|  |  |  |
| --- | --- | --- |
| 指令 | 說明 | 舉例 |
| sc.parallelize() | 建立RDD  可以使用[index]尋找指定對象  填入的對象是list，內容可以是string或者數值  若放入的元素是多個pair，則第一項是Key，第二項是Value | intRDD = sc.parallelize([(1,2),(3,4),(5,6)])  intRDD.filter(lambda x: x[0]<5).collect()  # [(1,2),(3,4),(5,6)] |
| .keys() | 列出全部的key值 |  |
| .values() | 列出全部的value值 |  |
| .sortByKey(ascending = True) | 將pair依照key由小排到大 |  |
| .reduceByKey(func) | 將所有key值相同的value進行func reduce |  |
| .mapValues(func) | 針對pairs的Value進行func的運算 | A.mapValues(lambda x : x\*x).collect() |
| .collect() | 把RDD轉換成list顯示 | intRDD.collect() |
| def func(x):  return function of x | 建立函數 | def addOne(x):  return (x+1) |
| .map() | 將對象依輸入函數轉換對象 | intRDD.map(addOne).collect()  # map透過 addOne將intRDD中的元素各加1，最後將結果轉換為List |
| lambda x : function of x | 語法匿名函數 | lambda x : x+1 |
| .filter() | 利用填入的函數過濾 | intRDD.filter(lambda x : x>4).collect()  # 過濾掉intRDD中x<=4的元素 |
| .distinct() | 去除重複元素 |  |
| .randomSplit() | 將對象依想要的比例隨機切割 | sRDD = intRDD.randomSplit([0.4,0.6])  sRDD[0].collect()  # 按照4比6將intRDD拆成隨機的兩個RDD |
| .groupBy() | 將整個集合依據函數分拆 | gRDD=intRDD.groupBy(lambda x:”even” if (x % 2 == 0) else “odd”).collect() |
| .union() | 聯集 | A.union(B) |
| .intersection() | 交集 | A.intersection(B) |
| .subtract | 差集 | A.subtract(B) |
| .cartesian() | 笛卡爾乘積 | A.cartesian(B)  # A x B |
| .first() | 取出第一筆 | A.first() |
| .take() | 取出指定index的資料 | A.take(2)  # 取出A的二個元素 |
| .takeOrdered(index, key = func) | 由小到大排序後，依序取出前i筆 | A.takeOrdered(i, lambda x: x)  # 依序取出前i筆 |
| .stats() | 針對一個List做出數量、平均、標準差、最大與最小值的計算 | A.stats() |
| .min() | 最小值 |  |
| .max() | 最大值 |  |
| .stdev() | 標準差 |  |
| .count() | 計算RDD中元素的數量 |  |
| .sum() | 加總 |  |
| .mean() | 平均 |  |
| .join() | 一個一個將A和B中擁有相同key值的value收集起來變成{value(A), value(B)}，建立一個新的RDD叫做{(key, (value A, value B)} | A.join(B)  # A與B作join |
| .leftOuterJoin()  .rightOuterJoin() | 顯示那些key出現在左/右邊的原集合之key的經過join之後的RDD  如果出現在左/右邊的key在另一邊的key中未出現，則join的value值為None  如果另一邊出現那些不在左/右邊的元素，則忽略 |  |
| .subtractByKey() | 移除有相同key的資料 |  |
| .countByKey() | 計算每一個key值的筆數  傳回defaultdict(int, {key: num}) |  |
| .collectAsMap() | 建立Key-Value的dict |  |
| .broadcast() | 廣播變數  使用.value來讀取廣播變數的值  可以節省很多記憶體 | A = sc.broadcast(B)  # 定義A是B的廣播，內容上都不變，但在執行的過程廣播可以減少記憶體的使用 |
| .accumulator() | .accumulator(0.0)→Double型態  .accumulator(0)→Int型態  .add()進行累加  .value取得accumulator的值 |  |
| .foreach(func) | 遍歷所有對象集合的元素 | A.foreach(lambda i: [total.add(i), num.add(1)])  # 對A中的元素進行加總 |
| .persist() | 將需要重複運算的RDD儲存在記憶體裡  利用.is\_cached確認是不是持久化了 | A.persist() |
| Spark WordCount |  | textFile=sc.textFile("data/test.txt")  #讀取本地檔案  stringRDD=textFile.flatMap(lambda line: line.split(" "))  #取出每個文字建立去階層化(flatMap)的RDD  countsRDD=stringRDD.map(lambda word: (word, 1)).reduceByKey(lambda x,y:x+y)  #map將RDD轉換成Key-Value，reduceByKey根據key值做value的總和  countsRDD.saveAsTextFile("data/output") |
| sc.textFile() | 讀取本地文件 | sc.textFile(“data/test.txt”) |
|  |  |  |
|  |  |  |
|  |  |  |