Spark Checkpoint写操作代码分析

上次我对Spark RDD缓存的相关代码《Spark RDD缓存代码分析》进行了简要的介绍，本文将对Spark RDD的checkpint相关的代码进行相关的介绍。先来看看怎么使用checkpont：

```
scala> val data = sc.parallelize(List("www", "iteblog", "com"))
data: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[2] at parallelize at <console>:15

scala> sc.setCheckpointDir("/www/iteblog/com")

scala> data.checkpoint

scala> data.count
```

先是初始化好相关的RDD，因为checkpoint是将RDD中的数据写到磁盘，所以需要指定一个checkpint目录，也就是sc.setCheckpointDir("/www/iteblog/com")，这步执行完之后会在/www/iteblog/com路径下创建相关的文件夹，比如：/www/iteblog/com/ada54d92-eeb2-4cff-89fb-89a297edd4dc；然后对data RDD进行checkpoint，整个代码运行完，会在/www/iteblog/com/ada54d92-eeb2-4cff-89fb-89a297edd4dc生存相关的文件：

```
Found 4 items
-rw-r--r-- 3 iteblog iteblog 0 2015-11-25 15:05 /www/iteblog/com/ada54d92-eeb2-4cff-89fb-89a297edd4dc/rdd-2/part-00000
-rw-r--r-- 3 iteblog iteblog 5 2015-11-25 15:05 /www/iteblog/com/ada54d92-eeb2-4cff-89fb-89a297edd4dc/rdd-2/part-00001
-rw-r--r-- 3 iteblog iteblog 9 2015-11-25 15:05 /www/iteblog/com/ada54d92-eeb2-4cff-89fb-89a297edd4dc/rdd-2/part-00002
-rw-r--r-- 3 iteblog iteblog 5 2015-11-25 15:05 /www/iteblog/com/ada54d92-eeb2-4cff-89fb-89a297edd4dc/rdd-2/part-00003
```

现在来对checkpint的相关代码进行简单介绍。首先就是设置checkpint的目录，这个代码如下：

```
//////////////////////////////////////////////////////////////////////
 User: 过往记忆
 Date: 2015-11-25
 Time: 22:12
 bolg:
 本文地址：/archives/1535
 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 过往记忆博客微信公共帐号：iteblog_hadoop
//////////////////////////////////////////////////////////////////////

def setCheckpointDir(directory: String) {

  // If we are running on a cluster, log a warning if the directory is local.
  // Otherwise, the driver may attempt to reconstruct the checkpointed RDD from
  // its own local file system, which is incorrect because the checkpoint files
  // are actually on the executor machines.
  if (!isLocal && Utils.nonLocalPaths(directory).isEmpty) {
    logWarning("Checkpoint directory must be non-local " +
      "if Spark is running on a cluster: " + directory)
  }

  checkpointDir = Option(directory).map { dir =>
    val path = new Path(dir, UUID.randomUUID().toString)
    val fs = path.getFileSystem(hadoopConfiguration)
    fs.mkdirs(path)
```

```
    fs.getFileStatus(path).getPath.toString
  }
}
```

从上面注释可以看出，如果是非local模式，directory要求是HDFS上的目录。事实上，如果你是非local模式，但是指定的checkpint路径是本地路径，程序运行的时候会出现类似以下的异常：

```
org.apache.spark.SparkException: Checkpoint RDD ReliableCheckpointRDD[1] at count at <con
sole>:18(0) has different number of partitions from original RDD ParallelCollectionRDD[0] at p
arallelize at <console>:15(4)
 at org.apache.spark.rdd.ReliableRDDCheckpointData.doCheckpoint(ReliableRDDCheckpointDa
ta.scala:73)
 at org.apache.spark.rdd.RDDCheckpointData.checkpoint(RDDCheckpointData.scala:74)
 at org.apache.spark.rdd.RDD$$anonfun$doCheckpoint$1.apply$mcV$sp(RDD.scala:1655)
 at org.apache.spark.rdd.RDD$$anonfun$doCheckpoint$1.apply(RDD.scala:1652)
 at org.apache.spark.rdd.RDD$$anonfun$doCheckpoint$1.apply(RDD.scala:1652)
 at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:147)
 at org.apache.spark.rdd.RDD.doCheckpoint(RDD.scala:1651)
 at org.apache.spark.SparkContext.runJob(SparkContext.scala:1826)
 at org.apache.spark.SparkContext.runJob(SparkContext.scala:1837)
 at org.apache.spark.SparkContext.runJob(SparkContext.scala:1850)
 at org.apache.spark.SparkContext.runJob(SparkContext.scala:1921)
 at org.apache.spark.rdd.RDD.count(RDD.scala:1125)
 at $iwC$$iwC$$iwC$$iwC$$iwC$$iwC.<init>(<console>:18)
 at $iwC$$iwC$$iwC$$iwC$$iwC.<init>(<console>:23)
 at $iwC$$iwC$$iwC$$iwC.<init>(<console>:25)
 at $iwC$$iwC$$iwC.<init>(<console>:27)
 at $iwC$$iwC.<init>(<console>:29)
 at $iwC.<init>(<console>:31)
 at <init>(<console>:33)
 at .<init>(<console>:37)
 at .<clinit>(<console>)
 at .<init>(<console>:7)
 at .<clinit>(<console>)
 at $print(<console>)
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
 at java.lang.reflect.Method.invoke(Method.java:606)
 at org.apache.spark.repl.SparkIMain$ReadEvalPrint.call(SparkIMain.scala:1065)
 at org.apache.spark.repl.SparkIMain$Request.loadAndRun(SparkIMain.scala:1340)
 at org.apache.spark.repl.SparkIMain.loadAndRunReq$1(SparkIMain.scala:840)
 at org.apache.spark.repl.SparkIMain.interpret(SparkIMain.scala:871)
```

```
 at org.apache.spark.repl.SparkIMain.interpret(SparkIMain.scala:819)
 at org.apache.spark.repl.SparkILoop.reallyInterpret$1(SparkILoop.scala:857)
 at org.apache.spark.repl.SparkILoop.interpretStartingWith(SparkILoop.scala:902)
 at org.apache.spark.repl.SparkILoop.command(SparkILoop.scala:814)
 at org.apache.spark.repl.SparkILoop.processLine$1(SparkILoop.scala:657)
 at org.apache.spark.repl.SparkILoop.innerLoop$1(SparkILoop.scala:665)
 at org.apache.spark.repl.SparkILoop.org$apache$spark$repl$SparkILoop$$loop(SparkILoop.scala:670)
 at org.apache.spark.repl.SparkILoop$$anonfun$org$apache$spark$repl$SparkILoop$$process$1.apply$mcZ$sp(SparkILoop.scala:997)
 at org.apache.spark.repl.SparkILoop$$anonfun$org$apache$spark$repl$SparkILoop$$process$1.apply(SparkILoop.scala:945)
 at org.apache.spark.repl.SparkILoop$$anonfun$org$apache$spark$repl$SparkILoop$$process$1.apply(SparkILoop.scala:945)
 at scala.tools.nsc.util.ScalaClassLoader$.savingContextLoader(ScalaClassLoader.scala:135)
 at org.apache.spark.repl.SparkILoop.org$apache$spark$repl$SparkILoop$$process(SparkILoop.scala:945)
 at org.apache.spark.repl.SparkILoop.process(SparkILoop.scala:1059)
 at org.apache.spark.repl.Main$.main(Main.scala:31)
 at org.apache.spark.repl.Main.main(Main.scala)
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
 at java.lang.reflect.Method.invoke(Method.java:606)
 at org.apache.spark.deploy.SparkSubmit$.org$apache$spark$deploy$SparkSubmit$$runMain(SparkSubmit.scala:674)
 at org.apache.spark.deploy.SparkSubmit$.doRunMain$1(SparkSubmit.scala:180)
 at org.apache.spark.deploy.SparkSubmit$.submit(SparkSubmit.scala:205)
 at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:120)
 at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
```

setCheckpointDir的过程主要是在指定的目录下创建一个文件夹，这个文件夹会在后面用到。然后我们对RDD进行checkpoint，主要做的事情如下：

```
//////////////////////////////////////////////////////////////////////
 User: 过往记忆
 Date: 2015-11-25
 Time: 22:12
 bolg:
 本文地址：/archives/1535
 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 过往记忆博客微信公共帐号：iteblog_hadoop
//////////////////////////////////////////////////////////////////////
```

```
def checkpoint(): Unit = RDDCheckpointData.synchronized {
  // NOTE: we use a global lock here due to complexities downstream with ensuring
  // children RDD partitions point to the correct parent partitions. In the future
  // we should revisit this consideration.
  if (context.checkpointDir.isEmpty) {
    throw new SparkException("Checkpoint directory has not been set in the SparkContext")
  } else if (checkpointData.isEmpty) {
    checkpointData = Some(new ReliableRDDCheckpointData(this))
  }
}
```

程序第一步就是判断checkpointDir是否为空，如果为空直接抛出异常，而这个checkpointDir是由上面的setCheckpointDir函数设置的。这里我们应该设置了checkpointDir，所以直接判断checkpointData.isEmpty是否成立，checkpointData是什么东西呢？它的类型如下：

```
private[spark] var checkpointData: Option[RDDCheckpointData[T]] = None
```

RDDCheckpointData类是和RDD一一对应的，保存着一切和RDD checkpint相关的所有信息，而且具体的Checkpint操作都是它（子类）进行的。而对RDD调用checkpoint函数主要就是初始化ReliableRDDCheckpointData对象，供以后进行checkpint操作。从这段代码我们知道，对RDD调用checkpoint函数，其实就是初始化了checkpointData，并不立即执行checkpint操作，你可以理解成这里只是对RDD进行checkpint标记操作。

那什么触发真正的checkpoint操作？仔细看上面例子，执行data.count之后才会生成checkpoint文件。是的，只有在Action触发Job的时候才会进行checkpint。Spark在执行完Job之后会判断是否需要checkpint：

```
def runJob[T, U: ClassTag](
    rdd: RDD[T],
    func: (TaskContext, Iterator[T]) => U,
    partitions: Seq[Int],
    resultHandler: (Int, U) => Unit): Unit = {
  if (stopped.get()) {
    throw new IllegalStateException("SparkContext has been shutdown")
  }
  val callSite = getCallSite
  val cleanedFunc = clean(func)
  logInfo("Starting job: " + callSite.shortForm)
  if (conf.getBoolean("spark.logLineage", false)) {
```

```
    logInfo("RDD's recursive dependencies:₩n" + rdd.toDebugString)
  }
  dagScheduler.runJob(rdd, cleanedFunc, partitions, callSite, resultHandler, localProperties.get)
  progressBar.foreach(_.finishAll())
  rdd.doCheckpoint()
}
```

注意看最后一句代码rdd.doCheckpoint()，这个就是触发RDD的checkpoint的，而doCheckpoint函数的实现如下：

```
private[spark] def doCheckpoint(): Unit = {
  RDDOperationScope.withScope(sc, "checkpoint", allowNesting = false, ignoreParent = true) {
    if (!doCheckpointCalled) {
      doCheckpointCalled = true
      if (checkpointData.isDefined) {
        checkpointData.get.checkpoint()
      } else {
        dependencies.foreach(_.rdd.doCheckpoint())
      }
    }
  }
}
```

又看到checkpointData了吧？这个就是在执行checkpint()函数定义的，所以如果你的RDD调用了checkpint()函数，那么checkpointData.isDefined肯定是true的。而如果你的父RDD调用了checkpint()函数，最后也会执行你父RDD的checkpointData.get.checkpoint()代码。我们来看看checkpointData中的checkpoint()是如何实现的，代码如下：

```
final def checkpoint(): Unit = {
  // Guard against multiple threads checkpointing the same RDD by
  // atomically flipping the state of this RDDCheckpointData
  RDDCheckpointData.synchronized {
    if (cpState == Initialized) {
      cpState = CheckpointingInProgress
    } else {
      return
    }
  }

  val newRDD = doCheckpoint()
```

```
  // Update our state and truncate the RDD lineage
  RDDCheckpointData.synchronized {
    cpRDD = Some(newRDD)
    cpState = Checkpointed
    rdd.markCheckpointed()
  }
}
```

为了防止多个线程对同一个RDD进行checkpint操作，首先是把checkpint的状态由Initialized变成CheckpointingInProgress，所以如果另一个线程发现checkpint的状态不是Initialized就直接return了。最后就是doCheckpoint实现了：

```
//////////////////////////////////////////////////////////////////////
 User: 过往记忆
 Date: 2015-11-25
 Time: 22:12
 bolg:
 本文地址：/archives/1535
 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 过往记忆博客微信公共帐号：iteblog_hadoop
//////////////////////////////////////////////////////////////////////

protected override def doCheckpoint(): CheckpointRDD[T] = {

  // Create the output path for the checkpoint
  val path = new Path(cpDir)
  val fs = path.getFileSystem(rdd.context.hadoopConfiguration)
  if (!fs.mkdirs(path)) {
    throw new SparkException(s"Failed to create checkpoint path $cpDir")
  }

  // Save to file, and reload it as an RDD
  val broadcastedConf = rdd.context.broadcast(
    new SerializableConfiguration(rdd.context.hadoopConfiguration))
  // TODO: This is expensive because it computes the RDD again unnecessarily (SPARK-8582)
  rdd.context.runJob(rdd, ReliableCheckpointRDD.writeCheckpointFile[T](cpDir, broadcastedCo
nf) _)
  val newRDD = new ReliableCheckpointRDD[T](rdd.context, cpDir)
  if (newRDD.partitions.length != rdd.partitions.length) {
    throw new SparkException(
      s"Checkpoint RDD $newRDD(${newRDD.partitions.length}) has different " +
        s"number of partitions from original RDD $rdd(${rdd.partitions.length})")
```

```
  }

  // Optionally clean our checkpoint files if the reference is out of scope
  if (rdd.conf.getBoolean("spark.cleaner.referenceTracking.cleanCheckpoints", false)) {
    rdd.context.cleaner.foreach { cleaner =>
      cleaner.registerRDDCheckpointDataForCleanup(newRDD, rdd.id)
    }
  }

  logInfo(s"Done checkpointing RDD ${rdd.id} to $cpDir, new parent is RDD ${newRDD.id}")

  newRDD
}
```

首先是创建写RDD的目录，然后启动一个Job去写Checkpint文件，主要由ReliableCheckpointRDD.writeCheckpointFile来实现写操作。

```
def writeCheckpointFile[T: ClassTag](
    path: String,
    broadcastedConf: Broadcast[SerializableConfiguration],
    blockSize: Int = -1)(ctx: TaskContext, iterator: Iterator[T]) {
  val env = SparkEnv.get
  val outputDir = new Path(path)
  val fs = outputDir.getFileSystem(broadcastedConf.value.value)

  val finalOutputName = ReliableCheckpointRDD.checkpointFileName(ctx.partitionId())
  val finalOutputPath = new Path(outputDir, finalOutputName)
  val tempOutputPath =
    new Path(outputDir, s".$finalOutputName-attempt-${ctx.attemptNumber()}")

  if (fs.exists(tempOutputPath)) {
    throw new IOException(s"Checkpoint failed: temporary path $tempOutputPath already exist
s")
  }
  val bufferSize = env.conf.getInt("spark.buffer.size", 65536)

  val fileOutputStream = if (blockSize < 0) {
    fs.create(tempOutputPath, false, bufferSize)
  } else {
    // This is mainly for testing purpose
    fs.create(tempOutputPath, false, bufferSize, fs.getDefaultReplication, blockSize)
  }
  val serializer = env.serializer.newInstance()
```

```
  val serializeStream = serializer.serializeStream(fileOutputStream)
  Utils.tryWithSafeFinally {
    serializeStream.writeAll(iterator)
  } {
    serializeStream.close()
  }

  if (!fs.rename(tempOutputPath, finalOutputPath)) {
    if (!fs.exists(finalOutputPath)) {
      logInfo(s"Deleting tempOutputPath $tempOutputPath")
      fs.delete(tempOutputPath, false)
      throw new IOException("Checkpoint failed: failed to save output of task: " +
        s"${ctx.attemptNumber()} and final output path does not exist: $finalOutputPath")
    } else {
      // Some other copy of this task must've finished before us and renamed it
      logInfo(s"Final output path $finalOutputPath already exists; not overwriting it")
      fs.delete(tempOutputPath, false)
    }
  }
}
```

写完Checkpint文件之后，会返回newRDD，并最后赋值给cpRDD，并将Checkpint的状态变成Checkpointed。最后将这个RDD的依赖全部清除（markCheckpointed()）

```
private[spark] def markCheckpointed(): Unit = {
  clearDependencies()
  partitions_ = null
  deps = null    // Forget the constructor argument for dependencies too
}
```

整个写操作就完成了。明天将会介绍如何从Checkpint中恢复RDD，请关注本博客，或者iteblog_hadoop公共账号。