[2.1. Tạo RDD cặp từ RDD thông thường 3](#_Toc192273133)

[2.2. Tạo RDD cặp từ file bên ngoài 4](#_Toc192273134)

[3. Các câu lệnh biến đổi trên RDD cặp 4](#_Toc192273135)

[3.1. Các câu lệnh tổng hợp 4](#_Toc192273136)

[3.2. Câu lệnh sắp xếp 7](#_Toc192273137)

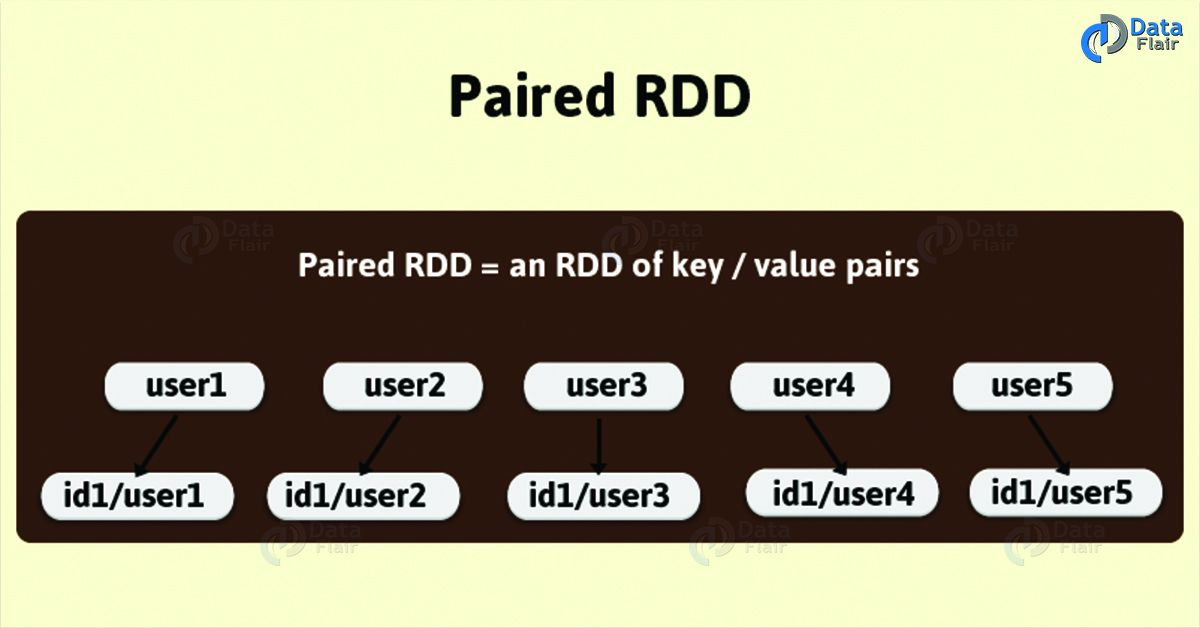
[3.3. Các câu lệnh nhóm dữ liệu 8](#_Toc192273138)

[3.4. Các câu lệnh kết nối 9](#_Toc192273139)

[3.5. Một số câu lệnh khác 11](#_Toc192273140)

## RDD cặp là gì?

Trong Spark, RDD cặp là một RDD chứa một cặp khóa/giá trị. Về cơ bản, cặp khóa/giá trị bao gồm hai mục dữ liệu được liên kết với nhau. Trong đó, khóa là mã dịnh danh, còn giá trị là dữ liệu tương ứng với khóa đó.



## Khởi tạo RDD cặp

### Tạo RDD cặp từ RDD thông thường

Có một số cách để có RDD cặp trong Spark. Trong một số trường hợp, chúng ta có một RDD thông thường mà chúng ta muốn chuyển thành RDD cặp, chúng ta có thể thực hiện điều này bằng cách chạy hàm *map()* trả về cặp khóa/giá trị.

Ví dụ 1: Tạo RDD cặp sử dụng từ đầu tiên làm khóa trong Python

*pairs = lines.map(lambda x: (x.split(“ “)[0], x))*

Giải thích ví dụ 1:

- Giả sử *lines* là một RDD thường chứa các dòng.

- *lines.map()* là phép biến đổi RDD thường thành RDD cặp.

- *lambda x: (x.split(“ “)[0], x)* là một hàm lambda (hàm ẩn danh) được áp dụng cho mỗi dòng x trong RDD.

- *lines*. *x.split(“ “)[0]* chia các dòng x thành các từ, sử dụng dấu cách “ “ làm dấu phân cách và lấy phần tử đầu tiên trong danh sách các từ đó.

- Cuối cùng trả về một bộ chứa hai phần tử: phần tử đầu tiên là từ đầu tiên của dòng (khóa), và phần tử thứ hai là toàn bộ dòng (giá trị).

### Tạo RDD cặp từ file bên ngoài

Chúng ta cũng có thể tạo RDD cặp từ một file dữ liệu bên ngoài.

Ví dụ 2: Tạo RDD cặp từ một file dữ liệu bên ngoài trong Python

*pair\_rdd = sc.textFile("path/to/file.txt").map(lambda line: line.split(",")).map(lambda parts: (parts[0], parts[1]))*

Giải thích ví dụ 2:

- *sc.textFile("path/to/file.txt")*: trả về một RDD thường, trong đó mỗi phần tử là một dòng trong file văn bản từ đường dẫn được chỉ định.

- *.map(lambda line: line.split(","))*: kết quả của phép biến đổi trả về một RDD, trong đó mỗi phần tử là một danh sách các chuỗi được phân cách bằng dấu “,”.

- *.map(lambda parts: (parts[0], parts[1]))*: kết quả của phép biến đổi trả về một RDD cặp, trong đó mỗi phần tử là một cặp khóa/giá trị: khóa là phần tử đầu tiên của chuỗi, giá trị là phần tử thứ hai của chuỗi.

## Các câu lệnh biến đổi trên RDD cặp

### Các câu lệnh tổng hợp

Khi dữ liệu được mô tả theo cặp khóa/giá trị, người ta thường mong muốn tổng hợp số liệu thống kê trên tất cả các phần tử có cùng khóa.

1. **reduceByKey()**

reduceByKey() được sử dụng để kết hợp các giá trị dựa trên cùng một khóa.

Ví dụ 3: Sử dụng hàm reduceByKey() trong Python

*from pyspark import SparkContext*

*sc = SparkContext("local", "ReduceByKey Example")*

*data = [("apple", 2), ("banana", 3), ("apple", 4), ("orange", 1), ("banana", 1)]*

*rdd = sc.parallelize(data)*

*result = rdd.reduceByKey(lambda x, y: x + y)*

*print(result.collect())*

Kết quả của *result.collect()* là: [('orange', 1), ('banana', 4), ('apple', 6)]

1. **foldByKey()**

foldByKey() sử dụng một giá trị rỗng cùng loại dữ liệu trong RDD và hàm kết hợp.

Ví dụ 4: Sử dụng hàm foldByKey() trong Python

*from pyspark import SparkContext*

*sc = SparkContext("local", "FoldByKey Example")*

*data = [("apple", 2), ("banana", 3), ("apple", 4), ("orange", 1), ("banana", 1)]*

*rdd = sc.parallelize(data)*

*result = rdd.foldByKey(2, lambda x, y: x + y)*

*print(result.collect())*

Kết quả của *result.collect()* là: [('orange', 3), ('banana', 6), ('apple', 8)]

1. **combineByKey()**

*combineByKey()* là một trong những phép biến đổi mạnh mẽ nhất trong Apache Spark, cho phép bạn thực hiện các phép toán phức tạp trên RDD cặp (Pair RDD). Nó cung cấp sự linh hoạt cao hơn so với *reduceByKey()* hoặc *foldByKey()*, cho phép bạn xử lý dữ liệu theo nhiều cách khác nhau.

Cú pháp:

combineByKey(createCombiner, mergeValue, mergeCombiners, numPartitions)

Trong đó:

**createCombiner**:

* Hàm này được gọi lần đầu tiên cho mỗi key mới.
* Nó chuyển đổi giá trị ban đầu (value) thành một giá trị kết hợp (combiner).

**mergeValue**:

* Hàm này được gọi cho mỗi giá trị (value) tiếp theo có cùng key.
* Nó kết hợp giá trị (value) với giá trị kết hợp (combiner) hiện tại.

**mergeCombiners**:

* Hàm này được gọi để kết hợp các giá trị kết hợp (combiners) từ các phân vùng (partitions) khác nhau.

**numPartitions** (tùy chọn):

* Số lượng phân vùng (partitions) cho RDD kết quả.

Ví dụ 5: Sử dụng hàm combineByKey() trong Python

from pyspark import SparkContext

sc = SparkContext("local", "CombineByKey Example")

data = [("apple", 10), ("banana", 20), ("apple", 30), ("orange", 15), ("banana", 25)]

rdd = sc.parallelize(data)

result = rdd.combineByKey(

lambda value: (value, 1), # createCombiner

lambda acc, value: (acc[0] + value, acc[1] + 1), # mergeValue

lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1]) # mergeCombiners

)

print(result.collect())

Kết quả: [('orange', (15, 1)), ('banana', (45, 2)), ('apple', (40, 2))]

Giải thích:

**createCombiner**: Khi gặp một key mới (ví dụ: "apple" lần đầu tiên), hàm này sẽ chuyển đổi giá trị (ví dụ: 10) thành một tuple (10, 1), trong đó 10 là tổng số lượng và 1 là số lượng sản phẩm.

**mergeValue**: Khi gặp một giá trị khác có cùng key (ví dụ: "apple" với giá trị 30), hàm này sẽ kết hợp giá trị 30 với tuple (10, 1) hiện tại, tạo ra tuple (40, 2).

**mergeCombiners**: Hàm này được sử dụng để kết hợp các tuple từ các phân vùng khác nhau.

### Câu lệnh sắp xếp

Sắp xếp dữ liệu khá hữu ích trong nhiều trường hợp, đặc biệt là khi bạn đang tạo đầu ra. Chúng ta có thể sắp xếp RDD bằng các cặp khóa/giá trị miền là có thứ tự được xác định trên khóa. Sau khi sắp xếp dữ liệu, bất kỳ lệnh gọi tiếp theo nào trên dữ liệu đã sắp xếp để collect() hoặc save() sẽ dẫn đến dữ liệu đã sắp xếp.

Cú pháp:

sortByKey(ascending=True, numPartitions=None, keyfunc=lambda x: x)

- **ascending**: Đây là một tham số boolean. Nếu True (mặc định), các phần tử sẽ được sắp xếp theo thứ tự tăng dần. Nếu False, các phần tử sẽ được sắp xếp theo thứ tự giảm dần.

- **numPartitions**: Đây là một tham số tùy chọn. Nó chỉ định số lượng phân vùng cho RDD kết quả. Nếu không được chỉ định, Spark sẽ sử dụng số lượng phân vùng mặc định.

- **keyfunc**: Đây là một tham số tùy chọn. Đây là một hàm được áp dụng cho key trước khi so sánh. Điều này cho phép bạn sắp xếp dựa trên một giá trị được chuyển đổi từ key.

Ví dụ 6: Sử dụng hàm sortByKey() trong Python

from pyspark import SparkContext

sc = SparkContext("local", "SortByKey Example")

data = [("apple", 10), ("banana", 20), ("orange", 15), ("grape", 5)]

rdd = sc.parallelize(data)

# Sắp xếp theo key (tên sản phẩm) theo thứ tự tăng dần

sorted\_rdd\_asc = rdd.sortByKey()

print("Sắp xếp tăng dần:", sorted\_rdd\_asc.collect())

# Sắp xếp theo key theo thứ tự giảm dần

sorted\_rdd\_desc = rdd.sortByKey(ascending=False)

print("Sắp xếp giảm dần:", sorted\_rdd\_desc.collect())

Kết quả:

Sắp xếp tăng dần: [('apple', 10), ('banana', 20), ('grape', 5), ('orange', 15)]

Sắp xếp giảm dần: [('orange', 15), ('grape', 5), ('banana', 20), ('apple', 10)]

### Các câu lệnh nhóm dữ liệu

1. **groupByKey()**

groupByKey() sẽ nhóm dữ liệu của chúng ta bằng khóa trong RDD. Trên RDD bao gồm các khóa kiểu K và tập các giá trị V, chúng ta sẽ nhận được RDD kiểu [K, [V]].

Ví dụ 7: Sử dụng hàm groupByKey() trong Python

from pyspark import SparkContext

sc = SparkContext("local", "GroupByKey Example")

data = [("apple", 10), ("banana", 20), ("apple", 30), ("orange", 15), ("banana", 25)]

rdd = sc.parallelize(data)

grouped\_rdd = rdd.groupByKey()

# Chuyển đổi ResultIterable thành danh sách Python

result = grouped\_rdd.mapValues(list).collect()

print(result)

Kết quả: [('orange', [15]), ('banana', [20, 25]), ('apple', [10, 30])]

1. **cogroup()**

Ngoài việc nhóm dữ liệu từ một RDD duy nhất, chúng ta có thể nhóm dữ liệu chia sẻ cùng một khóa từ nhiều RDD bằng cách sử dụng một hàm gọi là cogroup(). cogroup() trên hai RDD chia sẻ cùng một loại khóa K, với các loại giá trị tương ứng là V và W trả về cho chúng ta RDD[(K, (Giá\_trị[V], Giá\_trị[W]))]. Nếu một trong các RDD không có các phần tử cho một khóa nhất định có trong RDD kia, thì Giá\_trị tương ứng chỉ đơn giản là rỗng. cogroup() cung cấp cho chúng ta sức mạnh để nhóm dữ liệu từ nhiều RDD.

Ví dụ 8: Sử dụng cogroup() trong Python

from pyspark import SparkContext

sc = SparkContext("local", "Cogroup Example")

# Cửa hàng 1

store1\_sales = sc.parallelize([("apple", 10), ("banana", 20), ("apple", 30)])

# Cửa hàng 2

store2\_sales = sc.parallelize([("apple", 15), ("banana", 25), ("orange", 35)])

# Cogroup dữ liệu từ hai cửa hàng

cogrouped\_sales = store1\_sales.cogroup(store2\_sales)

# Chuyển đổi ResultIterable thành danh sách Python và in kết quả

result = cogrouped\_sales.mapValues(lambda x: (list(x[0]), list(x[1]))).collect()

print(result)

Kết quả: [('orange', ([], [35])), ('banana', ([20], [25])), ('apple', ([10, 30], [15]))]

### Các câu lệnh kết nối

Một số thao tác hữu ích nhất mà chúng ta có được với dữ liệu có khóa đến từ việc sử dụng nó cùng với dữ liệu có khóa khác. Kết nối dữ liệu với nhau có lẽ là một trong những thao tác phổ biến nhất trên một RDD cặp và chúng ta có đầy đủ các tùy chọn bao gồm kết nối phải và trái, kết nối chéo và kết nối trong.

1. **join()**

Toán tử kết nối đơn giản là nối trong. Chỉ các khóa có trong cả hai RDD cặp mới được xuất ra. Khi có nhiều giá trị cho cùng một khóa trong một trong các đầu vào, RDD cặp kết quả sẽ có một mục nhập cho mọi cặp giá trị có thể có với khóa đó từ hai RDD đầu vào.

Ví dụ 9: Kết nối trong trong Python

from pyspark import SparkContext

sc = SparkContext("local", "Join Example")

rdd1 = sc.parallelize([(1, "apple"), (2, "banana"), (3, "orange")])

rdd2 = sc.parallelize([(1, 1.0), (2, 1.5), (4, 2.0)]

# Inner Join

inner\_join\_result = rdd1.join(rdd2)

print("Inner Join:", inner\_join\_result.collect())

Kết quả: [(1, ('apple', 1.0)), (2, ('banana', 1.5))]

1. **leftOuterJoin() / rightOuterJoin()**

Đôi khi chúng ta không cần khóa phải có trong cả hai RDD để muốn có nó trong kết quả của mình. Ví dụ, nếu chúng ta đang nối thông tin khách hàng với các khuyến nghị, chúng ta có thể không muốn loại bỏ khách hàng nếu chưa có bất kỳ khuyến nghị nào. leftOuterJoin(other) và rightOuterJoin(other) đều nối các RDD cặp với nhau theo khóa, trong đó một trong hai cặp RDD có thể bị thiếu khóa.

Ví dụ 10: Kết nối trái/phải trong Python

from pyspark import SparkContext

sc = SparkContext("local", "Left Outer Join Example")

rdd1 = sc.parallelize([(1, "apple"), (2, "banana"), (3, "orange")])

rdd2 = sc.parallelize([(1, 1.0), (2, 1.5), (4, 2.0)])

# Left Outer Join

left\_outer\_join\_result = rdd1.leftOuterJoin(rdd2)

print("Left Outer Join:", left\_outer\_join\_result.collect())

Kết quả: [(1, ('apple', 1.0)), (2, ('banana', 1.5)), (3, ('orange', None))]

Ví dụ 11: Kết nối phải trong Python

from pyspark import SparkContext

sc = SparkContext("local", "Right Outer Join Example")

rdd1 = sc.parallelize([(1, "apple"), (2, "banana"), (3, "orange")])

rdd2 = sc.parallelize([(1, 1.0), (2, 1.5), (4, 2.0)])

# Right Outer Join

right\_outer\_join\_result = rdd1.rightOuterJoin(rdd2)

print("Right Outer Join:", right\_outer\_join\_result.collect())

Kết quả: [(1, ('apple', 1.0)), (2, ('banana', 1.5)), (4, (None, 2.0))]

### Một số câu lệnh khác

Giả sử rdd = [(1, 2), (3, 4), (3, 6)], other = [(3, 9)]

|  |  |  |
| --- | --- | --- |
| **Tên hàm** | **Mục đích** | **Ví dụ** |
| mapValues() | Tạo ra một RDD mới có khóa giữ nguyên và giá trị được thay đổi theo hàm được cung cấp | doubled\_values = rdd.mapValues(lambda x: x \* 2)  Kết quả: [(1, 4), (3, 8), (3, 12)] |
| flatMapValues() | Áp dụng một hàm mà trả về một vòng lặp cho mỗi giá trị của một RDD cặp, và cho mỗi phần tử được trả về, tạo ra một khóa/giá trị nhập với khóa cũ. | result = rdd.flatMapValues(lambda x: list(range(1, x + 1)))  Kết quả: [(1, 1), (1, 2), (3, 1), (3, 2), (3, 3), (3, 4), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6)] |
| keys() | Trả về một RDD chứa các khóa | keys = rdd.keys()  Kết quả: [1, 3, 3] |
| values() | Trả về một RDD chứa các giá trị | values = rdd.values()  Kết quả: [2, 4, 6] |
| substractByKey() | Xóa các phần tử có khóa trong RDD khác | result = rdd.substractByKey(other)  Kết quả: [(1, 2)] |

# Tài liệu tham khảo

|  |  |
| --- | --- |
| [1] | B. W. J. S. D. T. D. Denny Lee, “Learning Spark : lightning-fast data analytics,” trong *Learning Spark : lightning-fast data analytics*, O'Reilly Media, 2020, pp. 47-60. |