



# Spark MLlib机器学习 第1周

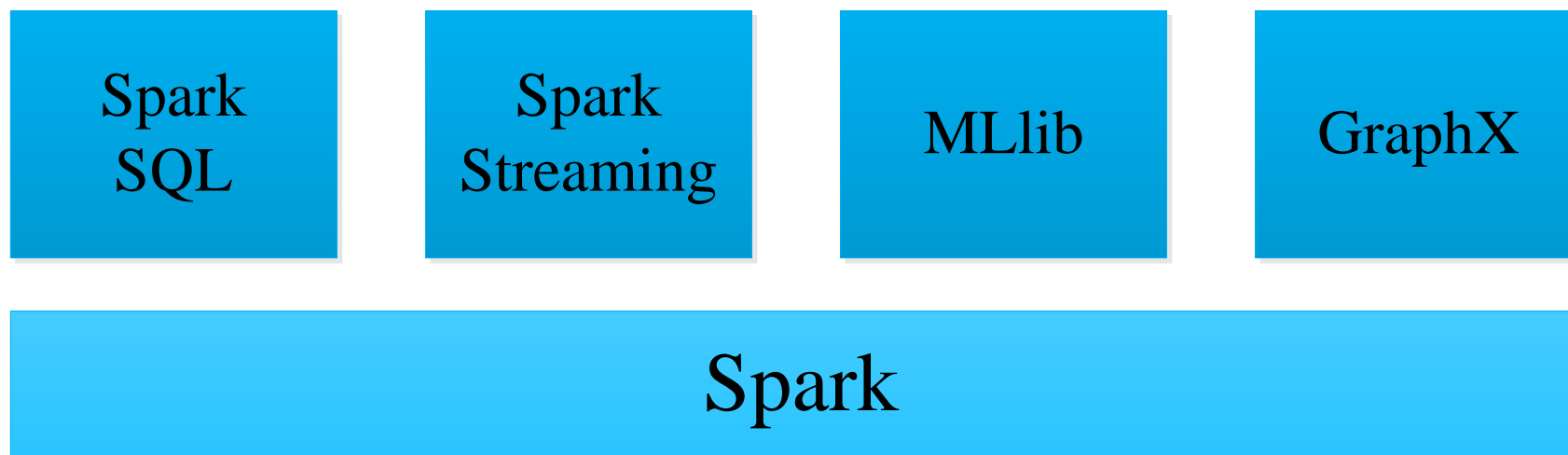
DATAGURU专业数据分析社区

- 黄美灵，网名：sunbow，Spark爱好者，现从事移动互联网的计算广告和数据变现工作。
- 《Spark MLlib机器学习：算法、源码及实战详解》作者
- CSDN博客专家
- <http://blog.csdn.net/sunbow0>

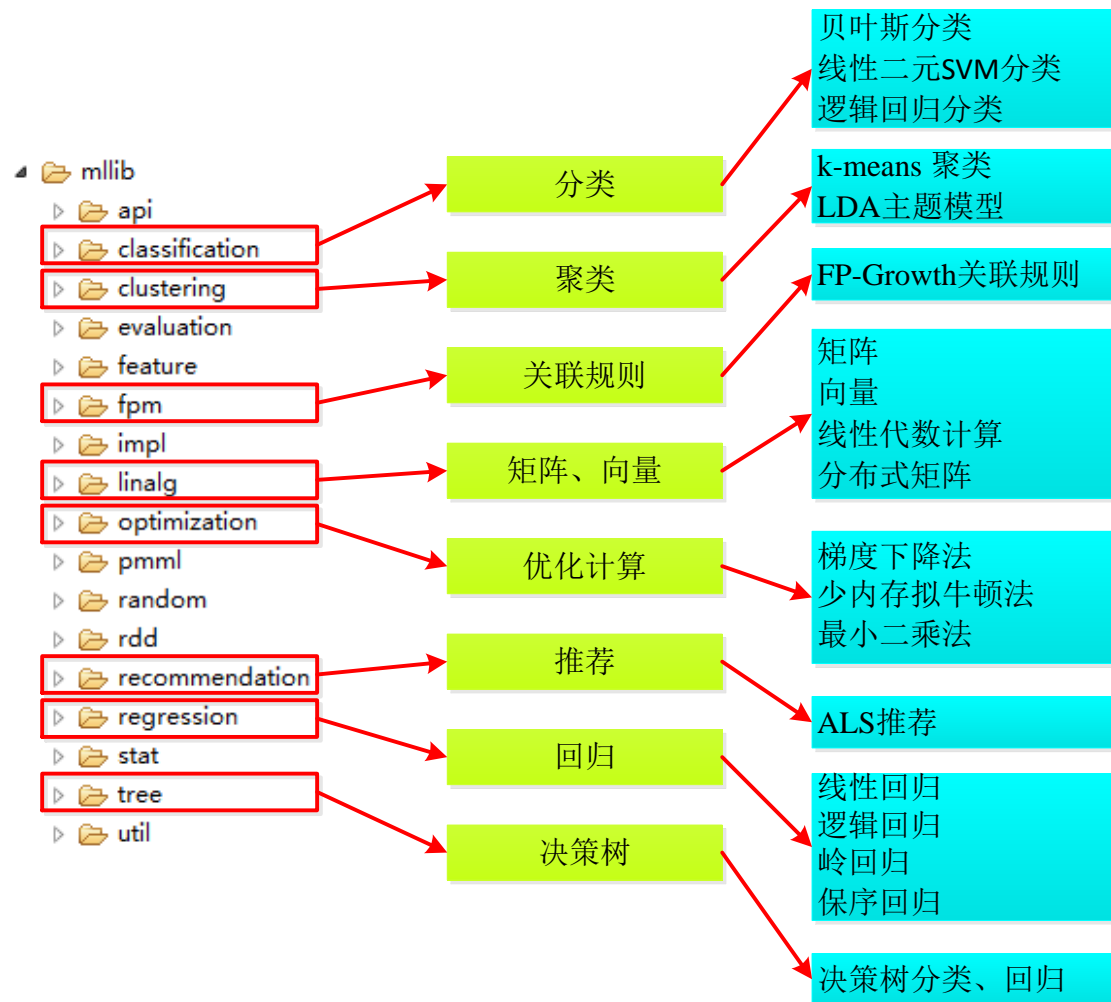


# 第一课 Spark MLlib基础入门


- 第一课 Spark MLlib基础入门
- 1、Spark介绍
- 2、Spark MLlib介绍
- 3、课程的基础环境
- 4、Spark RDD操作



# Spark Mlib 介绍



- Spark1.5.1、Spark1.4.1
- `sudo spark-shell --executor-memory 2g --driver-memory 1g --total-executor-cores 2 --num-executors 1 --master spark://192.168.180.156:7077`

 **Spark Master at spark://192.168.180.156:7077**

URL: spark://192.168.180.156:7077  
REST URL: spark://192.168.180.156:6066 (cluster mode)  
Alive Workers: 1  
Cores in use: 2 Total, 2 Used  
Memory in use: 2.0 GB Total, 2.0 GB Used  
Applications: 1 Running, 8 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

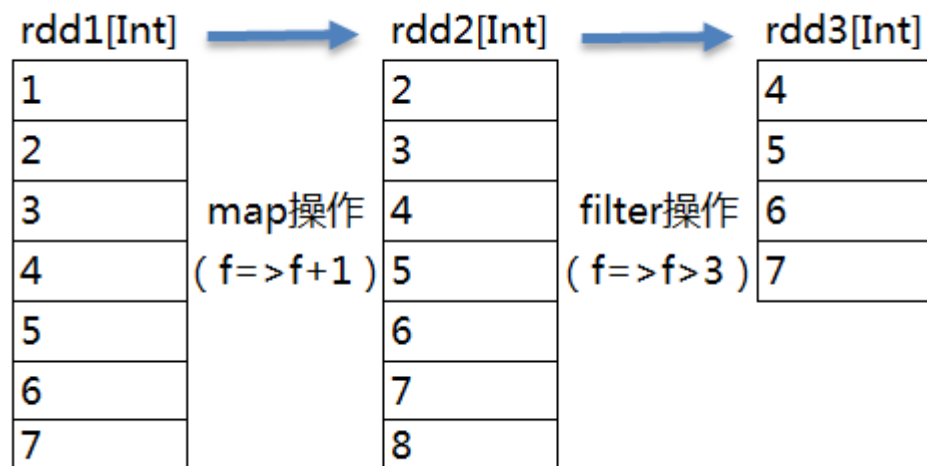
## Workers

Worker Id	Address	State	Cores	Memory
<a href="#">worker-20160412143006-192.168.180.156-42236</a>	192.168.180.156:42236	ALIVE	2 (2 Used)	2.0 GB (2.0 GB Used)

## Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
<a href="#">app-20160506184703-0008</a> (kill)	Spark shell	2	2.0 GB	2016/05/06 18:47:03	root	RUNNING	1.2 min

- RDD是什么？Resilient Distributed Datasets ( RDD , ) 弹性分布式数据集
- 存储在硬盘或者内存上
- 分区
- 每一行（每个元素）
- 父子依赖（血缘）关系



## 1、RDD分区文件

part001
part002
part003

## 2、每个分区文件的元素

part001	part002	part003
a,1,1.0	c,1,1.0	f,1,1.0
a,2,1.0	c,2,1.0	f,2,1.0
a,3,1.0	d,3,1.0	g,3,1.0
a,4,1.0	d,4,1.0	g,4,1.0
b,4,1.0	e,3,1.0	h,3,1.0
b,4,1.0	e,4,1.0	h,4,1.0

## ■ 1) 数据集

```
val data = Array(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
val distData = sc.parallelize(data, 3)
```

```
scala> val data = Array(1, 2, 3, 4, 5, 6, 7, 8, 9)
data: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9)

scala> val distData = sc.parallelize(data, 3)
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:23

scala> distData.collect
res0: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9)

scala> distData.take(1)
res1: Array[Int] = Array(1)
```



## ■ 2) 外部数据源

`val distFile1 = sc.textFile("data.txt")` //本地当前目录下文件

`val distFile2 = sc.textFile("hdfs://192.168.1.100:9000/input/data.txt")` //HDFS文件

`val distFile3 = sc.textFile("file:/input/data.txt")` //本地指定目录下文件

`val distFile4 = sc.textFile("/input/data.txt")` //本地指定目录下文件

注意：textFile可以读取多个文件，或者1个文件夹，也支持压缩文件、包含通配符的路径。

`textFile("/input/001.txt, /input/002.txt ")` //读取多个文件

`textFile("/input")` //读取目录

`textFile("/input/*.txt")` //含通配符的路径

`textFile("/input/*.gz")` //读取压缩文件

```
scala> val distFile2 = sc.textFile("hdfs://192.168.180.79:9000/user/huan  
distFile2: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1] at te  
  
scala> distFile2.take(1)  
res0: Array[String] = Array(-0.4307829,-1.63735562648104 -2.00621178480  
728919298 -0.864466507337306)
```

## ■ 1 ) map

map是对RDD中的每个元素都执行一个指定的函数来产生一个新的RDD；RDD之间的元素是一对一关系；

```
val rdd1 = sc.parallelize(1 to 9, 3)
```

```
val rdd2 = rdd1.map(x => x*2)
```

```
rdd2.collect
```

```
res3: Array[Int] = Array(2, 4, 6, 8, 10, 12, 14, 16, 18)
```

## ■ 2 ) filter

Filter是对RDD元素进行过滤；返回一个新的数据集，是经过func函数后返回值为true的原元素组成；

```
val rdd3 = rdd2.filter(x => x > 10)
```

```
rdd3.collect
```

```
res4: Array[Int] = Array(12, 14, 16, 18)
```

## ■ 3 ) flatMap

flatMap类似于map，但是每一个输入元素，会被映射为0到多个输出元素（因此，func函数的返回值是一个Seq，而不是单一元素），RDD之间的元素是一对多关系；

```
val rdd4 = rdd3. flatMap (x => x to 20)
```

```
res5: Array[Int] = Array(12, 13, 14, 15, 16, 17, 18, 19, 20, 14, 15, 16, 17, 18, 19, 20, 16, 17, 18, 19, 20, 18, 19, 20)
```

## ■ 4 ) mapPartitions

mapPartitions是map的一个变种。map的输入函数是应用于RDD中每个元素，而mapPartitions的输入函数是每个分区的数据，也就是把每个分区中的内容作为整体来处理的。

## ■ 5 ) mapPartitionsWithIndex

mapPartitionsWithSplit与mapPartitions的功能类似，只是多传入split index而已，所有func 函数必需是 (Int, Iterator<T>) => Iterator<U> 类型。

## ■ 6 ) sample

sample(withReplacement,fraction,seed)是根据给定的随机种子seed，随机抽样出数量为frac的数据。withReplacement：是否放回抽样；fraction：比例，0.1表示10%；

```
val a = sc.parallelize(1 to 10000, 3)
```

```
a.sample(false, 0.1, 0).count
```

```
res24: Long = 960
```

## ■ 7 ) union

union(otherDataset)是数据合并，返回一个新的数据集，由原数据集和otherDataset联合而成。

```
val rdd8 = rdd1.union(rdd3)
```

```
rdd8.collect
```

```
res14: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 14, 16, 18)
```

## ■ 8 ) intersection

intersection(otherDataset)是数据交集，返回一个新的数据集，包含两个数据集的交集数据；

```
val rdd9 = rdd8.intersection(rdd1)
```

```
rdd9.collect
```

```
res16: Array[Int] = Array(6, 1, 7, 8, 2, 3, 9, 4, 5)
```

## ■ 9 ) distinct

distinct([numTasks]))是数据去重，返回一个数据集，是对两个数据集去除重复数据，numTasks参数是设置任务并行数量。

```
val rdd10 = rdd8.union(rdd9).distinct
```

```
rdd10.collect
```

```
res19: Array[Int] = Array(12, 1, 14, 2, 3, 4, 16, 5, 6, 18, 7, 8, 9)
```

## ■ 10 ) groupByKey

groupByKey([numTasks])是数据分组操作，在一个由 ( K,V ) 对组成的数据集上调用，返回一个 ( K,Seq[V])对的数据集。

```
val rdd0 = sc.parallelize(Array((1,1), (1,2), (1,3), (2,1), (2,2), (2,3)), 3)
```

```
val rdd11 = rdd0.groupByKey()
```

```
rdd11.collect
```

```
res33: Array[(Int, Iterable[Int])] = Array((1,ArrayBuffer(1, 2, 3)), (2,ArrayBuffer(1, 2, 3)))
```

## ■ 11 ) reduceByKey

reduceByKey(func, [numTasks])是数据分组聚合操作，在一个 ( K,V)对的数据集上使用，返回一个 ( K,V ) 对的数据集，key相同的值，都被使用指定的reduce函数聚合到一起。

```
val rdd12 = rdd0.reduceByKey((x,y) => x + y)
```

```
rdd12.collect
```

```
res34: Array[(Int, Int)] = Array((1,6), (2,6))
```

## ■ 12 ) aggregateByKey

aggregateByKey(zeroValue: U)(seqOp: (U, T) => U, combOp: (U, U) => U) 和 reduceByKey 的不同在于，reduceByKey 输入输出都是 (K, V)，而 aggregateByKey 输出是 (K, U)，可以不同于输入 (K, V)，aggregateByKey 的三个参数：

zeroValue: U，初始值，比如空列表 {}；

seqOp: (U, T) => U，seq 操作符，描述如何将 T 合并入 U，比如如何将 item 合并到列表；

combOp: (U, U) => U，comb 操作符，描述如果合并两个 U，比如合并两个列表；

所以 aggregateByKey 可以看成更高抽象的，更灵活的 reduce 或 group。

```
val z = sc.parallelize(List(1,2,3,4,5,6), 2)
```

```
z.aggregate(0)(math.max(_,_), _ + _)
```

```
res40: Int = 9
```

```
val z = sc.parallelize(List((1, 3), (1, 2), (1, 4), (2, 3)))
```

```
z.aggregateByKey(0)(math.max(_,_), _ + _)
```

```
res2: Array[(Int, Int)] = Array((2,3), (1,9))
```

## ■ 13 ) combineByKey

combineByKey是对RDD中的数据集按照Key进行聚合操作。聚合操作的逻辑是通过自定义函数提供给combineByKey。

combineByKey[C](createCombiner: (V)  $\Rightarrow$  C, mergeValue: (C, V)  $\Rightarrow$  C, mergeCombiners: (C, C)

$\Rightarrow$  C, numPartitions: Int):RDD[(K, C)]把(K,V) 类型的RDD转换为(K,C)类型的RDD , C和V可以不一样。combineByKey三个参数 :

```
val data = Array((1, 1.0), (1, 2.0), (1, 3.0), (2, 4.0), (2, 5.0), (2, 6.0))
```

```
val rdd = sc.parallelize(data, 2)
```

```
val combine1 = rdd.combineByKey(createCombiner = (v:Double) => (v:Double, 1),
```

```
mergeValue = (c:(Double, Int), v:Double) => (c._1 + v, c._2 + 1),
```

```
mergeCombiners = (c1:(Double, Int), c2:(Double, Int)) => (c1._1 + c2._1, c1._2 + c2._2),
```

```
numPartitions = 2 )
```

```
combine1.collect
```

```
res0: Array[(Int, (Double, Int))] = Array((2,(15.0,3)), (1,(6.0,3)))
```



## ■ 14 ) sortByKey

sortByKey([ascending],[numTasks])是排序操作，对(K,V)类型的数据按照K进行排序，其中K需要实现Ordered方法。

```
val rdd14 = rdd0.sortByKey()
```

```
rdd14.collect
```

```
res36: Array[(Int, Int)] = Array((1,1), (1,2), (1,3), (2,1), (2,2), (2,3))
```

## ■ 15 ) join

join(otherDataset, [numTasks])是连接操作，将输入数据集(K,V)和另外一个数据集(K,W)进行Join，得到(K, (V,W))；该操作是对于相同K的V和W集合进行笛卡尔积操作，也即V和W的所有组合；

```
val rdd15 = rdd0.join(rdd0)
```

```
rdd15.collect
```

```
res37: Array[(Int, (Int, Int))] = Array((1,(1,1)), (1,(1,2)), (1,(1,3)), (1,(2,1)), (1,(2,2)), (1,(2,3)), (1,(3,1)), (1,(3,2)), (1,(3,3)), (2,(1,1)),  
(2,(1,2)), (2,(1,3)), (2,(2,1)), (2,(2,2)), (2,(2,3)), (2,(3,1)), (2,(3,2)), (2,(3,3)))
```

连接操作除join外，还有左连接、右连接、全连接操作函数：leftOuterJoin、rightOuterJoin、fullOuterJoin。

## ■ 16 ) cogroup

cogroup(otherDataset, [numTasks])是将输入数据集(K, V)和另外一个数据集(K, W)进行cogroup , 得到一个格式为(K, Seq[V], Seq[W])的数据集。

```
val rdd16 = rdd0.cogroup(rdd0)
```

```
rdd16.collect
```

```
res38: Array[(Int, (Iterable[Int], Iterable[Int]))] = Array((1,(ArrayBuffer(1, 2, 3),ArrayBuffer(1, 2, 3))), (2,(ArrayBuffer(1, 2, 3),ArrayBuffer(1, 2, 3))))
```

## ■ 17 ) cartesian

cartesian(otherDataset)是做笛卡尔积：对于数据集T和U 进行笛卡尔积操作 , 得到(T, U)格式的数据集。

```
val rdd17 = rdd1.cartesian(rdd3)
```

```
rdd17.collect
```

```
res39: Array[(Int, Int)] = Array((1,12), (2,12), (3,12), (1,14), (1,16), (1,18), (2,14), (2,16), (2,18), (3,14), (3,16), (3,18), (4,12), (5,12), (6,12), (4,14), (4,16), (4,18), (5,14), (5,16), (5,18), (6,14), (6,16), (6,18), (7,12), (8,12), (9,12), (7,14), (7,16), (7,18), (8,14), (8,16), (8,18), (9,14), (9,16), (9,18))
```

## ■ 1 ) reduce

reduce(func)是对数据集的所有元素执行聚集(func)函数，该函数必须是可交换的。

```
val rdd1 = sc.parallelize(1 to 9, 3)
```

```
val rdd2 = rdd1.reduce(_ + _)
```

```
rdd2: Int = 45
```

## ■ 2 ) collect

collect是将数据集中的所有元素以一个array的形式返回。

```
rdd1.collect()
```

```
res8: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

## ■ 3 ) count

返回数据集中元素的个数。

```
rdd1.count()
```

```
res9: Long = 9
```

## ■ 4 ) first

返回数据集中的第一个元素，类似于take(1)。

```
rdd1.first()
```

```
res10: Int = 1
```

## ■ 5 ) take

Take(n)返回一个包含数据集中前n个元素的数组，当前该操作不能并行。

```
rdd1.take(3)
```

```
res11: Array[Int] = Array(1, 2, 3)
```

## ■ 6 ) takeSample

takeSample(withReplacement,num, [seed])返回包含随机的num个元素的数组，和Sample不同，takeSample 是行动操作，所以返回的是数组而不是RDD，其中第一个参数withReplacement是抽样时是否放回，第二个参数num会精确指定抽样数，而不是比例。

```
rdd1.takeSample(true, 4)
```

```
res15: Array[Int] = Array(9, 5, 5, 6)
```

## ■ 7 ) takeOrdered

takeOrdered(n , [ordering])是返回包含随机的n个元素的数组，按照顺序输出。

```
rdd1.takeOrdered(4)
```

```
res16: Array[Int] = Array(1, 2, 3, 4)
```

## ■ 8 ) saveAsTextFile

把数据集中的元素写到一个文本文件，Spark会对每个元素调用toString方法来把每个元素存成文本文件的一行。

## ■ 9 ) countByKey

对于(K, V)类型的RDD. 返回一个(K, Int)的map，Int为K的个数。

## ■ 10 ) foreach

foreach(func)是对数据集中的每个元素都执行func函数。

- Spark API
- <http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.package>
- Scala API
- <http://www.scala-lang.org/api/2.10.4/#scala.Any>

**【声明】** 本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

<http://edu.dataguru.cn>

- Dataguru ( 炼数成金 ) 是专业数据分析网站 , 提供教育 , 媒体 , 内容 , 社区 , 出版 , 数据分析业务等服务。我们的课程采用新兴的互联网教育形式 , 独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围 , 重竞争压力的特点 , 同时又发挥互联网的威力打破时空限制 , 把天南地北志同道合的朋友组织在一起交流学习 , 使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本 , 直线下降至百元范围 , 造福大众。我们的目标是 : 低成本传播高价值知识 , 构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情 , 请看我们的培训网站 <http://edu.dataguru.cn>



# Thanks

**FAQ时间**