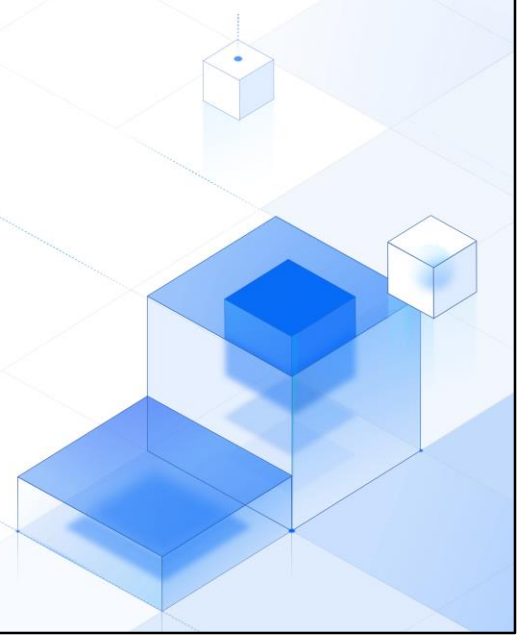
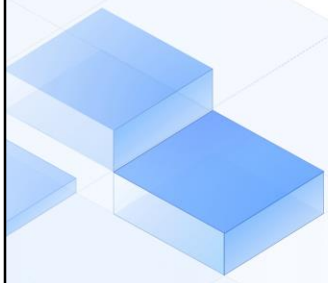
OCEANBASE

OBCP 认证培训

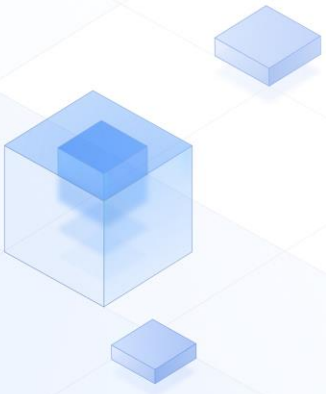


目录



OCEANBASE

目录

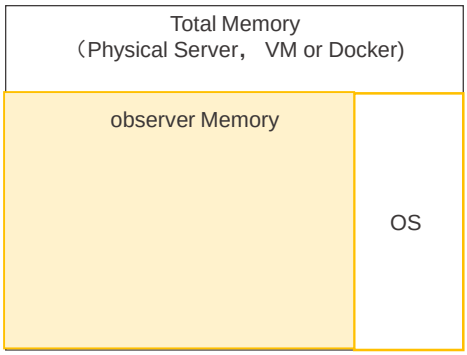


OCEANBASE


2.1 内存管理
OCEANBASE

2.1 内存结构（一）

- OceanBase是支持多租户架构的准内存分布式数据库，对大容量内存的管理和使用提出了很高要求。
- OceanBase会占据物理服务器的大部分内存并进行统一管理。



通过参数设定observer占用的内存上限

- memory_limit_percentage
- memory_limit

OCEANBASE

2.1 内存结构（二）

OceanBase提供两种方式设置observer内存上限

- 按照物理机器总内存的百分比计算observer内存上限：由memory_limit_percentage参数配置。
- 直接设置observer内存上限：由memory_limit参数配置。

memory_limit=0时，memory_limit_percentage决定observer内存大小；否则由memory_limit决定observer内存大小。

以100GB物理内存的机器为例，下述表格展示了不同配置下机器上的observer内存上限：

	memory_limit_percentage	memory_limit	observer内存上限
场景1	80	0	80GB
场景2	80	90GB	90GB

场景1：memory_limit=0，因此由memory_limit_percentage确定observer内存大小，即100GB*80% = 80GB。

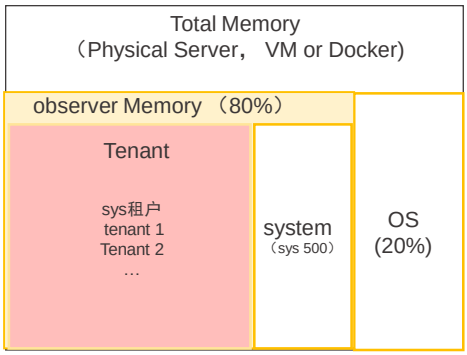
场景2：memory_limit='90GB'，因此observer内存上限就是90GB，memory_limit_percentage参数失效。

OCEANBASE

2.1 内存结构（三）

OB系统内部内存

- 每一个observer都包含多个租户（sys租户 & 非sys租户）的数据，但observer的内存并不是全部分配给租户。
- observer中有些内存不属于任何租户，属于所有租户共享的资源，称为“系统内部内存”。



通过参数设定“系统内部内存”上限

- system_memory

租户可用的总内存

- “observer内存上限” – “系统内部内存”

OCEANBASE

OceanBase 支持多租户架构，但是 OceanBase 内存上限中配置的内容并不能全部分配给租户使用。因为每一个 OBServer上租户都会共享部分资源或功能，这些资源或功能所使用的内存由于并不属于任何一个普通租户，所以这类内存被归结为系统内部内存。系统内部内存可使用的内存上限是可以通过 system_memory配置参数配置的，它的含义是系统内部可使用 OceanBase 内存上限的百分比。

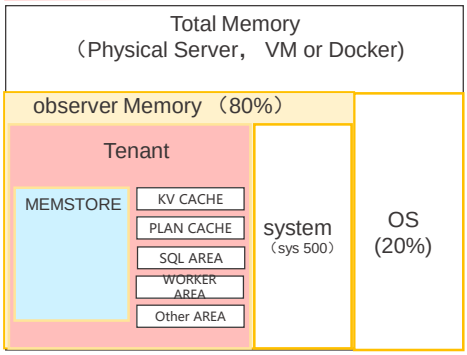
例如，假设 OceanBase 内存上限为 80 GB， system_memory 的值为 8G，可用于租户分配的内存就是剩下80-8=72 GB。

2.1 内存结构（四）

每个租户内部的内存总体上分为两个部分

- 不可动态伸缩的内存：MemStore。
- 可动态伸缩的内存：KVCache

MemStore用来保存DML产生的增量数据，空间不可被占用；KVCache空间会被其它众多内存模块复用。



MemStore

- 大小由参数memstore_limit_percentage决定，表示租户的 MemStore 部分占租户总内存的百分比。
- 默认值为50，即占用租户内存的50%。
- 当MemStore内存使用超过freeze_trigger_percentage定义的百分比时，触发冻结及后续的转储/合并等行为。

KVCache

- 保存来自SSTable的热数据，提高查询速度。
- 大小可动态伸缩，会被其它各种Cache挤占。

OCEANBASE

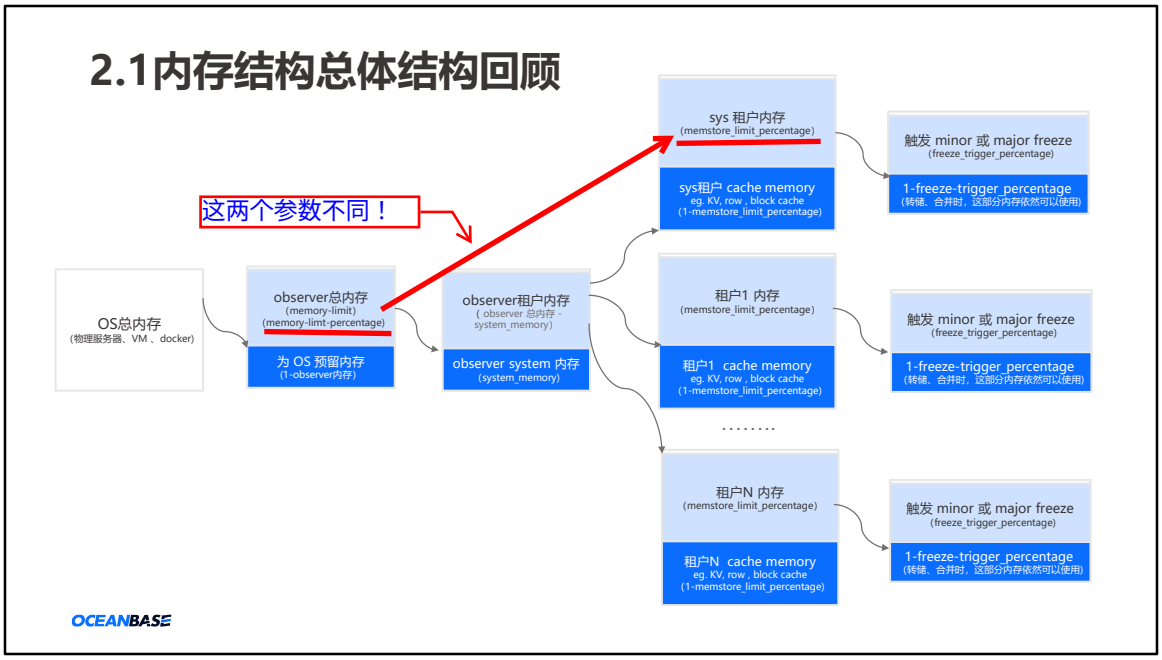
OceanBase把租户内部的内存总体上分为两个部分：
1. 不可动态伸缩的内存
2. 可动态伸缩的内存
其中，不可动态伸缩的内存主要由保存数据库增量更新的MemStore使用；可动态伸缩的内存主要由KVCache进行管理。
可动态伸缩的KVCache会尽量使用除去不可动态伸缩后租户的全部内存。
除去memstore和kvcache，其他模块的内存使用大小默认不超过10G
select * from gv$memory where used > 1024*1024*1024*10 and CONTEXT not in ('OB_MEMSTORE','OB_KVSTORE_CACHE');

KVcache中的重要组成部分:
sys租户：location_cache；location_cache中存放partition的 leader 信息；
普通租户：user_block_cache、block_index_cache、bf_cache、user_row_cache;

PLANCHE
PLANCACHE中缓存的执行计划是真实的。(explain 看到的执行计划是预估的)
用途：避免硬解析SQL语句，从CACHE中获得语句的“已分析”版本，加速语句执行。

SQL AREA:
执行 sql 需要的内存，主要是parser和优化器使用。
内存模块称作 OB_SQL_ARENA。

WORKER AREA:
工作线程所占用的内存。
内存模块称作 OB_WORKER_EXECUTION。



OceanBase 占用了服务器的大量内存 （OBServer Total Memory），然后一部分用于自身系统运行（Memory Reserved For OBServer），一部分用于划分给创建的租户（Allocatable Memory For OBServer)。每个租户等同于传统数据库的一个实例，不同租户的内存模块组成是一样的，其内存又分为装载增量数据的 MemStore（Tenant MemStore）以及 KVCache 缓存（Tenant Cache Memory）。

2.1 常见内存问题处理：外部客户常见报错处理

```
ERROR 4030 (HY000): OB-4030:Over tenant memory limits
```

- 判断MemStore是否超过上限

```
select /*+ READ_CONSISTENCY(WEAK),query_timeout(100000000) */ TENANT_ID,IP,
  round(ACTIVE/1024/1024/1024,2)ACTIVE_GB,
  round(TOTAL/1024/1024/1024,2) TOTAL_GB,
  round(FREEZE_TRIGGER/1024/1024/1024,2) FREEZE_TRIGGER_GB,
  round(TOTAL/FREEZE_TRIGGER*100,2) percent_trigger,
  round(MEM_LIMIT/1024/1024/1024,2) MEM_LIMIT_GB
from gv$memstore
where tenant_id >1000 or TENANT_ID=1
order by tenant_id,TOTAL_GB desc;
```

查看TOTAL_GB是否已经达到MEM_LIMIT_GB，即已经将MemStore全部写满。

当您看到上述错误信息时，首先需判断是不是 MemStore 内存超限，当 MmeStore 内存超限时，需要检查数据写入是否过量或未做限流。当遇到大量写入且数据转储跟不上写入速度的时候就会报这种错误。运行下述语句查看内存状态。

该问题的紧急应对措施是增加租户内存。问题解决之后需要分析原因，如果是因为未做限流引起，需要加上相应措施，然后回滚之前加上的租户内存动作。如果确实因为业务规模增长导致租户内存不足以支撑业务时，需要根据转储的频度设置合理的租户内存大小。

2.1 常见内存问题处理：外部客户常见报错处理

```
ERROR 4030 (HY000): OB-4030:Over tenant memory limits
```

- 如MemStore内存未超限，判断是MemStore之外的哪个module占用内存空间最高

```
select tenant_id, svr_ip, mod_name, sum(hold) module_sum
from __all_virtual_memory_info
where tenant_id>1000 and hold<>0 and
      mod_name not in ('OB_KVSTORE_CACHE', 'OB_MEMSTORE')
group by tenant_id,svr_ip, mod_name
order by module_sum desc;
```

查看排名靠前的内存模块。

判断标准：module_sum > min_memory - 租户memstore

模块内存超限，可能需要先调整单独模块的内存，如ob_sql_work_area_percentage（排序、分布式中间结果）、ob_interm_result_mem_limit（分布式中间结果），如租户内存过小，也需要加租户内存

2.1 常见内存问题处理：外部客户常见报错处理

```
ERROR 4030 (HY000): OB-4030:Over tenant memory limits
```

- 除去MemStore和KVCache，查看使用超过一定大小（比如10GB）的内存模块

```
select *
from gv$memory
where used > 1024*1024*1024*10
      and CONTEXT not in ('OB_MEMSTORE','OB_KVSTORE_CACHE')
order by used desc;
```

查看排名靠前的内存模块。

2.1 常见内存问题处理：外部客户常见报错处理

500租户内存超限

- tenant_id=500的租户是OB内部租户，简称500租户。
- 500租户的内存使用量没有被v$memory和gv$memory统计，需要单独查询__all_virtual_memory_info表

```
select svr_ip,mod_name,sum(hold) system_memory_sum
from __all_virtual_memory_info
where tenant_id=500 and hold<>0
group by svr_ip,mod_name
order by system_memory_sum desc;
```

查看排名靠前的内存模块。

判断标准：system_memory_sum > SYSTEM_CACHE（默认租户内存的20%）

2.1 常见内存问题处理：外部客户常见报错处理

alloc memory或allocate memory相关的报错

- observer日志报错信息举例

…… alloc memory failed(ret=-4013, size=65536, mem=NULL, label_="OB_SQL_HASH_SET", ctx_id_=9……

…… oops, alloc failed, tenant_id=1001 ctx_id=9 hold=266338304 limit=268435455 ……

…… allocate memory failed(nbyte=213552, operator type="PHY_MULTI_PART_INSERT") ……

报错的原因，通常是系统内存耗尽，或者已经达到了内存使用的上限。

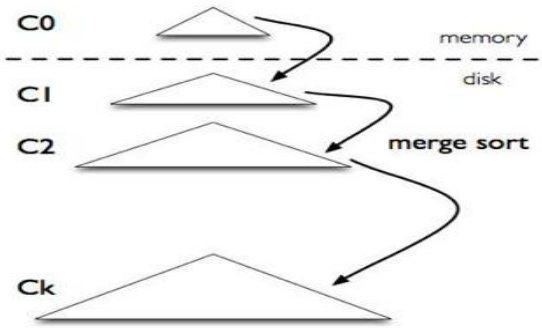
如第一个常见内存问题中所述，模块内存超限，可能需要先调整单独模块的内存。如租户内存过小，也需要加租户内存。


2.2 内存数据落盘策略
2.2.1 转储与合并
2.2.2 合并操作策略
2.2.3 转储操作策略
OCEANBASE

2.2.1 LSM Tree 技术简介

LSM Tree（The Log-Structured Merge-Tree）简介

- 将某个对象（Partition）中的数据按照“key-value”形式在磁盘上有序存储（SSTable）；
- 数据更新先记录在MemStore中的MemTable里，然后再合并（Merge）到底层的ssTable里；
- SSTable和MemTable之间可以有多级中间数据，同样以key-value形式保存在磁盘上，逐级向下合并。



OCEANBASE

作为企业IT基础架构的核心部分，数据库技术一直是大家讨论的热点，也是很多人关注的领域。如果简单划分的话，数据库内核可以分为计算层和存储层，其中计算层负责接收用户发送过来的SQL语句，调用存储的功能来实现数据的存取，所以存储层的设计会直接影响计算层存取的效率，影响SQL语句的性能。
相对于传统的page based数据库存储方式，OceanBase使用了现在非常流行的LSM Tree作为存储引擎保存数据的基本数据结构，这在分布式的通用关系型数据库当中是很少见的。

首先需要说明的是，LSM Tree技术出现的一个最主要的原因就是磁盘的随机写速度要远远低于顺序写的速度，而数据库要面临很多写密集型的场景，所以很多数据库产品就把LSM Tree的思想引入到了数据库领域。LSM Tree，顾名思义，就是The Log-Structured Merge-Tree 的缩写。从这个名称里面可以看到几个关键的信息： 第一：log-structred，通过日志的方式来组织的 第二：merge，可以合并的 第三：tree，一种树形结构实际上它并不是一棵树，也不是一种具体的数据结构，它实际上是一种数据保存和更新的思想。简单的说，就是将数据按照key来进行排序（在数据库中就是表的主键），之后形成一棵一棵小的树形结构，或者不是树形结构，是一张小表也可以，这些数据通常被称为基线数据；之后把每次数据的改变（也就是log）都记录下来，也按照主键进行排序，之后定期的把log中对数据的改变合并（merge）到基线数据当中。下面的图形描述

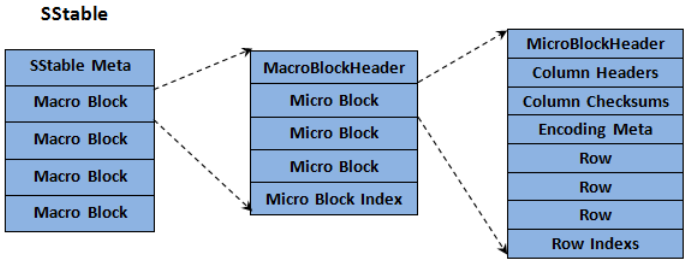
了LSM Tree的基本结构。

图中的C0代表了缓存在内存中的数据，当内存中的数据达到了一定的阈值后，就会把数据内存中的数据排序后保存到磁盘当中，这就形成了磁盘中C1级别的增量数据（这些数据也是按照主键排序的），这个过程通常被称为转储。当C1级别的数据也达到一定阈值的时候，就会触发另外的一次合并（合并的过程可以认为是一种归并排序的过程），形成C2级别的数据，以此类推，如果这个逐级合并的结构定义了k层的话，那么最后的第k层数据就是最后的基线数据，这个过程通常被称为合并。**用一句话来简单描述的话，LSM Tree就是一个基于归并排序的数据存储思想**。从上面的结构中不难看出，LSM Tree对写密集型的应用是非常友好的，因为绝大部分的写操作都是顺序的。但是对很多读操作是要损失一些性能的，因为数据在磁盘上可能存在多个版本，所以通常情况下，使用了LSM Tree的存储引擎都会选择把很多个版本的数据存在内存中，根据查询的需要，构建出满足要求的数据版本。在数据库领域，很多产品都使用了LSM Tree结构来作为数据库的存储引擎，例如：OceanBase，LevelDB，HBase等。

2.2.1 SSTable

SSTable

- 将数据按照主键或者隐含列（不可见）的顺序在磁盘上有序排列，以B+ Tree数据结构实现“key-value”存储。
- 数据被分成2MB的固定大小宏块（Macro Block），每个宏块包含一定key值范围的数据。
- 为了避免读取少量数据时的“读放大”，每个宏块内部又分为多个微块（Micro Block），大小一般配为16KB（可变）；微块内的记录数和具体的数据特征相关。



OCEANBASE

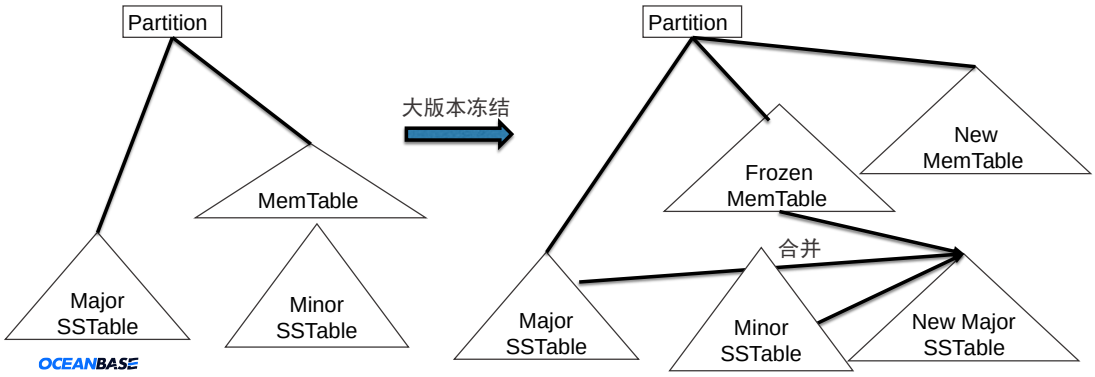
OceanBase的数据文件以宏块（Macro Block）为单位组织数据，每个宏块大小为2MB。宏块内部又划分出很多个16K（压缩前的大小）大小的微块(Micro Block)，而每个微块里面包含多个行(Row)。OceanBase内部IO的最小单位是微块。

宏块可以合并和分裂。由于删除数据，相邻的几个宏块中的所有行可以在一个宏块中存放时，相邻的多个宏块会合并成一个宏块；由于宏块内插入和更新数据导致空间不足，需要将数据存放到多个宏块中时，宏块会进行分裂。

2.2.1 基于 LSM Tree 的实践：合并

OceanBase中最简单的LSM Tree只有C0层（MemTable）和C1层（SSTable）。两层数据的合并过程如下：

- 将所有observer上的MemTable数据做大版本冻结（Major Freeze），其余内存作为新的MemTable继续使用；
- 将冻结后的MemTable数据合并（Merge）到SSTable中，形成新的SSTable，并覆盖旧的SSTable；
- 合并完成后，冻结的MemTable内存才可以被清空并重新使用。

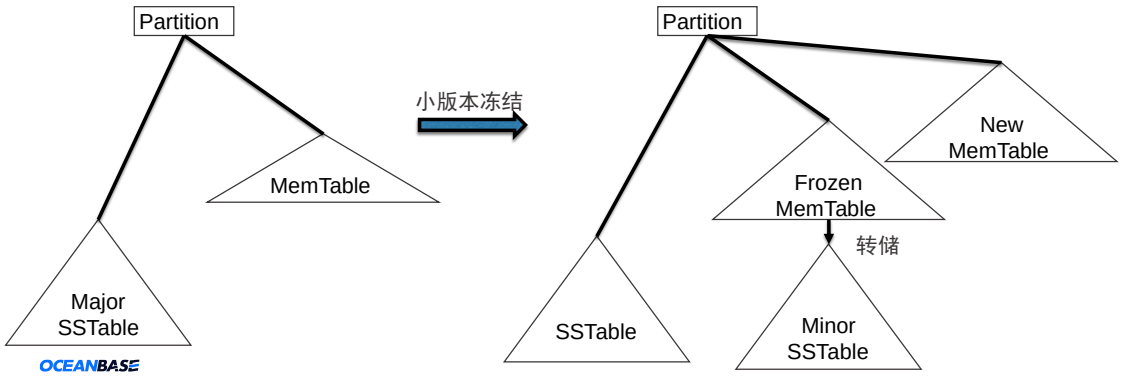


合并：合并操作（Major Freeze）是将动静态数据做归并，也就是产生新的C1层的数据，会比较费时。当转储产生的增量数据积累到一定程度时，通过Major Freeze实现大版本的合并。由于在合并的过程中为了保证数据的一致性，就需要在合并的过程中暂停正在被合并的数据上的事务，这对性能来说是会有影响的，OceanBase对合并操作进行了细化，分为增量合并，轮转合并和全量合并。

2.2.1 基于 LSM Tree 的实践：转储

为了解决2层LSM Tree合并时引发的问题（资源消耗大，内存释放速度慢等），引入了“转储”机制：

- 将MemTable数据做小版本冻结（Minor Freeze）后写到磁盘上单独的转储文件里，不与SSTable数据做合并；
- 转储文件写完之后，冻结的MemTable内存被清空并重新使用；
- 每次转储会将MemTable数据与前一次转储的数据合并（Merge），转储文件最终会合并到SSTable中。



OceanBase数据库采用了基于LSM Tree结构作为数据库的存储引擎，数据被分为基线数据（SSTable）和增量数据（MemTable）两部分，基线数据被保存在磁盘中，当需要读取的时候会被加载到数据库的缓存中，当数据被不断插入（或者修改时）在内存中缓存增量数据，当增量数据达到一定阈值时，就把增量数据刷新到磁盘上，当磁盘上的增量数据达到一定阈值时再把磁盘上的增量数据和基线数据进行合并。

对于LSM Tree结构，如果保存多个层次的MemTable的话，会带来很大的空间存储问题，OceanBase对LSM Tree结构进行了简化，只保留了C0层和C1层，也就是说，内存中的增量数据会被以MemTable的方式保存在磁盘中，这个过程被称之为转储（compaction），当转储了一定的次数之后，就需要把磁盘上的MemTable与基线数据进行合并（merge）

2.2.2 OB合并触发方式

三种合并触发方式

- 定时合并
- MemStore使用率达到阈值自动合并
- 手动合并

OCEANBASE

2.2.2 OB合并方式：定时合并

- 由major_freeze_duty_time参数控制定时合并时间，可以修改参数控制合并时间：

```
alter system set major_freeze_duty_time='02:00'
```

```
mysql> show parameters like '%major_freeze_duty_time%' \G;
*************************** 1. row ***************************
         zone: ET15SQA_1
     svr_type: observer
       svr_ip: 100.81.166.54
     svr_port: 2882
         name: major_freeze_duty_time
    data_type: NULL
        value: 02:00
 value_strict: NULL
         info: the start time of system daily merge procedure. Range: [00:00, 24:00)
  need_reboot: 0
      section: daily_merge
visible_level:
```

OCEANBASE

2.2.2 OB合并方式：MemStore使用率达到阈值自动合并

当租户的 MemStore内存使用率达到freeze_trigger_percentage参数的值， 并且转储的次数已经达到了major_compact_trigger/minor_freeze_times参数的值，会自动触发合并：

- 通过查询(g)v$memstore视图来查看各租户的memstore内存使用情况。
- 可以修改以下参数的值来影响触发合并的时机：

```
alter system set freeze_trigger_percentage = 40;
alter system set major_compact_trigger = 100;
```

```
mysql> select * from oceanbase.v$memstore;
+-----------+-----------+-----------+--------------------+--------------------+
| TENANT_ID | ACTIVE    | TOTAL     | FREEZE_TRIGGER     | MEM_LIMIT          |
+-----------+-----------+-----------+--------------------+--------------------+
|         1 | 849175576 | 855385112 |        12025908370 |        17179869150 |
|       500 |         0 |         0 | 3228180212899171530 | 4611686018427387900 |
|      1010 |  66379344 | 104234576 |          751619260 |         1073741800 |
|      1013 | 381563184 | 489353520 |          751619260 |         1073741800 |
|      1016 |         0 |  37691392 |          375809630 |          536870900 |
|      1025 | 610066512 | 642424912 |         7516192740 |        10737418200 |
|      1034 |         0 |    532480 |         7516192740 |        10737418200 |
+-----------+-----------+-----------+--------------------+--------------------+
```

OCEANBASE

某个租户的 memstore 写到一定的比例会后会自行发起冻结，所谓 memstore，是指租户申请的内存资源中有多少可以来存放更新数据， 比如一个租户可使用的内存为8G(建立租户时，Resource Unit 的 min_memory)， 有一个参数 memstore_limit_percentage 来控制有多少内存可以用来存写入的数据(其余的内存会被用作其它用途，比如缓存)。

假如 memstore_limit_percentage 为 50%, 即 memstore 的总大小为 4G。
当 memstore 写入超过一定比例 (由参数 freeze_trigger_percentage 控制， 默认为70%)，故该租户的 memstore 超过 4G *0.7 = 2.8G时会触发冻结。可以通过 `select* from oceanbase.v$memstore来查看各租户的memstore`信息。

可使用show parameters like ‘minor_freeze%’查看再两次合并之间可以有多少次转储。

2.2.2 OB合并方式：手动合并

- 可以在"root@sys"用户下，通过以下命令发起手动合并（忽略当前MemStore的使用率）：

```
alter system major freeze;
```

- 合并发起以后，可以在"oceanbase"数据库里用以下命令查看合并状态：

```
select * from __all_zone; 或者 select * from __all_zone where name = 'merge_status';
```

```
mysql> select * from __all_zone where zone='ET15SQA_3';
+----------------------------+----------------------------+-----------+--------------------+-----------------+----------------+
| gmt_create                 | gmt_modified               | zone      | name               | value           | info           |
+----------------------------+----------------------------+-----------+--------------------+-----------------+----------------+
| 2016-04-29 14:55:53.378903 | 2017-06-26 02:04:14.325282 | ET15SQA_3 | all_merged_version |            1249 |                |
| 2016-04-29 14:55:53.378402 | 2017-06-26 02:00:06.590930 | ET15SQA_3 | broadcast_version  |            1249 |                |
| 2016-04-29 14:55:53.379224 | 2017-06-26 02:04:14.324942 | ET15SQA_3 | is_merge_timeout   |               0 |                |
| 2016-04-29 14:55:53.378242 | 2017-06-26 02:04:14.323853 | ET15SQA_3 | is_merging         |               0 |                |
| 2016-04-29 14:55:53.378723 | 2017-06-26 02:04:14.324597 | ET15SQA_3 | last_merged_time   | 1498413854323150 |                |
| 2016-04-29 14:55:53.378562 | 2017-06-26 02:04:14.324284 | ET15SQA_3 | last_merged_version |            1249 |                |
| 2016-04-29 14:55:53.379064 | 2017-06-26 02:00:06.591288 | ET15SQA_3 | merge_start_time   | 1498413606590133 |                |
| 2016-04-29 14:55:53.379596 | 2017-06-26 02:04:14.325831 | ET15SQA_3 | merge_status       |               0 | IDLE           |
| 2017-01-03 14:47:29.144475 | 2017-01-03 14:47:29.144475 | ET15SQA_3 | region             |               0 | default_region |
| 2016-04-29 14:55:53.378083 | 2017-06-10 20:11:38.874648 | ET15SQA_3 | status             |               2 | ACTIVE         |
| 2016-04-29 14:55:53.379436 | 2016-04-29 14:55:53.379436 | ET15SQA_3 | suspend_merging    |               0 |                |
+----------------------------+----------------------------+-----------+--------------------+-----------------+----------------+
11 rows in set (0.05 sec)
```

OCEANBASE

2.2.2 轮转合并

借助自身天然具备的多副本分布式架构，OceanBase引入了轮转合并机制：

- 一般情况下，OceanBase会有3份（或更多）数据副本；可以轮流为每份副本单独做合并。
- 当一个副本在合并时，这个副本上的业务流量可以暂时切到其它没有合并的副本上。
- 某个副本合并完成后，将流量切回这个副本，然后以类似的方式为下一个副本做合并，直至所有副本完成合并。

关于轮转合并的更多说明：

- 通过参数enable_merge_by_turn开启或者关闭轮转合并。
- 以ZONE为单位轮转合并，只有一个ZONE合并完成后才开始下一个ZONE的合并；合并整体时间变长。
- 某一个ZONE的合并开始之前，会将这个ZONE上的Leader服务切换到其它ZONE；切换动作对长事务有影响。
- 由于正在合并的ZONE上没有Leader，避免了合并对在线服务带来的性能影响。

OCEANBASE

轮转合并是一种各个副本轮流进行合并的策略既可用于全量合并，也可用于增量合并。对全量合并和增量合并而言是一个正交的概念，可以配置也可以不配置。如果不配置，则多个副本同步合并。

2.2.2 设置轮转合并顺序

- 合并开始前，通过参数zone_merge_order设置合并顺序；只对轮转合并有效。
- 场景举例

  假设集群中有三个zone，分别是z1,z2,z3，想设置轮转合并的顺序为"z1 -> z2 -> z3"，步骤如下：

```
alter system set enable_manual_merge = false;  -- 关闭手动合并
alter system set enable_merge_by_turn = true;  -- 开启轮转合并
alter system set zone_merge_order = 'z1,z2,z3';  -- 设置合并顺序
```

- 取消自定义的合并顺序

```
alter system set zone_merge_order = '';  -- 取消自定义合并顺序
```

OCEANBASE

在加zone或者减zone的情况下，用户设置的旧的zone_merge_order在每日合并的时候，不会生效，同时会报警

2.2.2 OB轮转合并示例

假设集群中的设置是zone_merge_order = 'z1,z2,z3,z4,z5', zone_merge_concurrency = 3，一次轮转合并的大概过程如下：

事件	调度	并发合并的ZONE	合并完成的ZONE
1. 开始合并。	z1,z2,z3发起合并	z1,z2,z3	
2. 一段时间后，z2完成合并。	z4发起合并	z1,z3,z4	z2
3. 一段时间后，z3完成合并。	z5发起合并	z1,z4,z5	z2,z3
4. 一段时间后，全部ZONE完成合并。			z1,z2,z3,z4,z5

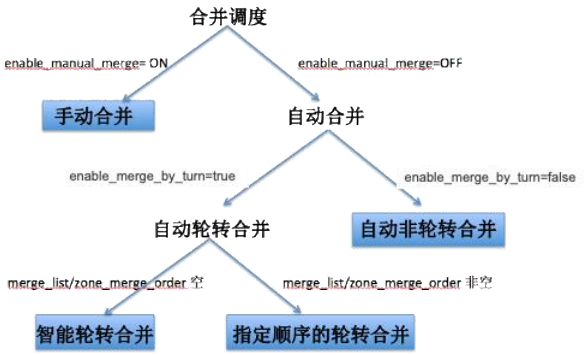
OCEANBASE

合并切主检查以后，如果zone_merge_list设置的不合理，则有可能影响每日合并的并发度

2.2.2 OB 每日合并策略

可通过以下参数控制每日合并的策略：

- enable_manual_merge: 是否开启手动合并（默认值False）；
- enable_merge_by_turn: 是否开启自动轮转合并（默认值True）；
- zone_merge_order: 指定自动轮转合并的合并顺序（默认值NULL）；



OCEANBASE

简要来说：

手动合并：通过alter system的命令指定zone开始合并；

自动非轮转合并：所有zone一起进行合并；不做切主操作，不受合并并发度的限制；

自动智能轮转合并：RS按照一定策略依次调度起每个zone开始合并，直到所有zone都合并完成；

自动指定顺序的轮转合并：用户指定合并的顺序，RS

根据这个顺序依次调度起zone进行合并；

zone_merge_concurrency： 每日合并的并发度；在各种合并策略下的作用不尽相同；具体在下面的版本介绍里面给出；

2.2.2 合并控制

合并线程数，由参数merge_thread_count控制

- 控制可以同时执行合并的分区个数；单分区暂不支持拆分合并，分区表可以加快合并速度。
- 默认值为0，实际取值为`min(10, cpu_cnt * 0.3)`。
- 最大取值不要超过48：值太大会占用太多CPU和IO资源，对observer的性能影响较大；而且容易触发系统报警，比如CPU使用率超过90%可能会触发主机报警。
- 如对合并速度没有特殊要求，建议使用默认值0 。

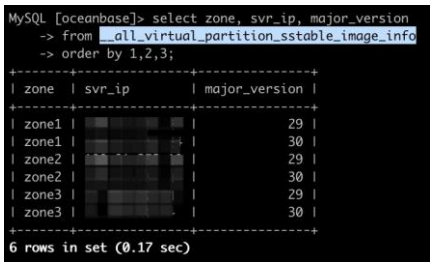
OCEANBASE

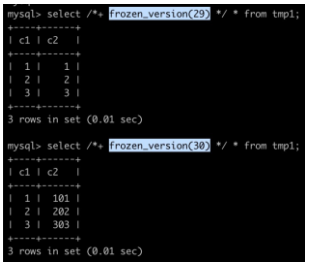
2.2.2 合并控制

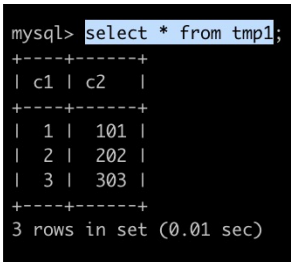
设置SSTable中保留的数据合并版本个数

- 由参数max_kept_major_version_number控制，默认值为2。
- 调大参数值可以保留更多历史数据，但同时占用更多的存储空间。
- 在hint中利用`frozen_version(<major_version>)`指定历史版本。

实际用例







2.2.2 合并注意事项

合并超时时间

- 由参数zone_merge_timeout定义超时阈值；默认值为'3h'（3个小时）。
- 如果某个ZONE的合并执行超过阈值，合并状态被设置为TIMEOUT。

空间警告水位

- 由参数data_disk_usage_limit_percentage定义数据盘空间使用阈值，默认值90。
- 当数据盘空间使用量超过阈值后，合并任务打印ERROR警告日志，合并任务失败；需要尽快扩大数据盘物理空间，并调大data_disk_usage_limit_percentage参数的值。
- 当数据盘空间使用量超过阈值后，禁止数据迁入。

OCEANBASE

2.2.2 查看合并记录和状态

__all_rootservice_event_history表，查看合并记录：

version	zone	start_time	finish_time	merge_time_minute	module	event
215	[redacted]	2019-07-15 02:00:00.578840	2019-07-15 02:38:59.361596	38	daily_merge	start_merge
215	[redacted]	2019-07-15 02:00:00.581834	2019-07-15 02:38:59.364602	38	daily_merge	start_merge
215	[redacted]	2019-07-15 02:00:00.584847	2019-07-15 02:38:55.337379	38	daily_merge	start_merge
216	[redacted]	2019-07-16 02:00:00.714304	2019-07-16 02:28:58.388486	28	daily_merge	start_merge
216	[redacted]	2019-07-16 02:00:00.717919	2019-07-16 02:28:58.391507	28	daily_merge	start_merge
216	[redacted]	2019-07-16 02:00:00.721122	2019-07-16 02:28:54.423908	28	daily_merge	start_merge

__all_zone表，查看当前合并状态：

最近一次合并的版本号

zone	name	value
	frozen_version	216
	last_merged_version	216
[redacted]	last_merged_version	216
[redacted]	last_merged_version	216
[redacted]	last_merged_version	216

当前未在合并

zone	name	value	info
	merge_status	0	IDLE
[redacted]	merge_status	0	IDLE
[redacted]	merge_status	0	IDLE
[redacted]	merge_status	0	IDLE

2.2.2 查看 OB 集群合并和冻结状态

```
OceanBase (root@oceanbase)> select gmt_modified,zone,name,value,info from __all_zone;
+----------------------------+-------+--------------------------+----------------+-----------+
| gmt_modified               | zone  | name                     | value          | info      |
+----------------------------+-------+--------------------------+----------------+-----------+
| 2016-05-09 13:52:22.440089 |       | cluster                  |              0 | ob1.maoqi |
| 2016-05-09 13:52:22.459746 |       | config_version           |              0 |           |
| 2016-05-09 13:52:22.449436 |       | frozen_time              |              0 |           |
| 2016-05-09 13:52:22.449013 |       | frozen_version           |              1 |           |
| 2016-05-09 13:52:22.449851 |       | global_broadcast_version |              1 |           |
| 2016-05-09 13:52:22.468929 |       | is_merge_error           |              0 |           |
| 2016-05-09 13:52:22.458695 |       | last_merged_version      |              1 |           |
| 2016-05-09 13:52:22.460153 |       | lease_info_version       |              0 |           |
| 2016-05-09 13:52:22.469481 |       | merge_list               |              0 |           |
| 2016-05-09 13:52:22.469949 |       | merge_status             |              0 | IDLE      |
| 2016-05-09 13:52:22.459323 |       | privilege_version        |              0 |           |
| 2016-05-09 13:52:22.448586 |       | try_frozen_version       |              1 |           |
| 2016-05-09 13:52:22.480815 | zone1 | all_merged_version       |              1 |           |
| 2016-05-09 13:52:22.479565 | zone1 | broadcast_version        |              1 |           |
| 2016-05-09 13:52:22.489516 | zone1 | is_merge_timeout         |              0 |           |
| 2016-05-09 13:52:22.479039 | zone1 | is_merging               |              0 |           |
| 2016-05-09 13:52:22.480399 | zone1 | last_merged_time         | 1462773142470278 |           |
| 2016-05-09 13:52:22.479968 | zone1 | last_merged_version      |              1 |           |
| 2016-05-09 13:52:22.489112 | zone1 | merge_start_time         | 1462773142470278 |           |
| 2016-05-09 13:52:22.490351 | zone1 | merge_status             |              0 | IDLE      |
| 2016-05-09 13:52:22.470383 | zone1 | status                   |              2 | ACTIVE    |
| 2016-05-09 13:52:22.489941 | zone1 | suspend_merging          |              0 |           |
+----------------------------+-------+--------------------------+----------------+-----------+
```

合并状态：

- IDLE
  - 未合并
- MERGING
  - 正在合并
- TIMEOUT
  - 合并超时
- ERROR
  - 合并出错

全局信息：

frozen_time 表示冻结时间，在上面的例子里，这个值为0，说明系统还没有发生过冻结。

frozen_version 表示版本号，从1开始。

global_broadcast_version 表示这个版本已经通过各ObServer进行合并。

is_merge_error 表示合并过程中是否出现错误。1表示合并遇到了问题，这种

情况一般需要手工处理。
last_merged_version 表示上次合并完成的版本。
try_frozen_version 表示正在进行哪个版本的冻结，如果try_frozen_version和frozne_version相等，则表示该版本已经完成冻结。
merge_list 表示ZONE的合并顺序。

zone相关信息：
all_merged_version 表示本zone最近一次已经合并完成的版本。
broadcast_version 表示本zone收到的可以进行合并的版本。
is_merge_timeout 如果在一段时间还没有合并完成，则将此字段置为1，表示合并时间太长，具体时间可以通过参数zone_merge_timeout来控制。
merge_start_time / last_merged_time 分别表示合并开始、结束时间。
last_merged_version 表示本zone最近一

次合并完成的版本。
merge_status 表示本zone的合并状态，取值有：
IDLE
MERGING
TIMEOUT
ERROR
suspend_merging 表示是否暂停合并，有时候因为合并造成压力过大或其它原因，我们想暂停合并，可以通过 alter system suspend merge 来暂停合并，通过alter system resume merge来恢复合并

2.2.2 问答 ?!

- freeze_trigger_percentage 这个参数的主要目的是什么？ 它的取值需要考虑哪些因素?
- 轮转合并主要解决了什么问题？ 又引入了什么新问题?
- 如何针对每日合并做性能调优?

OCEANBASE

关闭每日合并:

major_freeze_duty_time = 'disable'
ENABLE_MANUAL_MERGE = true

2.2.3 转储的基本概念

转储功能的引入，是为了解决合并操作引发的一系列问题

- 资源消耗高，对在线业务性能影响较大。
- 单个租户MemStore使用率高会触发集群级合并，其它租户成为受害者。
- 合并耗时长，MemStore内存释放不及时，容易造成MemStore满而数据写入失败的情况。

转储的基本设计思路

- 每个MemStore触发单独的冻结（freeze_trigger_percentage）及数据合并，不影响其它租户；也可以通过命令为指定指定租户、指定observer、指定分区做转储。
- 只和上一次转储的数据做合并，不和SSTable的数据做合并。

OCEANBASE

2.2.3 转储带来的影响

转储的优势

- 每个租户的转储不影响observer上其它的租户，也不会触发集群级转储，避免关联影响。
- 资源消耗小，对在线业务性能影响较低。
- 耗时相对较短，MemStore更快释放，降低发生MemStore写满的概率。

转储的副作用

- 数据层级增多，查询链路变长，查询性能下降。
- 冗余数据增多，占用更多磁盘空间。

OCEANBASE

2.2.3 转储相关参数

major_compact_trigger /minor_freeze_times

- 控制两次合并之间的转储次数，达到此次数则自动触发合并（Major Freeze）。
- 设置为 0表示关闭转储，即每次租户MemStore使用率达到冻结阈值（freeze_trigger_percentage）都直接触发集群合并。

minor_merge_concurrency

- 并发做转储的分区个数；单个分区暂时不支持拆分转储，分区表可加快速度。
- 并发转储的分区过少，会影响转储的性能和效果（比如MemStore内存释放不够快）。
- 并发转储的分区过多，同样会消耗过多资源，影响在线交易的性能。

OCEANBASE

2.2.3 转储适用场景

转储功能比较适用于以下场景

- 批处理、大量数据导入等场景，写MemStore的速度很快，需要MemStore内存尽快释放。
- 业务峰值交易量大，写入MemStore的数据很多，但又不想在峰值时段触发合并（Major Freeze），希望能将合并延后。

转储场景的常用配置方法

- 减小freeze_trigger_percentage的值（比如40），使MemStore尽早释放，进一步降低MemStore写满的概率。
- 增大major_compact_trigger /minor_freeze_times的值，尽量避免峰值交易时段触发合并（Major Freeze），将合并的时机延后到交易低谷时段的每日合并（major_freeze_duty_time）。

OCEANBASE

2.2.3 手动触发转储

```
ALTER SYSTEM MINOR FREEZE
    [{TENANT[=] ('tt1' [, 'tt2'...]) | PARTITION_ID [=] 'partidx%partcount@tableid'}]
    [SERVER [=] ('ip:port' [, 'ip:port'...])];
```

几个可选的控制参数

- tenant : 指定要执行minor freeze的租户
- partition_id : 指定要执行minor freeze的partition
- server : 指定要执行minor freeze的observer

当什么选项都不指定时，默认对所有observer上的所有租户执行转储。

手动触发的转储次数不受参数minor_freeze_times的限制，即手动触发的转储次数即使超过设置的次数，也不会触发合并（Major Freeze）。

OCEANBASE

组合使用这些参数可以达到不同粒度的控制效果，以下列举出几种常用的使用方式：

- 指定tenant: 对指定tenant在所有该租户有unit的observer上执行minor freeze
- 指定partition_id: 对指定partition的所有replica执行minor freeze
- 指定server: 对指定server执行minor freeze
- 指定tenant & server: 对指定tenant在指定的server上执行minor freeze
- 指定partition_id & server: 对指定partition在指定server上执行minor freeze

2.2.3 如何查看转储的记录

MemStore使用率达到freeze_trigger_percentage而触发的租户级转储，在__all_server_event_history表中查询：

```
MySQL [oceanbase]> select
    -> *
    -> from
    ->   __all_server_event_history
    -> where
    ->   (event like "%merge%" or event like "%minor%")
    -> order by
    ->   gmt_create desc limit 50
    -> ;
+----------------------------+-----------------+----------+--------------+-------------------------+-----------+--------+
| gmt_create                 | svr_ip          | svr_port | module       | event                   | name1     | value1 |
+----------------------------+-----------------+----------+--------------+-------------------------+-----------+--------+
| 2020-12-15 00:03:27.326387 | [redacted]      |    25882 | freeze       | do minor freeze success | tenant_id | 1001   |
| 2020-12-15 00:03:27.325064 | [redacted]      |    25882 | minor_merge  | minor merge start       | tenant_id | 1001   |
| 2020-12-15 00:03:27.289327 | [redacted]      |    25882 | freeze       | do minor freeze         | tenant_id | 1001   |
| 2020-12-15 00:03:25.774260 | [redacted]      |    25882 | freeze       | do minor freeze success | tenant_id | 1001   |
| 2020-12-15 00:03:25.774196 | [redacted]      |    25882 | minor_merge  | minor merge start       | tenant_id | 1001   |
| 2020-12-15 00:03:25.742751 | [redacted]      |    25882 | freeze       | do minor freeze         | tenant_id | 1001   |
| 2020-12-15 00:03:25.573175 | [redacted]      |    25882 | freeze       | do minor freeze success | tenant_id | 1001   |
| 2020-12-15 00:03:25.573120 | [redacted]      |    25882 | minor_merge  | minor merge start       | tenant_id | 1001   |
| 2020-12-15 00:03:25.547458 | [redacted]      |    25882 | freeze       | do minor freeze         | tenant_id | 1001   |
| 2020-12-15 00:02:53.230877 | [redacted]      |    25882 | minor_merge  | minor merge finish      | tenant_id | 1001   |
| 2020-12-15 00:02:53.099860 | [redacted]      |    25882 | minor_merge  | minor merge finish      | tenant_id | 1001   |
| 2020-12-15 00:02:44.032599 | [redacted]      |    25882 | minor_merge  | minor merge finish      | tenant_id | 1001   |
| 2020-12-15 00:02:16.485048 | [redacted]      |    25882 | freeze       | do minor freeze success | tenant_id | 1001   |
| 2020-12-15 00:02:16.483594 | [redacted]      |    25882 | minor_merge  | minor merge start       | tenant_id | 1001   |
| 2020-12-15 00:02:16.431847 | [redacted]      |    25882 | freeze       | do minor freeze         | tenant_id | 1001   |
| 2020-12-15 00:02:15.235066 | [redacted]      |    25882 | freeze       | do minor freeze success | tenant_id | 1001   |
| 2020-12-15 00:02:15.234964 | [redacted]      |    25882 | minor_merge  | minor merge start       | tenant_id | 1001   |
| 2020-12-15 00:02:15.200344 | [redacted]      |    25882 | freeze       | do minor freeze         | tenant_id | 1001   |
| 2020-12-15 00:02:14.987789 | [redacted]      |    25882 | freeze       | do minor freeze success | tenant_id | 1001   |
| 2020-12-15 00:02:14.987735 | [redacted]      |    25882 | minor_merge  | minor merge start       | tenant_id | 1001   |
| 2020-12-15 00:02:14.962074 | [redacted]      |    25882 | freeze       | do minor freeze         | tenant_id | 1001   |
+----------------------------+-----------------+----------+--------------+-------------------------+-----------+--------+
```

手工发起的转储，在__all_rootservice_event_history表中可以查到具体的选项：

```
MySQL [oceanbase]> select
    -> gmt_create,
    -> module,
    -> event,
    -> name1,
    -> value1,
    -> name2,
    -> value2,
    -> name3,
    -> value3,
    -> extra_info
    -> from
    -> __all_rootservice_event_history
    -> where
    -> module = 'root_service'
    -> order by
    -> gmt_create desc
    -> limit 50;
*************************** 1. row ***************************
gmt_create: 2020-12-15 00:11:17.750441
    module: root_service
     event: root_minor_freeze
     name1: svr
    value1: 0
     name2: data
    value2: {tenant_ids:[], partition_key:{tid:1100011134503307, partition_id:1, part_num:0}, server_list:[{[redacted]:25882}, {[redacted]:25882}], zone:""}
     name3:
    value3:
extra_info:
*************************** 2. row ***************************
gmt_create: 2020-12-15 00:11:02.738936
    module: root_service
     event: root_minor_freeze
     name1: svr
    value1: 0
     name2: data
    value2: {tenant_ids:[1001, 1002], partition_key:{tid:1040740073780551615, partition_id:1, part_idx:2684555453, subpart_idx:2684555453}, server_list:[{[redacted]:25882}, {[redacted]:25882}], zone:""}
     name3:
    value3:
extra_info:
*************************** 3. row ***************************
gmt_create: 2020-12-15 00:10:21.707624
    module: root_service
     event: root_minor_freeze
     name1: svr
    value1: 0
     name2: data
    value2: {tenant_ids:[], partition_key:{tid:1040627440729550613, partition_id:1, part_idx:2684555453, subpart_idx:2684555453}, server_list:[{[redacted]:25882}], zone:""}
     name3:
    value3:
extra_info:
*************************** 4. row ***************************
gmt_create: 2020-12-15 00:10:01.981636
```

OCEANBASE

2.2.1 转储&合并对比

合并与转储都能将MemStore内存中的数据冻结并写到磁盘上，数据写盘完成后也都能释放MemStore内存，但二者还是有很多不同之处：

合并（Major freeze）	转储（Minor freeze）
集群级行为，所有observer上所有租户的MemStore统一冻结。	以“租户+observer”为维度，每个MemStore独立触发冻结；也可以通过手工命令，为特定的分区单独执行。
MemTable数据和转储数据全部合并到SSTable中，完成后数据只剩一层。	只会和前一次的转储数据做合并，不涉及SSTable数据，完成后有转储和SSTable两层数据。
更新的数据量大（全部租户、全部observer、含SSTable），消耗较多的CPU和IO资源，MemStore内存释放较慢。	更新的数据量小（单独租户、单独observer、不含SSTable），消耗的资源更少，可加快MemStore内存的释放。
触发条件：单个租户的MemStore使用率达到freeze_trigger_precentage，并且转储已经达到指定次数；手工触发；定时触发。	触发条件：单个租户的MemStore使用率达到freeze_trigger_precentage；手工触发。

OCEANBASE


感谢学习
OCEANBASE