零、几个Greenplum的核心概念

- QD（Query Dispatcher、查询调度器）：Master节点上负责处理用户查询请求的进程，在PostgreSQL中称之为Backend进程。
- QE（Query Executor、查询执行器）：Segment 上负责执行 QD 分发来的查询任务的进程。
- Slice：Greenplum 把一个完整的分布式查询计划从下到上分成多个 Slice，通过Motion分割。
- Gang：在不同segments上执行同一个slice的所有QEs进程（可能在不同机器上）。
- Locus：数据的分布特性，主要包括分片的（MPP最常见的）、单体的（一般和Gather相关）和复制的（复制表）。
- Motion：在Gang之间通过Interconnect进行数据重分布的执行器节点，主要包括BroadCastMotion（广播），GatherMotion（聚集），HashedMotion（Hash分布）。

一、数据准备

```
CREATE TABLE athletes (id int, name text) DISTRIBUTED BY (id);
CREATE TABLE sports(id int, sport_name text, athlete_id int) DISTRIBUTED BY (id);
INSERT INTO athletes VALUES (1, 'Yongtao Huang'), (2, 'Ran Liu'), (3, 'Hongfei
Lyv');
INSERT INTO sports VALUES (1, 'Wrestling', 1), (2, 'Wrestling', 2), (3, 'Yoga',
3);

gpadmin=# \d
              List of relations
 Schema |   Name   | Type  |  Owner  | Storage
--------+----------+-------+---------+---------
 public | athletes | table | gpadmin | heap
 public | sports   | table | gpadmin | heap
(2 rows)
```

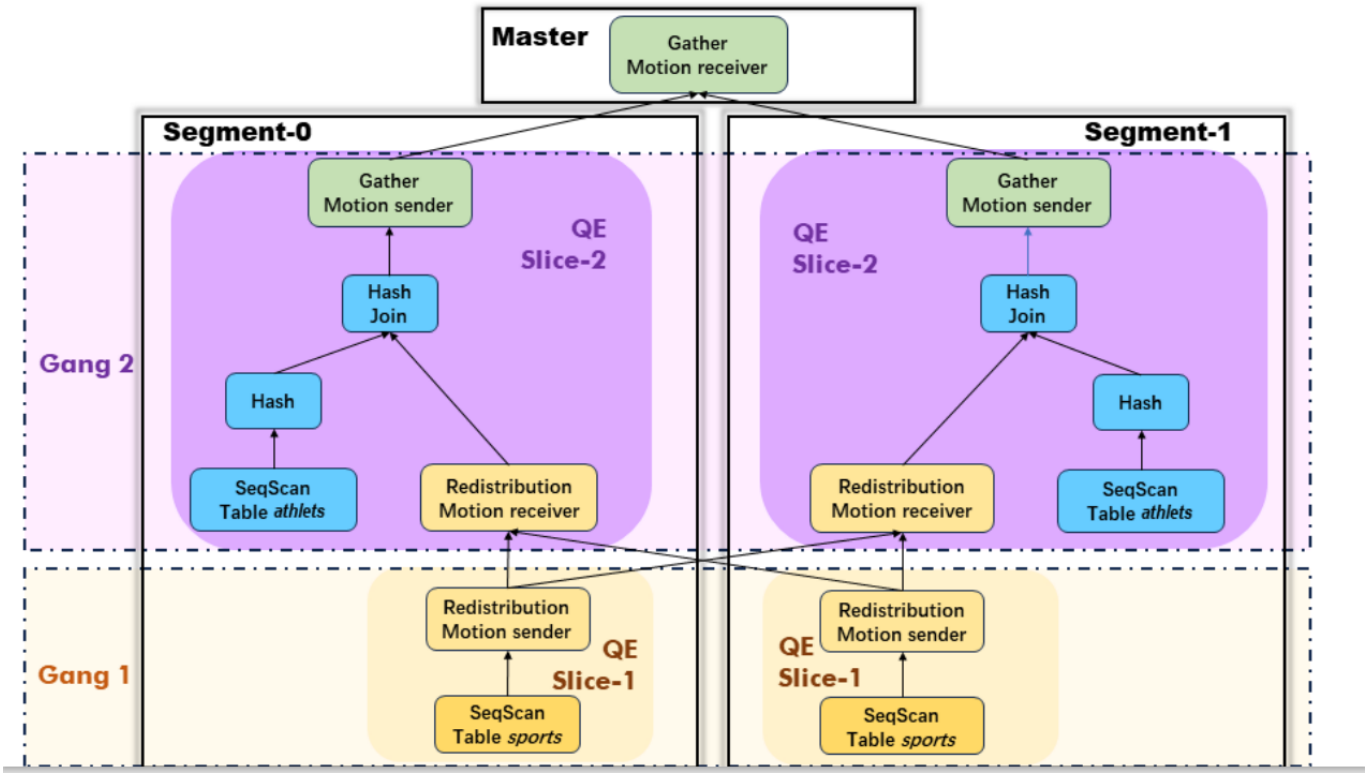
运行一个带Gather和Redistribute Motion的SELECT

```
gpadmin=# SELECT a.name athlete_name, s.sport_name
gpadmin-# FROM athletes a, sports s
gpadmin-# WHERE a.id=s.athlete_id;
 athlete_name  | sport_name
---------------+------------
 Ran Liu       | Wrestling
 Hongfei Lyv   | Yoga
 Yongtao Huang | Wrestling
(3 rows)

gpadmin=#
gpadmin=# explain SELECT a.name athlete_name, s.sport_name
gpadmin-# FROM athletes a, sports s
gpadmin-# WHERE a.id=s.athlete_id;
                                          QUERY PLAN
-----------------------------------------------------------------------------------
------------------
```

```
 Gather Motion 2:1  (slice1; segments: 2)  (cost=608.00..120128.28 rows=2316320
width=64)
   ->  Hash Join  (cost=608.00..85383.48 rows=1158160 width=64)
         Hash Cond: (s.athlete_id = a.id)
         ->  Redistribute Motion 2:2  (slice2; segments: 2)  (cost=0.00..750.50
rows=23350 width=36)
               Hash Key: s.athlete_id
               ->  Seq Scan on sports s  (cost=0.00..283.50 rows=23350 width=36)
         ->  Hash  (cost=298.00..298.00 rows=24800 width=36)
               ->  Seq Scan on athletes a  (cost=0.00..298.00 rows=24800 width=36)
 Optimizer: Postgres-based planner
(9 rows)
```

explain结果图片化，如下所示：



- Gang1：两个segments上执行SeqScan表sports，并重分布发送给其它Segment的所有相关进程。
- Gang2：两个segments上执行SeqScan表athlets，并和重分布接收的结果进行Hash Join，最终通过Gather节点发送给QD的的所有相关进程。

图中涉及两个slice，事实上还有一个slice-0，工作在Master上。从下面的GDB结果可以看出，`numSlices = 3`。

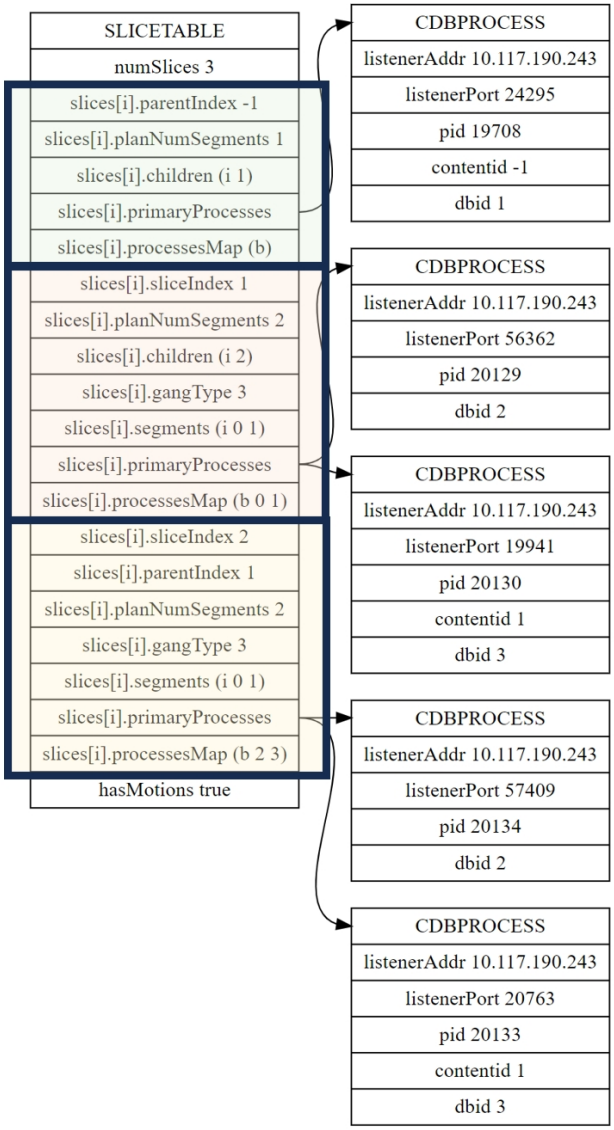
二、获取GDB信息

断点打在AssignGangs()函数的最后一行，也就是`b src/backend/executor/execUtils.c: 1339`。

```
[gpadmin@gpdb-local gpdb]$ ps aux | grep postgres | grep con
gpadmin  19708  0.0  0.0 466712  9024 ?        tsl  11:33   0:00 postgres: 15432,
gpadmin gpadmin [local] con26238 cmd3 SELECT
gpadmin  20129  0.0  0.0 526112  7544 ?        Ssl  11:34   0:00 postgres: 15434,
gpadmin gpadmin 10.117.190.243(12086) con26238 seg0 idle
```

```
gpadmin  20130  0.0  0.0 526112  7548 ?        Ssl  11:34   0:00 postgres: 15435,
gpadmin gpadmin 10.117.190.243(27174) con26238 seg1 idle
gpadmin  20134  0.0  0.0 526112  5232 ?        Ssl  11:34   0:00 postgres: 15434,
gpadmin gpadmin 10.117.190.243(12090) con26238 seg0 idle
gpadmin  20133  0.0  0.0 526112  5236 ?        Ssl  11:34   0:00 postgres: 15435,
gpadmin gpadmin 10.117.190.243(27178) con26238 seg1 idle
```

可以用工具把SliceTable结构体图片化，如下所示，用三种颜色区分出3个ExecSlice：



集群是1个Master2个Segment的。其中

- 19708是QD进程
- 20129和20130是两个Segment的一个Slices（QE进程组）
- 20134和20133是两个Segment的另外一个Slices（QE进程组）

```
(gdb) p *sliceTable
$1 = {type = T_SliceTable, localSlice = 0, numSlices = 3, slices = 0x2a2ce50,
hasMotions = true, instrument_options = 0, ic_instance_id = 0}
(gdb) p nodeToString(sliceTable)
```

```
$2 = 0x2a4f0f8 "{SLICETABLE :localSlice 0 :numSlices 3 :slices[i].sliceIndex 0
:slices[i].rootIndex 0 :slices[i].parentIndex -1 :slices[i].planNumSegments 1
:slices[i].children (i 1) :slices[i].gangType 0 :slices[i].segments <>
:slices[i].primaryGang <> :slices[i].primaryProcesses ({CDBPROCESS :listenerAddr
\\10.117.190.243 :listenerPort 24295 :pid 19708 :contentid -1 :dbid 1})
:slices[i].processesMap (b) :slices[i].sliceIndex 1 :slices[i].rootIndex 0
:slices[i].parentIndex 0 :slices[i].planNumSegments 2 :slices[i].children (i 2)
:slices[i].gangType 3 :slices[i].segments (i 0 1) :slices[i].primaryGang <>
:slices[i].primaryProcesses ({CDBPROCESS :listenerAddr \\10.117.190.243
:listenerPort 56362 :pid 20129 :contentid 0 :dbid 2} {CDBPROCESS :listenerAddr
\\10.117.190.243 :listenerPort 19941 :pid 20130 :contentid 1 :dbid 3})
:slices[i].processesMap (b 0 1) :slices[i].sliceIndex 2 :slices[i].rootIndex 0
:slices[i].parentIndex 1 :slices[i].planNumSegments 2 :slices[i].children <>
:slices[i].gangType 3 :slices[i].segments (i 0 1) :slices[i].primaryGang <>
:slices[i].primaryProcesses ({CDBPROCESS :listenerAddr \\10.117.190.243
:listenerPort 57409 :pid 20134 :contentid 0 :dbid 2} {CDBPROCESS :listenerAddr
\\10.117.190.243 :listenerPort 20763 :pid 20133 :contentid 1 :dbid 3})
:slices[i].processesMap (b 2 3) :hasMotions true :instrument_options 0
:ic_instance_id 0}"
```

```
typedef struct ExecSlice // 生成Gang的进程组和建立Interconnect的依据
{
    int         sliceIndex; // sliceIndex代表本节点在数组中的位置
    int         rootIndex;
    int         parentIndex; // 父节点的位置
    int         planNumSegments;
    List       *children; // 子节点的位置列表
    GangType    gangType;
    List        *segments;
    struct Gang *primaryGang; // 一个Gang结构体的指针，里面包含着生成gang进程组和建立
Interconnect的主要信息
    List        *primaryProcesses;
    Bitmapset   *processesMap; // 一个由QE唯一标志组成的bitmap，里面保存了所有要执行此
slice的QE的唯一标志。
} ExecSlice;
```

其中SliceTable中的slices数组（ExecSlice数组）是一颗树，通过sliceIndex，parentIndex和children联系起来。如上图，父子节点关系为 `slice-0-> slice-1 -> slice-2`.