> 再次强调，不要去记忆你不理解的内容

RDD累加器和广播变量

累加器

> 累加器用来对**信息进行聚合**，通常在向 Spark传递函数时，比如使用 map() 函数或者用 filter() 传条件时，可以使用驱动器程序中定义的变量，但是集群中运行的每个任务都**会得到这些变量的一份新的副本**，**更新这些副本的值也不会影响驱动器中的对应变量**(问题所在)。如果我们想实现所有分片处理时更新共享变量的功能，那么累加器可以实现我们想要的效果。

系统累加器

需求：针对一个输入的日志文件，我们需要计算文件中所有空行的数量，我们可以编写以下程序

1）读取文件

```
scala> val notice = sc.textFile("./NOTICE")
notice: org.apache.spark.rdd.RDD[String] = ./NOTICE MapPartitionsRDD[40] at
textFile at <console>:32
```

2）注册累加器

```
scala> val blanklines = sc.accumulator(0)
warning: there were two deprecation warnings; re-run with -deprecation for
details
blanklines: org.apache.spark.Accumulator[Int] = 0
```

3）使用累加器，遇到空行则自增1

```
scala> val tmp = notice.flatMap(line => {
   |   if (line == "") {
   |    blanklines += 1
   |   }
   |   line.split(" ")
   | })
tmp: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[41] at flatMap at
<console>:36
```

4）打印RDD中的数量

```
scala> tmp.count()
res31: Long = 3213
```

5）打印累加器的值,即文件中的空行数

```
scala> blanklines.value
res32: Int = 171
```

通过在驱动器中调用SparkContext.accumulator(initialValue)方法，创建出存有初始值的累加器。返回值为 org.apache.spark.Accumulator[T] 对象，其中 T 是初始值 initialValue 的类型。Spark闭包里的执行器代码可以使用累加器的 += 方法(在Java中是 add)增加累加器的值。 驱动器程序可以调用累加器的value属性(在Java中使用value()或setValue())来访问累加器的值。

注意：

（1）工作节点上的任务不能访问累加器的值。从这些任务的角度来看，**累加器是一个只写变量。**

【这说明不能再算子中打印累加器值！！！】

（2）对于要在行动操作中使用的累加器，Spark只会把每个任务对各累加器的修改应用一次。因此，如果想要一个无论在失败还是重复计算时都绝对可靠的累加器，我们必须把它放在 foreach() 这样的行动操作中。转化操作中累加器可能会发生不止一次更新。

自定义累加器

> 自定义累加器类型的功能在1.X版本中就已经提供了，但是使用起来比较麻烦，在2.0版本后，累加器的易用性有了较大的改进，而且官方还提供了一个新的抽象类：AccumulatorV2来提供更加友好的自定义类型累加器的实现方式。实现自定义类型累加器需要继承AccumulatorV2并覆写要求的方法。

需求：实现一个自定义的数值累加器

代码实现

```
import org.apache.spark.util.AccumulatorV2

class MyAccu extends AccumulatorV2[Int, Int] {

 var sum = 0

 //判断是否为空
 override def isZero: Boolean = sum == 0

 //复制
 override def copy(): AccumulatorV2[Int, Int] = {
  val accu = new MyAccu
  accu.sum = this.sum
  accu

 }

 //重置
 override def reset(): Unit = sum = 0

 //累加
 override def add(v: Int): Unit = sum += v

 //合并
 override def merge(other: AccumulatorV2[Int, Int]): Unit = sum += other.value

 //返回值
 override def value: Int = sum
}
```

3） 调用自定义累加器

```
import org.apache.spark.rdd.RDD
import org.apache.spark.{Accumulator, SparkConf, SparkContext}

object AccuTest {
 def main(args: Array[String]): Unit = {
  //创建SparkConf
  val sparkConf: SparkConf = new
SparkConf().setMaster("local[*]").setAppName("AccuTest")

  //创建SC
  val sc = new SparkContext(sparkConf)

  //创建自定义累加器对象
  val accu = new MyAccu

  //注册累加器!!!不要忘记
  sc.register(accu)

  //创建RDD
  val value: RDD[Int] = sc.parallelize(Array(1, 2, 3, 4))

  //在行动算子中对累加器的值进行修改
  value.foreach { x =>
   accu.add(1)
   println(x)

  }

  //打印累加器的值
  println(accu.value)

  //关闭SparkContext
  sc.stop()

 }
 }
```

> 可以理解为累加器是站在Driver和Executor上面的变量

广播变量

> 广播变量用来**高效分发较大的对象**。向所有工作节点发送**一个较大的只读值**，以供一个或多个Spark操作使用。比如，如果你的应用需要向所有节点发送一个**较大的只读查询表**，甚至是机器学习算法中的一个很大的特征向量，广播变量用起来都很顺手。 在多个并行操作中使用同一个变量，但是 Spark会为每个任务分别发送。

【其实就是创建一个大家都可以访问得到的大变量】

使用广播变量的过程如下：

（1）通过对一个类型T的对象调用SparkContext.broadcast创建出一个Broadcast[T]对象，任何可序列化的类型都可以这么实现。

（2）通过value属性访问该对象的值(在Java中为value()方法)。

（3）变量只会被发到各个节点一次，应作为只读值处理(修改这个值不会影响到别的节点)。

```
scala> val broadcastVar = sc.broadcast(Array(1, 2, 3))
broadcastVar: org.apache.spark.broadcast.Broadcast[Array[Int]] = Broadcast(35)

scala> broadcastVar.value
res33: Array[Int] = Array(1, 2, 3)
```