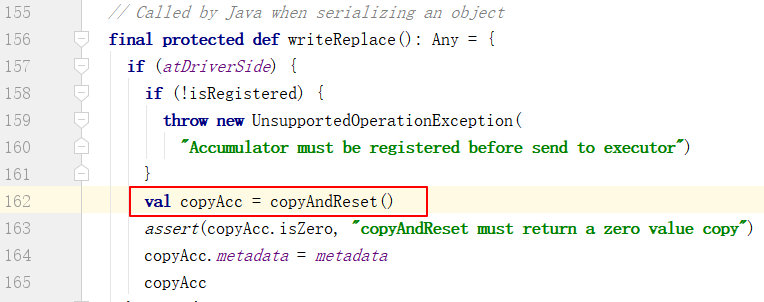
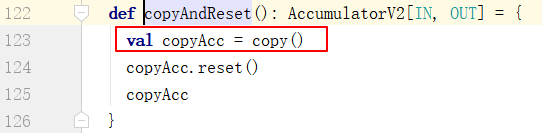
**调研人：李海波**

学生问题：Spark中累加器的copy方法到底调用了多少次，如何计算出来？

答：首先需要明确累加器的copy方法是在累加器对象进行序列化时进行的

**源码：AccumulatorV2.scala**



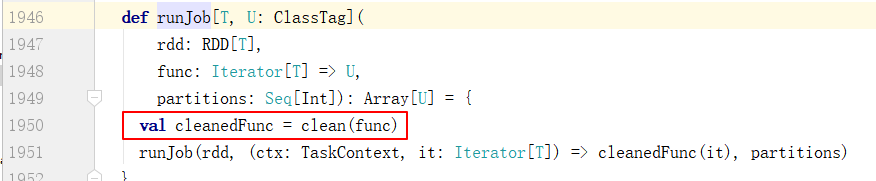


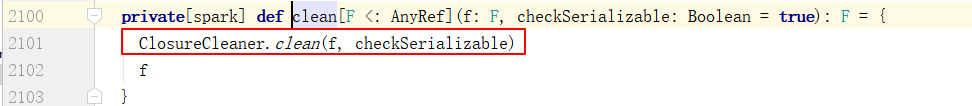
那么现在就需要确定在整个JOB执行过程中对累加器进行了多少次序列化。

接下来，我们看看在整个作业中，哪些地方可能会序列化：

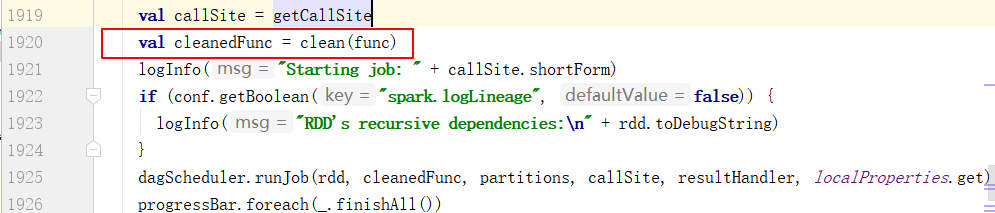
1. **在运行任务前进行闭包检测时**

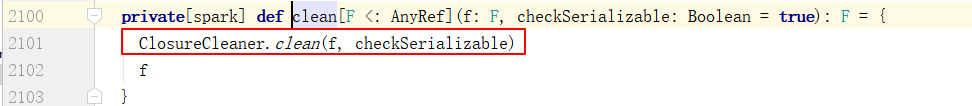
**源码：SparkContext.scala**





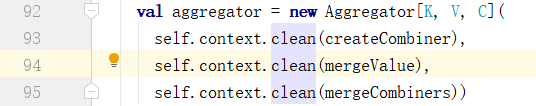
1. **在有向无环图调度器运行任务前进行闭包检测时**

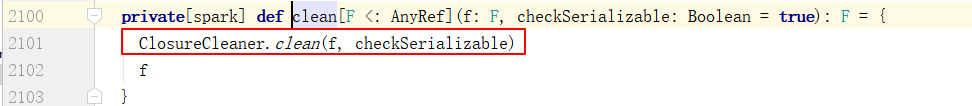




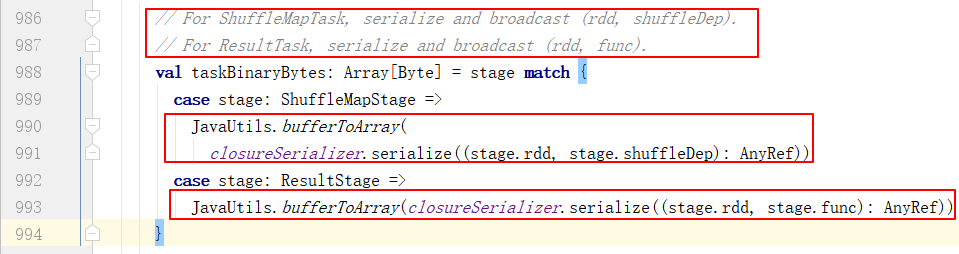
1. **在聚合算子进行处理时：**

**源码：PairRDDFunctions.scala**

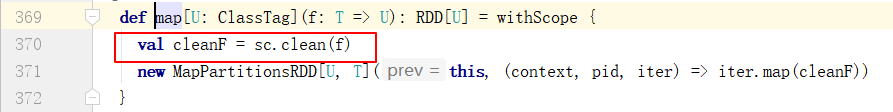


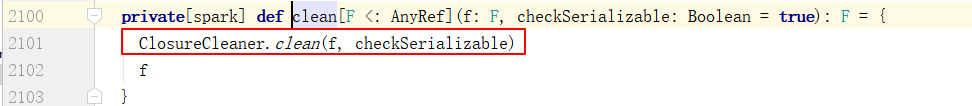


1. **在任务提交关联阶段时：**



1. **在算子进行闭包检测时：**





接下来我们就分不同场合看一看具体代码的执行情况：

1. **一个分区数据，0次转换,一次行动**

val rdd: RDD[Int] = sc.makeRDD(List(1,2),1)

val acc = new MyAcc

sc.register(acc, "wordcount")

rdd.foreach(

num => acc.add(num)

)

println("driver => " + acc.value)

**源码执行时，有4处涉及序列化操作：**

* **算子foreach执行时进行的闭包检测【RDD.scala(L917)】**
* **执行作业时进行的闭包检测【SparkContext.scala(L1950)】**
* **执行有向无环图调度器运行作业时的闭包检测【SparkContext.scala(L1920)】**
* **因为没有shuffle，所以只有一个阶段ResultStage，进行任务关联时【DAGScheduler.scala(L993)】**

1. **一个分区数据，一次转换，一次行动**

val rdd: RDD[Int] = sc.makeRDD(List(1,2),1)

val acc = new MyAcc

sc.register(acc, "wordcount")

rdd

.map(\_\*2)

.foreach(

num => acc.add(num)

)

println("driver => " + acc.value)

**源码执行时，有4处涉及序列化操作：**

* **算子foreach执行时进行的闭包检测【RDD.scala(L917)】**
* **执行作业时进行的闭包检测【SparkContext.scala(L1950)】**
* **执行有向无环图调度器运行作业时的闭包检测【SparkContext.scala(L1920)】**
* **因为没有shuffle，所以只有一个阶段ResultStage，进行任务关联时【DAGScheduler.scala(L993)】**

1. **一个分区数据，一次转换，一次聚合，一次行动**

val rdd: RDD[Int] = sc.makeRDD(List(1,2),1)

val acc = new MyAcc

sc.register(acc, "wordcount")

rdd

.map(("a",\_))

.reduceByKey(\_+\_)

.foreach{

case (s:String, num:Int) => {

acc.add(num)

}

}

println("driver => " + acc.value)

**源码执行时，有4处涉及序列化操作：**

* **算子foreach执行时进行的闭包检测【RDD.scala(L917)】**
* **执行作业时进行的闭包检测【SparkContext.scala(L1950)】**
* **执行有向无环图调度器运行作业时的闭包检测【SparkContext.scala(L1920)】**
* **因为没有shuffle，所以只有一个阶段ResultStage，进行任务关联时【DAGScheduler.scala(L993)】**

1. **两个分区数据，一次转换，一次聚合，一次行动**

val rdd: RDD[Int] = sc.makeRDD(List(1,2,3,4),2)

val acc = new MyAcc

sc.register(acc, "wordcount")

rdd

.map(num=>(num+"",num))

.repartition(2)

.foreach{

case (s:String, num:Int) => {

acc.add(num)

}

}

**源码执行时，有4处涉及序列化操作：**

* **行动算子foreach执行时进行的闭包检测【RDD.scala(L917)】**
* **执行作业时进行的闭包检测【SparkContext.scala(L1950)】**
* **执行有向无环图调度器运行作业时的闭包检测【SparkContext.scala(L1920)】**
* **因为没有shuffle，所以只有一个阶段ResultStage，进行任务关联时【DAGScheduler.scala(L993)】**

**总结：根据以上测试，可以大概判断出来，copy方法被调用的次数等于：**

**Copy次数 = 1（算子闭包检测） + 1（运行作业的闭包检测）+ 1（有向无环图调度的闭包检测）+1（阶段数量）=> 4**

**验证：**

* **下面基本的wordcount是几次copy？（4）**

val acc = new MyAcc

sc.register(acc, "wordcount")

val rdd: RDD[String] = sc.makeRDD(List("Hello Scala", "Hello Spark", "Hello Hadoop"))

rdd

.flatMap(\_.split(" "))

.map((\_,1))

.reduceByKey(\_+\_)

.foreach{

case (w,c)=>acc.add(c)

}

println("driver => " + acc.value)

* **下面复杂点的wordcount是几次copy？（6）**

val acc = new MyAcc

sc.register(acc, "wordcount")

val rdd: RDD[String] = sc.makeRDD(List("Hello Scala", "Hello Spark", "Hello Hadoop"))

rdd

.flatMap(\_.split(" "))

.map(s=>{

**acc.add(1)**

(s,1)

})

.reduceByKey(\_+\_)

.foreach{

case (w,c)=>acc.add(c)

}

println("driver => " + acc.value)

* **下面更复杂点的wordcount是几次copy？（8）**

val sc = new SparkContext(conf)

val acc = new MyAcc

sc.register(acc, "wordcount")

val rdd: RDD[String] = sc.makeRDD(List("Hello Scala", "Hello Spark", "Hello Hadoop"))

rdd

.flatMap(\_.split(" "))

.map(s=>{

**acc.add(1)**

(s,1)

})

.reduceByKey((x,y)=>{

**acc.add(1)**

x + y

})

.foreach{

case (w,c)=>acc.add(c)

}

println("driver => " + acc.value)

**另：以上测试结果不排除有的算子中包含嵌套操作导致多次的闭包检测问题，复杂问题复杂分析就可以了。**