Spark Checkpoint读操作代码分析

上次介绍了RDD的Checkpint写过程（《Spark Checkpoint写操作代码分析》），本文将介绍RDD如何读取已经Checkpint的数据。在RDD Checkpint完之后，Checkpint的信息（比如数据存放的目录）都由RDDCheckpointData去管理，所以当下次计算依赖了这个RDD的时候，首先是根据依赖关系判断出当前这个RDD是否被Checkpint了，主要是通过RDD的dependencies决定：

```
final def dependencies: Seq[Dependency[_]] = {
  checkpointRDD.map(r => List(new OneToOneDependency(r))).getOrElse {
    if (dependencies_ == null) {
      dependencies_ = getDependencies
    }
    dependencies_
  }
}
```

如果RDD被Checkpint了，那么checkpointRDD为Some(CheckpointRDD[T])了，所以依赖的RDD变成了CheckpointRDD。在计算数据的过程中会调用RDD的iterator方法：

```
final def iterator(split: Partition, context: TaskContext): Iterator[T] = {
  if (storageLevel != StorageLevel.NONE) {
    SparkEnv.get.cacheManager.getOrCompute(this, split, context, storageLevel)
  } else {
    computeOrReadCheckpoint(split, context)
  }
}

private[spark] def computeOrReadCheckpoint(split: Partition, context: TaskContext): Iterator[T]
 =
{
   if (isCheckpointed) firstParent[T].iterator(split, context) else compute(split, context)
}
```

计算的过程中首先会判断RDD是否被Checkpint了，而RDD Checkpint写之后这个条件肯定是true的。而firstParent已经变成了CheckpointRDD，所以会调用CheckpointRDD的iterator方法，该方法最终会调用ReliableCheckpointRDD的compute方法：

```
override def compute(split: Partition, context: TaskContext): Iterator[T] = {
  val file = new Path(checkpointPath, ReliableCheckpointRDD.checkpointFileName(split.index))
  ReliableCheckpointRDD.readCheckpointFile(file, broadcastedConf, context)
}
```

在compute方法中会通过ReliableCheckpointRDD的readCheckpointFile方法来从file路径里面读出已经Checkpint的数据，readCheckpointFile的实现如下：

```
def readCheckpointFile[T](
    path: Path,
    broadcastedConf: Broadcast[SerializableConfiguration],
    context: TaskContext): Iterator[T] = {
  val env = SparkEnv.get
  val fs = path.getFileSystem(broadcastedConf.value.value)
  val bufferSize = env.conf.getInt("spark.buffer.size", 65536)
  val fileInputStream = fs.open(path, bufferSize)
  val serializer = env.serializer.newInstance()
  val deserializeStream = serializer.deserializeStream(fileInputStream)

  // Register an on-task-completion callback to close the input stream.
  context.addTaskCompletionListener(context => deserializeStream.close())

  deserializeStream.asIterator.asInstanceOf[Iterator[T]]
}
```

最后数据就回被全部读取出来，整个Checkpint读过程完成了。