# Spark-sql与Spark-core的结合

**刘凯**

注意spark2.3,DataFrame被定义成为一个Type对象：

type DataFrame = Dataset[Row]

**以DataSet的show方法的源码**来一步步解析spark-sql是如何与spark-core相结合的：

DataSet.show()=>DataSet.show(20)=>DataSet.showString (numRows, truncate = 20)

=>head(20)=> withAction("head", limit(n).queryExecution)(collectFromPlan)

SparkSql由Sql逻辑转为Rdd的代码是在SparkPlan的doExecute方法实现的。Show()方法最终的返回值为Array[InternalRow]。而Rdd的执行结果是如何放入Array[InternalRow]请看第4个方法。

而Spark-sql的核心逻辑则在QueryExecutiron。该类描述了一个sql如何成为物理执行计划的整个流程。

**核心方法代码：**

1 withAction方法

// Wrap a Dataset action to track the QueryExecution and time cost, then report to the

// user-registered callback functions.

该方法最后的返回值为Array[InternalRow]请看第4步。

private def withAction[U](name: String, qe: QueryExecution)(action: SparkPlan => U) = {

try {

qe.executedPlan.foreach { plan =>

plan.resetMetrics()

}

val start = System.nanoTime()

val result = SQLExecution.withNewExecutionId(sparkSession, qe) {

action(qe.**executedPlan**)

}

val end = System.nanoTime()

sparkSession.listenerManager.onSuccess(name, qe, end - start)

result

} catch {

case e: Exception =>

sparkSession.listenerManager.onFailure(name, qe, e)

throw e

}

}

2 withNewExecutionId方法：

Wrap an action that will execute "queryExecution" to track all Spark jobs in the body so that

we can connect them with an execution.

def withNewExecutionId[T](

sparkSession: SparkSession,

queryExecution: QueryExecution)(body: => T): T = {

val sc = sparkSession.sparkContext

val oldExecutionId = sc.getLocalProperty(EXECUTION\_ID\_KEY)

val executionId = SQLExecution.nextExecutionId

sc.setLocalProperty(EXECUTION\_ID\_KEY, executionId.toString)

executionIdToQueryExecution.put(executionId, queryExecution)

try {

// sparkContext.getCallSite() would first try to pick up any call site that was previously

// set, then fall back to Utils.getCallSite(); call Utils.getCallSite() directly on

// streaming queries would give us call site like "run at <unknown>:0"

val callSite = sparkSession.sparkContext.getCallSite()

sparkSession.sparkContext.listenerBus.post(SparkListenerSQLExecutionStart(

executionId, callSite.shortForm, callSite.longForm, queryExecution.toString,

SparkPlanInfo.fromSparkPlan(queryExecution.executedPlan), System.currentTimeMillis()))

try {

body

} finally {

sparkSession.sparkContext.listenerBus.post(SparkListenerSQLExecutionEnd(

executionId, System.currentTimeMillis()))

}

} finally {

executionIdToQueryExecution.remove(executionId)

sc.setLocalProperty(EXECUTION\_ID\_KEY, oldExecutionId)

}

}

3 collectFromPlan将executedPlan转为RDD的方法

private def collectFromPlan(plan: SparkPlan): Array[T] = {

// This projection writes output to a `InternalRow`, which means applying this projection is not

// thread-safe. Here we create the projection inside this method to make `Dataset` thread-safe.

val objProj = GenerateSafeProjection.generate(deserializer :: Nil)

**plan.executeCollect()**.map { row =>

// The row returned by SafeProjection is `SpecificInternalRow`, which ignore the data type

// parameter of its `get` method, so it's safe to use null here.

objProj(row).get(0, null).asInstanceOf[T]

}

}

4 plan.executeCollect()方法：

def executeCollect(): Array[InternalRow] = {

val byteArrayRdd = getByteArrayRdd()

val results = ArrayBuffer[InternalRow]()

byteArrayRdd.collect().foreach { countAndBytes =>

decodeUnsafeRows(countAndBytes.\_2).foreach(results.+=)

}

results.toArray

}

5 getByteArrayRdd(n: Int = -1) 获得RDD[(Long, Array[Byte])]

private def getByteArrayRdd(n: Int = -1): RDD[(Long, Array[Byte])] = {

execute().mapPartitionsInternal { iter =>

var count = 0

val buffer = new Array[Byte](4 << 10) // 4K

val codec = CompressionCodec.createCodec(SparkEnv.get.conf)

val bos = new ByteArrayOutputStream()

val out = new DataOutputStream(codec.compressedOutputStream(bos))

while (iter.hasNext && (n < 0 || count < n)) {

val row = iter.next().asInstanceOf[UnsafeRow]

out.writeInt(row.getSizeInBytes)

row.writeToStream(out, buffer)

count += 1

}

out.writeInt(-1)

out.flush()

out.close()

Iterator((count, bos.toByteArray))

}

}

6 execute()方法

final def execute(): RDD[InternalRow] = executeQuery {

if (isCanonicalizedPlan) {

throw new IllegalStateException("A canonicalized plan is not supposed to be executed.")

}

doExecute()

}

7 executeQuery

对SparkPlan的DoExecute()的包装

final def execute(): RDD[InternalRow] = executeQuery {

if (isCanonicalizedPlan) {

throw new IllegalStateException("A canonicalized plan is not supposed to be executed.")

}

doExecute()

}

8 doExecute(): RDD[InternalRow]方法

该方法是核心方法是将SparkPlan（物理执行计划）转为Rdd的核心方法

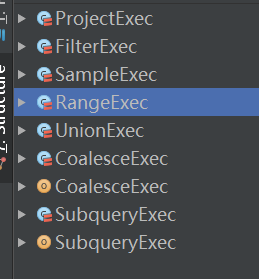
Produces the result of the query as an `RDD[InternalRow]`

Overridden by concrete implementations of SparkPlan.

**SparkPlan：**

The base class for physical operators.

The naming convention is that physical operators end with "Exec" suffix, e.g. [[ProjectExec]].



我们看一个最简单的doExecute()方法：

case class CoalesceExec(numPartitions: Int, child: SparkPlan) extends UnaryExecNode {

override def output: Seq[Attribute] = child.output

override def outputPartitioning: Partitioning = {

if (numPartitions == 1) SinglePartition

else UnknownPartitioning(numPartitions)

}

protected override def doExecute(): RDD[InternalRow] = {

if (numPartitions == 1 && child.execute().getNumPartitions < 1) {

// Make sure we don't output an RDD with 0 partitions, when claiming that we have a

// `SinglePartition`.

new CoalesceExec.EmptyRDDWithPartitions(sparkContext, numPartitions)

} else {

child.execute().coalesce(numPartitions, shuffle = false)

}

}

}