1 ，MapPartitions 是对每个分区进行操作，分区规则采用无序

**object** test {

**def** main(args: Array[*String*]): Unit = {

System.setProperty("hadoop.home.dir", "F:\\spark-1.4.0-bin-hadoop2.6");

**val** conf = **new** SparkConf().setAppName("spark rdd test").setMaster("local")

**val** spark = **new** SparkContext(conf)

**val** input = spark.parallelize(List(1,2,3,4,5,6,7), 3)

**val** result = input.mapPartitions(partition => Iterator(sumOfEveryPartition(partition)))

result.collect.foreach { println }

}

**def** sumOfEveryPartition(input:*Iterator*[Int]):Int={

**var** total = 0

input.foreach { x => total =total + x }

total

}

}

|  |
| --- |
| Result：  3  7  18 |

2 ， mapValue适用于带有kv键值对的rdd，对应的函数操作用于value

3，flatMapValues 每个一元素的Value被输入函数映射为一系列的值，然后这些值再与原RDD中的Key组成一系列新的KV对

**object** test {

**def** main(args: Array[*String*]): Unit = {

System.setProperty("hadoop.home.dir", "F:\\spark-1.4.0-bin-hadoop2.6");

**val** conf = **new** SparkConf().setAppName("spark rdd test").setMaster("local")

**val** spark = **new** SparkContext(conf)

**val** a = spark.parallelize(List((7,2),(3,4),(5,6)))

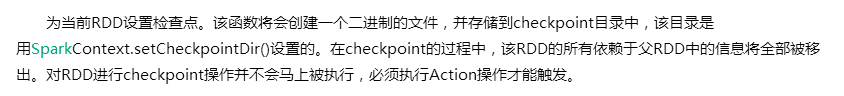
**val** b = a.flatMapValues(x=>1 to x)

b.collect.foreach(println(\_))

}

|  |
| --- |
| Result：  (7,1)  (7,2)  (3,1)  (3,2)  (3,3)  (3,4)  (5,1)  (5,2)  (5,3)  (5,4)  (5,5)  (5,6) |

4，checkpoint



5,coalesce

对rdd里面的分区进行合并；

使用与在filter后，每个分区的数据进行了过滤而缺少，重新分区，提高性能。