> 因为事物总是向着熵增的方向发展，所以一切符合熵增的，都非常容易和舒适，比如懒散。

RDD行动算子

reduce算子

作用：聚合RDD中的所有元素。

需求：创建一个键值对RDD，并进行聚合。

(1) 创建第一个RDD

```
scala> val rdd = sc.makeRDD(Array(("a",1),("a",2),("b",3),("b",4)))
rdd: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[8] at
makeRDD at <console>:24
```

(2) 使用reduce算子聚合元素

```
scala> rdd.reduce((x,y) => (x._1+y._1,x._2+y._2))
res4: (String, Int) = (aabb,10)
```

collect算子

作用：以数组的形式返回数据集中的所有数据。

需求：创建一个数值型RDD然后打印。

(1) 创建RDD

```
scala> val rdd = sc.parallelize(0 to 9)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at
<console>:24
```

(2) 打印结果

```
scala> rdd.collect
res5: Array[Int] = Array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

count算子

作用：返回RDD中的元素个数

需求：创建一个数值型RDD然后统计其个数

(1) 创建RDD

```
scala> val rdd = sc.parallelize(0 to 9)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at
<console>:24
```

(2) 统计个数

```
scala> rdd.count
res6: Long = 10
```

first算子

作用：返回RDD中的第一个元素

需求：创建一个数值型RDD然后返回第一个元素

(1) 创建RDD

```
scala> val rdd = sc.parallelize(0 to 9)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at
<console>:24
```

(2) 返回第一个元素

```
scala> rdd.first
res7: Int = 0
```

take算子

作用：返回RDD中的前n个元素组成的数组

需求：创建一个数值型RDD返回前n个元素

(1) 创建RDD

```
scala> val rdd = sc.parallelize(0 to 9)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at
<console>:24
```

(2) 返回前n个元素

```
scala> rdd.take(5)
res8: Array[Int] = Array(0, 1, 2, 3, 4)
```

takeOrdered算子

作用：在上一个算子返回的结果上进行排序

需求：创建一个数值型RDD，返回排序后的n个元素

(1) 创建RDD

```
scala>  val rdd = sc.parallelize(Array(2,1,3,5,4))
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[13] at parallelize at
<console>:24
```

(2) 返回排序后的n个元素

```
scala> rdd.takeOrdered(3)
res11: Array[Int] = Array(1, 2, 3)
```

aggregate算子

参数：(zeroValue: U)(seqOp: (U, T) ⇒ U, combOp: (U, U) ⇒ U)

作用：aggregate函数将每个分区里面的元素通过seqOp和初始值进行聚合，然后用combine函数将每个分区的结果和初始值(zeroValue)进行combine操作。这个函数最终返回的类型不需要和RDD中元素类型一致。

需求：创建一个数值型RDD，将所有元素相加得到结果

(1) 创建RDD

```
scala> val rdd = sc.parallelize(0 to 9)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at
<console>:24
```

(2) 统计个数

```
scala> rdd.aggregate(0)(_+_,_+_)
res12: Int = 45
```

fold算子

作用：折叠操作，aggregate的简化操作，seqop和combop是一样的操作。

需求：创建一个数值型RDD，将所有元素相加得到结果

(1) 创建RDD

```
scala> val rdd = sc.parallelize(0 to 9)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at
<console>:24
```

(2) 统计个数

```
scala> rdd.fold(0)(_+_)
res13: Int = 45
```

countByKey算子

作用：对(K,V)类型的RDD，返回一个(K,Int)的map，表示每一个key对应的元素个数。

需求：创建一个数值型RDD，统计每种key的个数

(1) 创建RDD

```
scala> val rdd = sc.parallelize(List(("a",1),("a",2),("b",3),("c",4),("c",5),
("c",6)),3)
rdd: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[17] at
parallelize at <console>:24
```

(2) 统计每种key的个数

```
scala> rdd.countByKey
res14: scala.collection.Map[String,Long] = Map(c -> 3, a -> 2, b -> 1)
```

foreach算子

作用：遍历所有元素，执行函数进行输出。

需求：创建一个数值型RDD，将所有元素进行输出

(1) 创建RDD

```
scala> val rdd = sc.parallelize(0 to 9)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at
<console>:24
```

(2) 输出RDD的值

```
scala> rdd.foreach(print(_))
0123456789
```