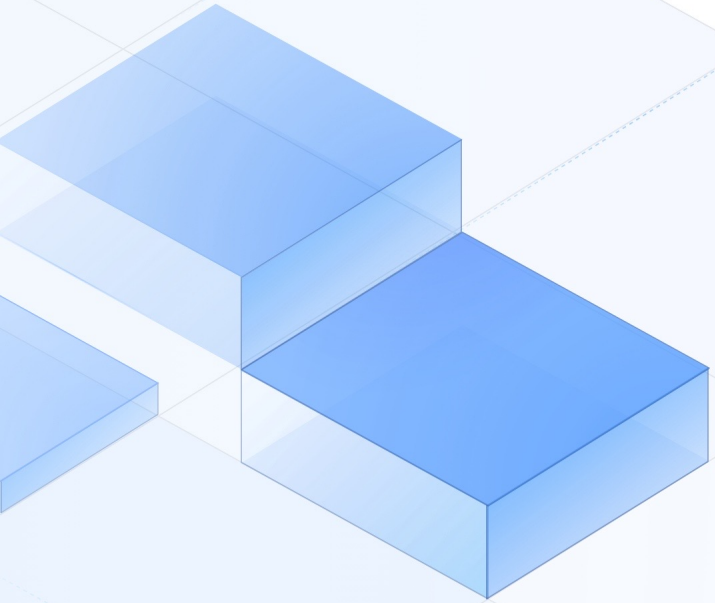
OCEANBASE

OBCP V3.0 培训教材

目录



OCEANBASE

OceanBase存储引擎高级技术

内存管理

内存数据落盘策略-合并和转储

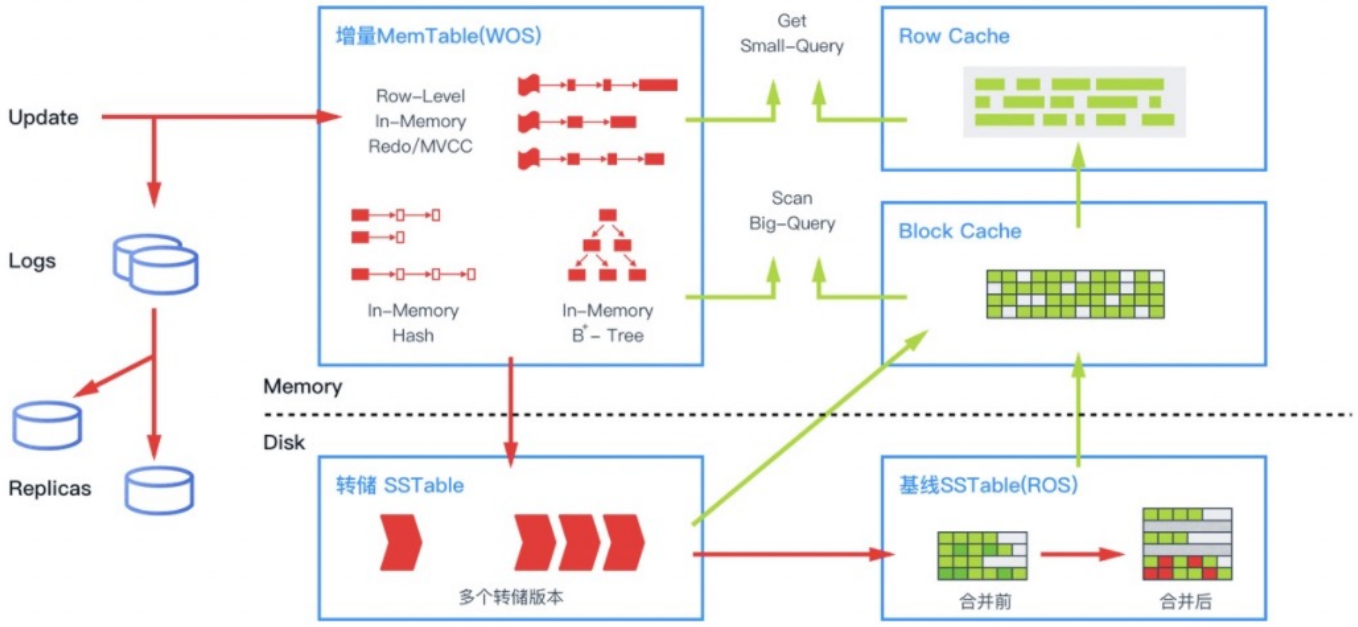
系统架构概念图

OceanBase 数据库的存储引擎基于 LSM Tree 架构

- 静态基线数据（放在 SSTable 中）
- 动态增量数据（放在 MemTable 中）

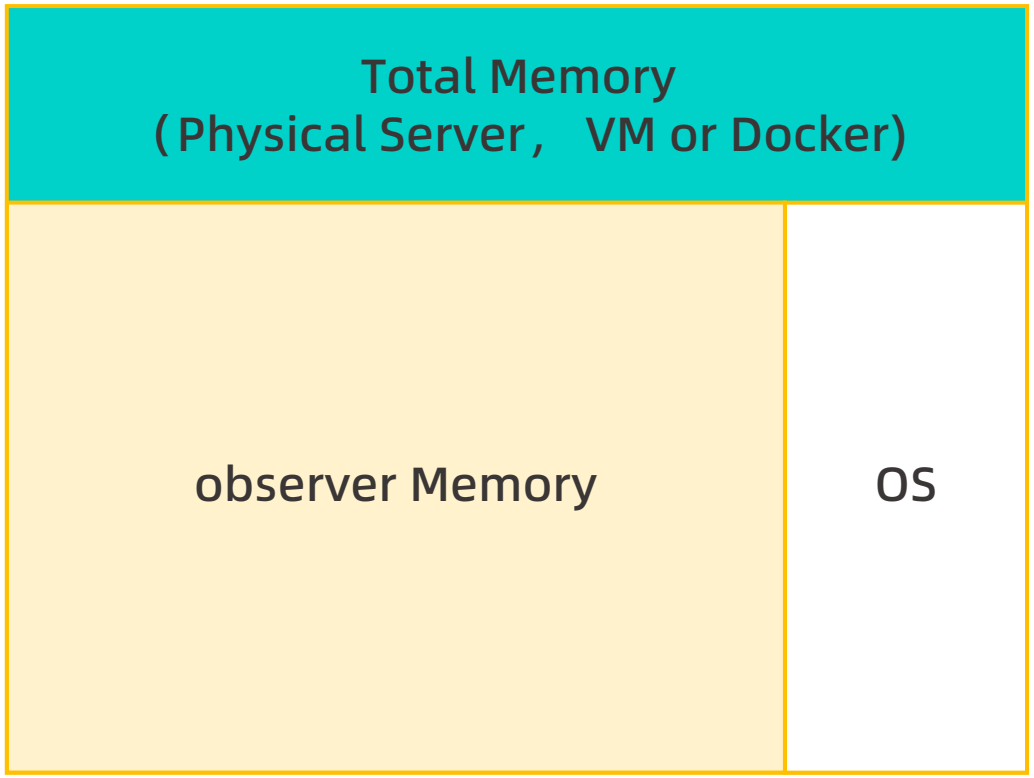
本质上是一个基线加增量的存储引擎，跟关系数据库差别很大，同时也借鉴了部分传统关系数据库存储引擎的优点。

由于 OceanBase 数据库采用基线加增量的设计，一部分数据在基线，一部分在增量，原理上每次查询都是既要读基线，也要读增量。为此，OceanBase 数据库做了很多的优化，尤其是针对单行的优化。



内存分配（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

memory_limit 的默认单位为 MB

例如，memory_limit='40G' 表示设置 OceanBase 数据库进程的使用内存上限是 40 GB。由于默认单位为 MB，则 memory_limit=40960 与 memory_limit='40G' 设置的值相同

如果希望限制运行中的 OceanBase 数据库的内存大小，可以直接修改 memory_limit 的值，使其达到预期。设置后，后台参数 Reload 线程会使其动态生效，无需重启。但是在设置时，需要保证 memory_limit 的值小于系统总的值

参数说明

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

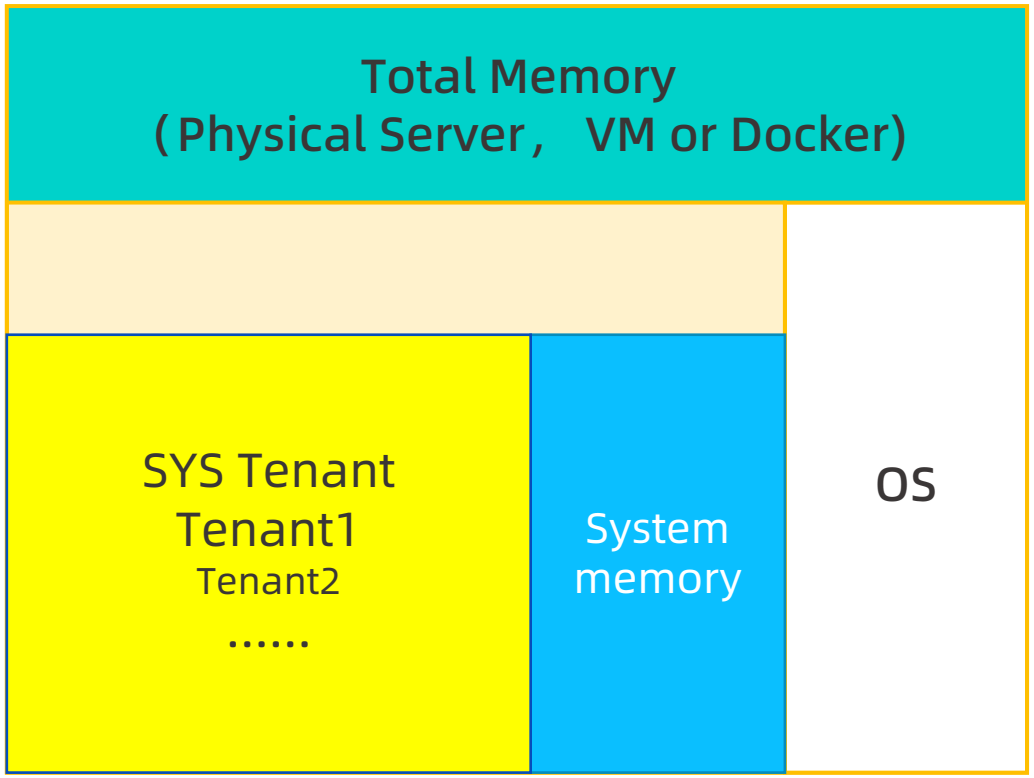
场景1：memory_limit=0，因此由memory_limit_percentage确定observer内存大小，即100GB*80% = 80GB

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

内存分配（二）

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（3.x版本默认值30G）

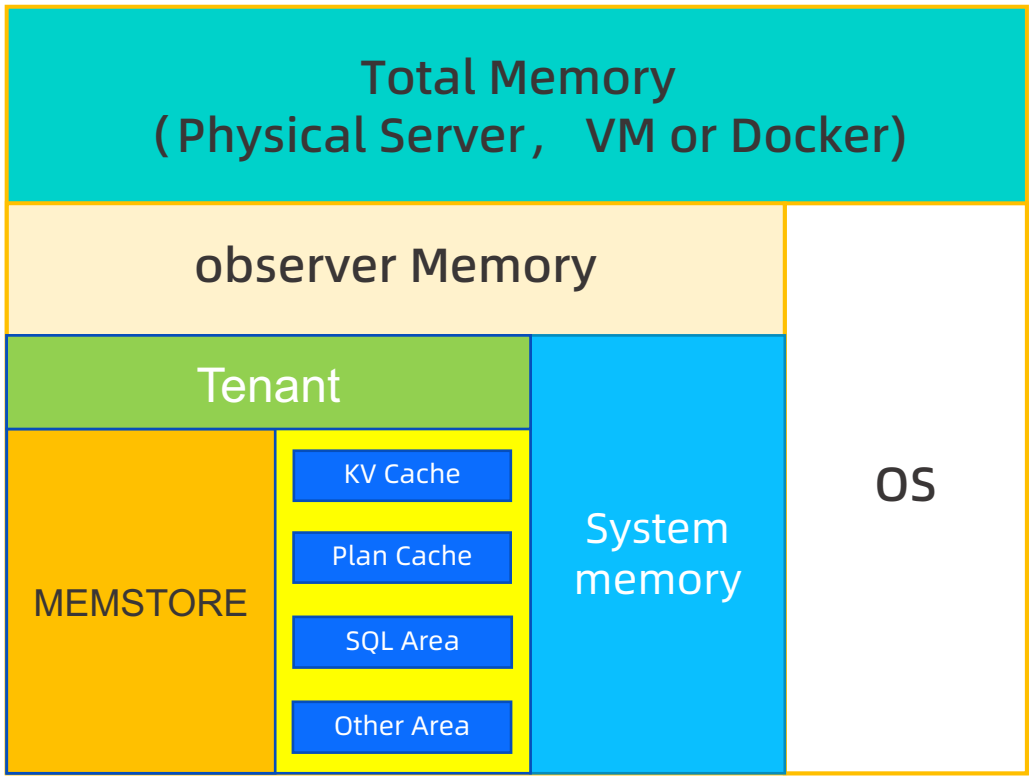
租户可用的总内存

- “observer内存上限”-“系统内部内存”

内存分配（三）

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， MemStore用来保存DML产生的增量数据，空间不可被占用
- 可动态伸缩的内存：KVCache， KVCache空间会被其它众多内存模块复用



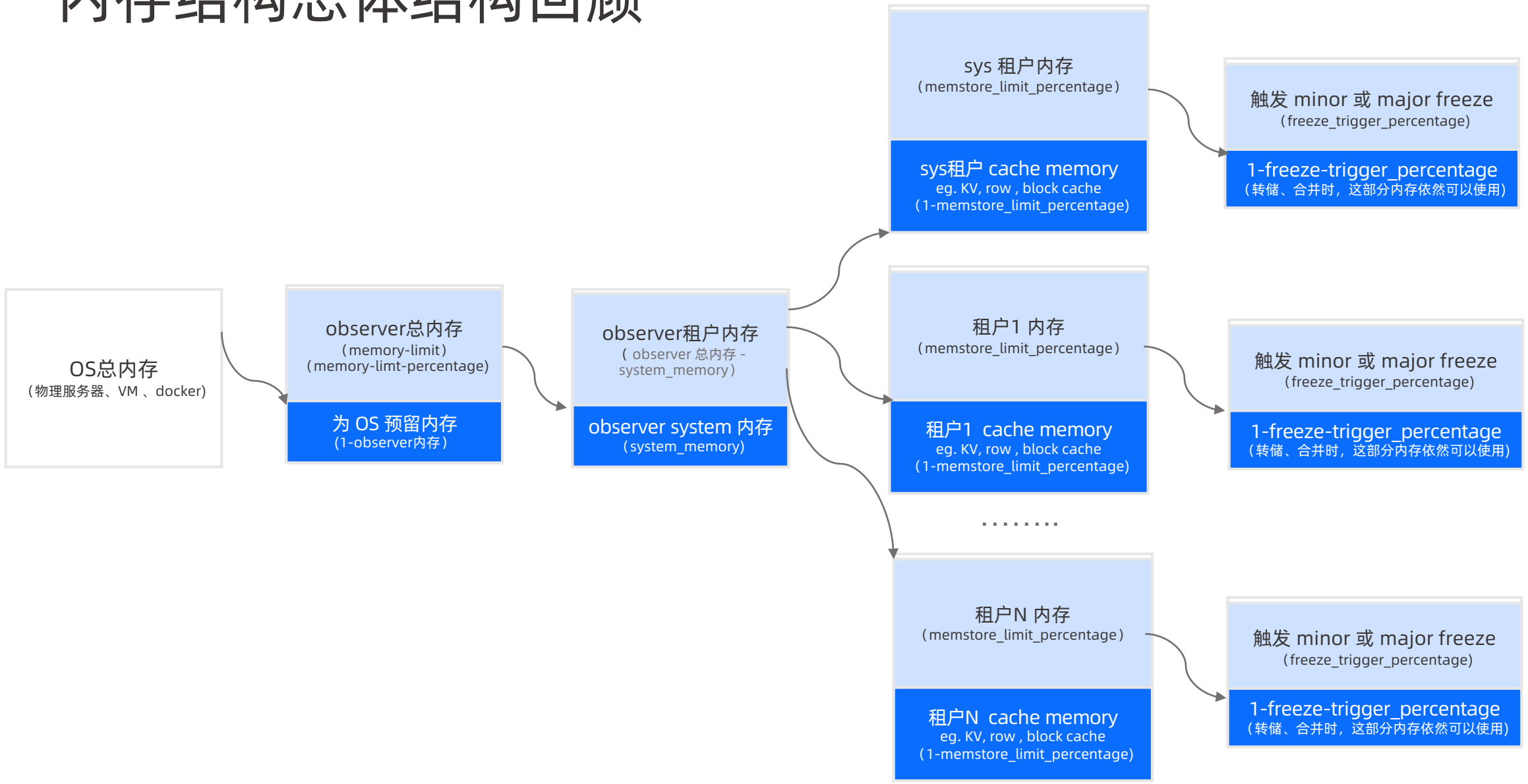
MemStore

- 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（默认70%），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

内存结构总体结构回顾


OS总内存
（物理服务器、VM 、docker）
observer总内存
（memory-limit）
（memory-limt-percentage）
为 OS 预留内存
(1-observer内存）
observer租户内存
（ observer 总内存 - system_memory）
observer system 内存
（system_memory）
sys 租户内存
（memstore_limit_percentage）
sys租户 cache memory
eg. KV, row , block cache
（1-memstore_limit_percentage）
触发 minor 或 major freeze
（freeze_trigger_percentage）
1-freeze-trigger_percentage
（转储、合并时，这部分内存依然可以使用）
租户1 内存
（memstore_limit_percentage）
租户1 cache memory
eg. KV, row , block cache
（1-memstore_limit_percentage）
触发 minor 或 major freeze
（freeze_trigger_percentage）
1-freeze-trigger_percentage
（转储、合并时，这部分内存依然可以使用）
.....
租户N 内存
（memstore_limit_percentage）
租户N cache memory
eg. KV, row , block cache
（1-memstore_limit_percentage）
触发 minor 或 major freeze
（freeze_trigger_percentage）
1-freeze-trigger_percentage
（转储、合并时，这部分内存依然可以使用）

OCEANBASE

OceanBase存储引擎高级技术

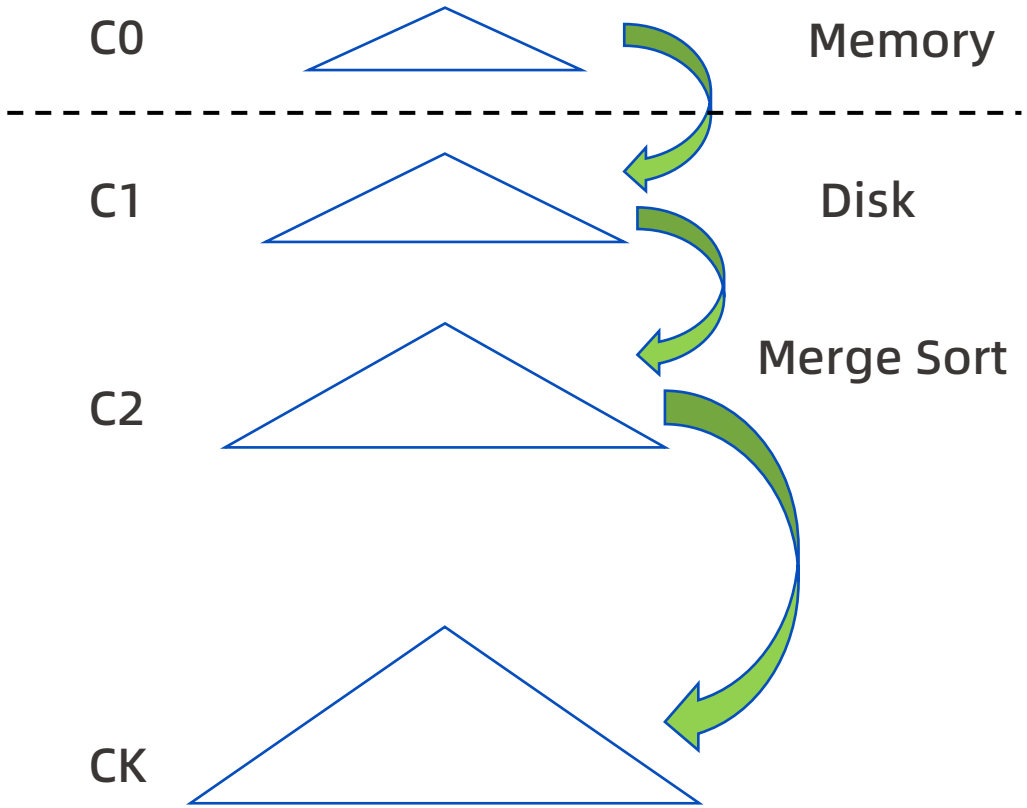
内存管理

内存数据落盘策略-合并和转储

LSM Tree 技术简介

LSM Tree（The Log-Structured Merge-Tree）核心特点是利用顺序写来提高写性能

- 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



memtable内存结构

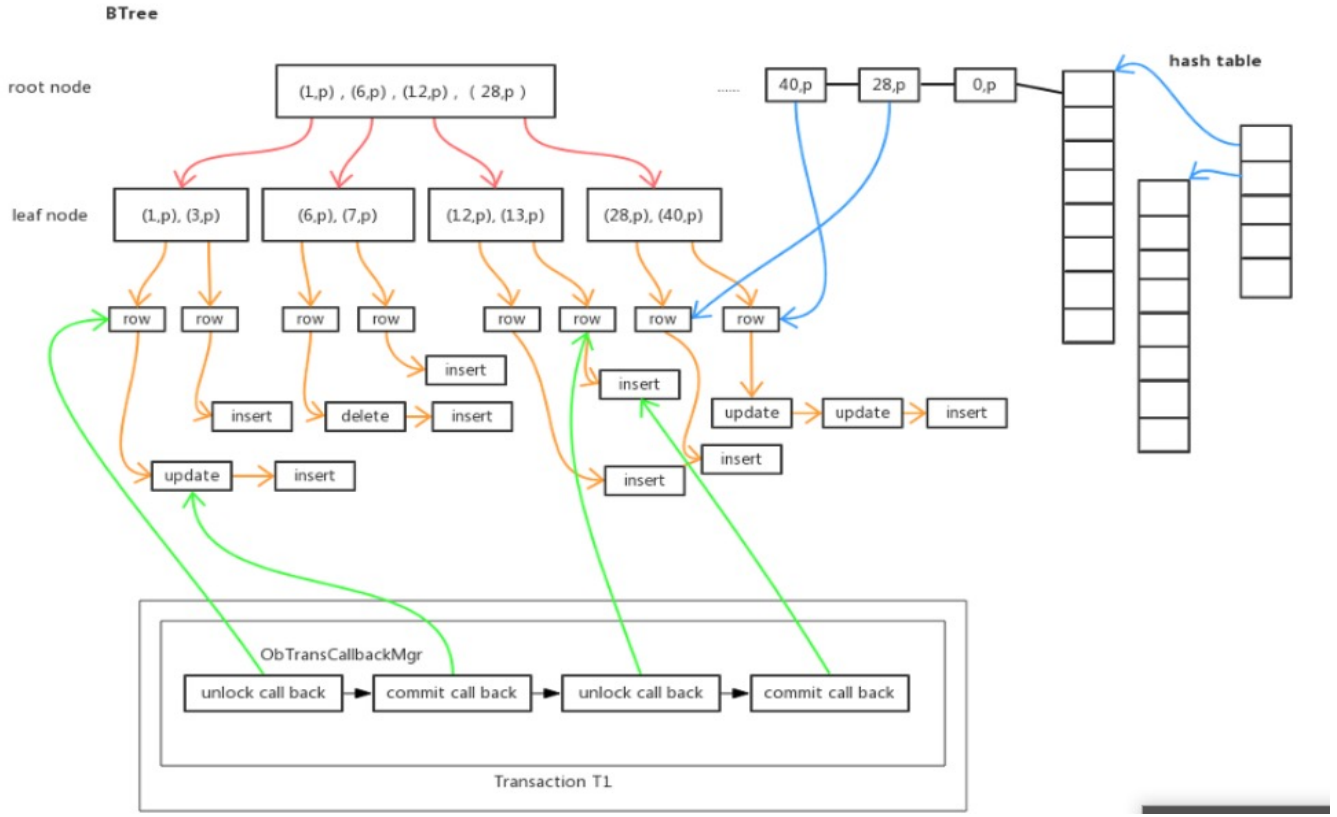
OceanBase MemTable采用双索引结构（B+树索引以及哈希索引）

特点：

- B+树索引能够更好地支持范围查找
- 哈希索引是针对单行查找的一种优化
- 每次事务执行时，MemTable会自动维护B+树索引与哈希索引之间的一致性

Undo流程：

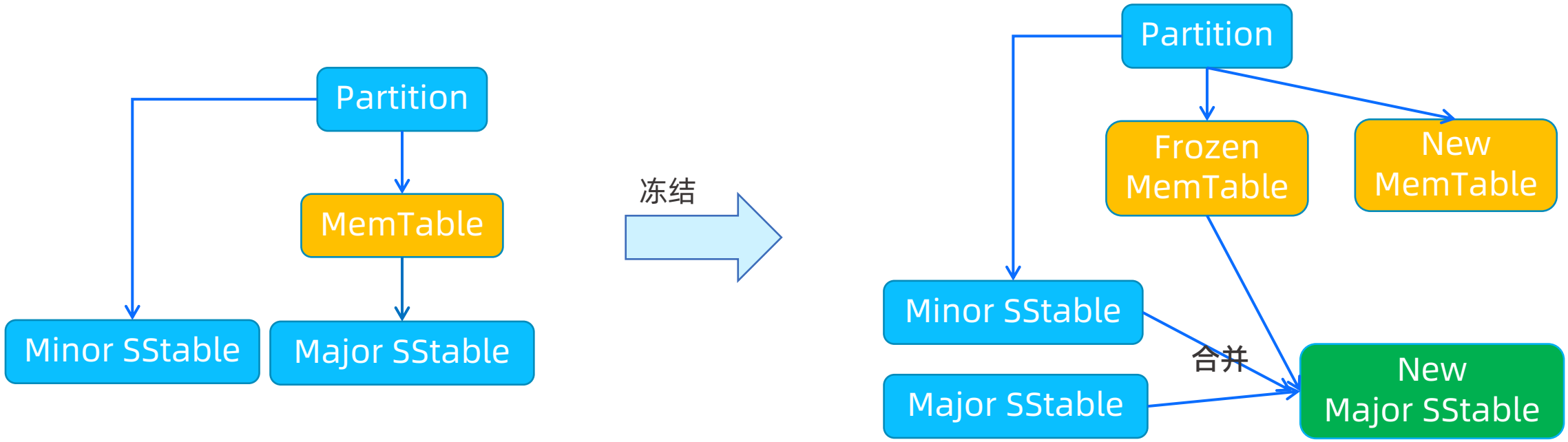
如果要读取更老的历史快照，只需要顺着内存中的反向指针往前回溯即可，相当于在内存中执行数据库Undo操作



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：

1. 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
2. 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
3. 合并完成后，冻结的MemTable内存才可以被清空并重新使用



合并的细化

合并按照合并的宏块的不同，可以细化为全量合并、增量合并，渐进合并三种方式：

- 全量合并 ： 合并时间长，耗费IO和CPU。把所有的静态数据都读取出来，和动态数据归并，再写到磁盘中
- 增量合并：只会读取被修改过的宏块数据，和动态数据归并，并写入磁盘，对于未修改过的宏块，则直接重用
- 渐进合并：每次全量合并一部分，若干轮次后整体数据被重写一遍

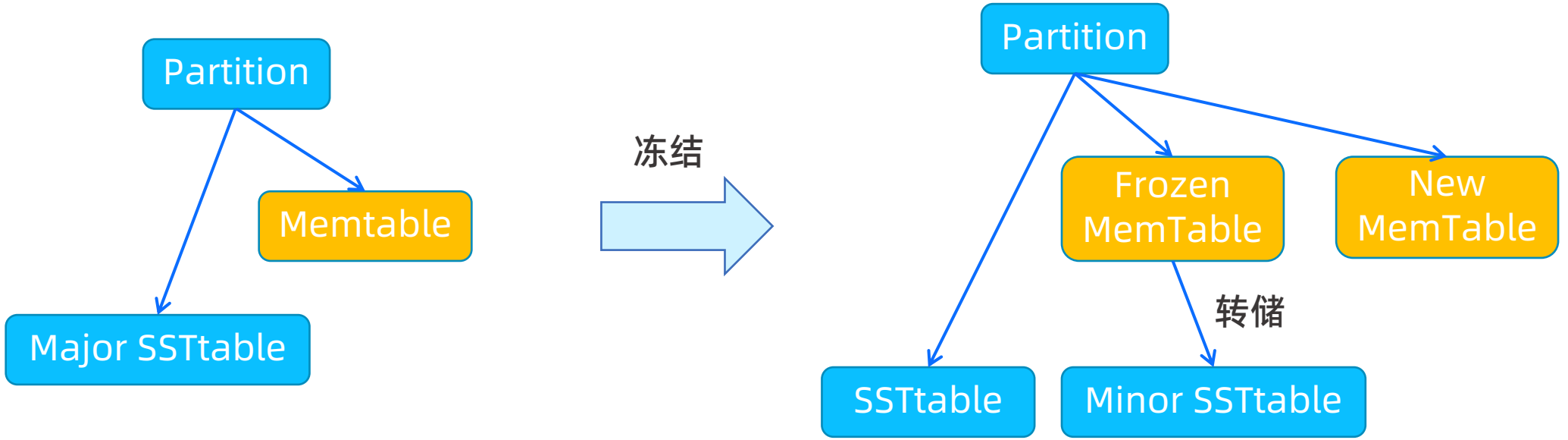
思考：

1. 对业务侧而言，合并存在哪些问题？
   - 集群性
   - 高消耗
   - 时间长
2. 如何避免？
   - 转储

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），OB引入了“转储”机制（C1层）：

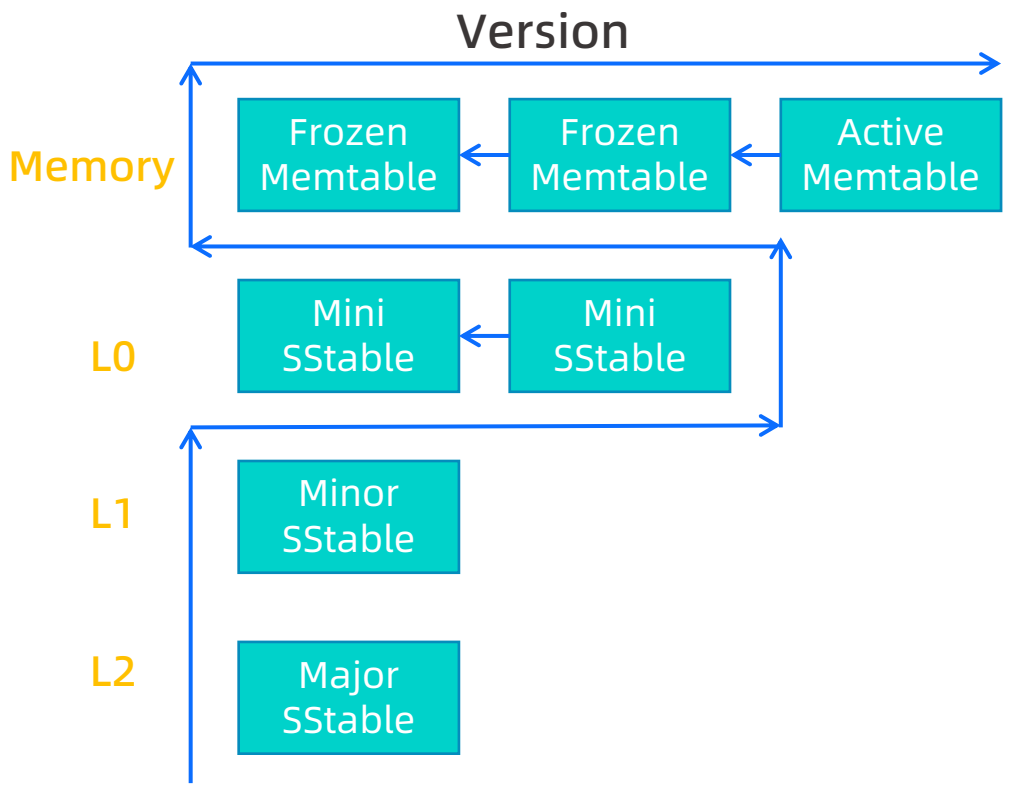
- 将MemTable数据做小版本冻结（Minor Freeze）后写到磁盘上单独的转储文件里，不与SSTable数据做合并
- 转储文件写完之后，冻结的MemTable内存被清空并重新使用
- 每次转储会将MemTable数据与前一次转储的数据合并（Merge），转储文件最终会合并到SSTable中



分层转储

为了优化转储越来越慢的问题，引入了“分层转储”机制：

- 多层compaction策略：新增 L0 层：被冻结的 MemTable 会直接 flush 为 Mini SSTable，可同时存在多个Mini SSTable。
- 架构变化: 3层 Vs 4层
  - 3层架构：memtable + minor sstable(L1) + major sstable (L2)
  - 4层架构：memtable + mini sstable(L0) + minor sstable(L1) + major sstable (L2)
- 参数 minor_compact_trigger 控制L0层Mini SSTable 总数
- 参数 major_compact_trigger 控制memtable dump flush次数达到时触发major compaction



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

转储相关参数

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

转储适用的场景

转储功能比较适用于以下场景

1. 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
2. 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

1. 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
2. 增大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）

转储对数据库的影响

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

手动触发转储

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

- 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）

查看转储记录

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 00:02:16.485040 | [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   |
```

手动转储，在__all_rootservice_event_history表中可以查到具体的选项

```
MySQL [oceanbase]>
MySQL [oceanbase]> select
    ->   gmt_create,
    ->   module,
    ->   event,
    ->   name1,
    ->   value1,
    ->   name2,
    ->   value2,
    ->   name3,
    ->   value3,
    ->   extra_info
    -> from
    ->   __all_rootservice_event_history
    -> where
    ->   event like '%minor%'
    -> order by
    ->   gmt_create desc
    -> limit 50 \G
*************************** 1. row ***************************
gmt_create: 2020-12-15 00:11:17.736441
    module: root_service
     event: root_minor_freeze
     name1: ret
    value1: 0
     name2: arg
    value2: {tenant_ids:[], partition_key:{tid:1100611139453887, partition_id:1, part_cnt:0}, server_list:["[redacted]:25882", "[redacted]:25882"], zone:""}
     name3:
    value3:
extra_info:
*************************** 2. row ***************************
gmt_create: 2020-12-15 00:11:02.730996
    module: root_service
     event: root_minor_freeze
     name1: ret
    value1: 0
     name2: arg
    value2: {tenant_ids:[1001, 1002], partition_key:{tid:1844674407370955161 5, partition_id:-1, part_idx:268435455, subpart_idx:268435455}, server_list:["[redacted]:25882", "[redacted]:25882"], zone:""}
     name3:
    value3:
extra_info:
*************************** 3. row ***************************
gmt_create: 2020-12-15 00:10:21.745924
    module: root_service
     event: root_minor_freeze
     name1: ret
    value1: 0
     name2: arg
    value2: {tenant_ids:[], partition_key:{tid:1844674407370955161 5, partition_id:-1, part_idx:268435455, subpart_idx:268435455}, server_list:["[redacted]:25882", "[redacted]:25882"], zone:""}
     name3:
    value3:
extra_info:
*************************** 4. row ***************************
gmt_create: 2020-12-15 00:10:03.902636
```

OB合并触发方式-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:
```

OB合并触发方式-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$memstore视图来查看各租户的memstore内存使用情况
- 查转储次数：gv$memstore, __all_virtual_tenant_memstore_info 中 freeze_cnt 列

```
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 |
+-----------+-----------+-----------+---------------------+---------------------+
```

OB合并触发方式-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)
```

查看 OB 集群合并和冻结状态 __all_zone

全局信息		Zone相关信息		合并状态	
frozen_time	冻结时间	all_merged_version	zone最近一次已经合并完成的版本	IDLE	未合并
frozen_version	版本号，从1开始	broadcast_version	本zone收到的可以进行合并的版本	TIMEOUT	合并超时
global_broadcast_version	这个版本已经通过各ObServer进行合并。	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	MERGING	正在合并
is_merge_error	合并过程中是否出现错误	merge_start_time / last_merged_time	合并开始、结束时间	ERROR	合并出错
last_merged_version	表示上次合并完成的版本	merge_status	合并状态		
try_frozen_version	正在进行哪个版本的冻结				
merge_list	Zone的合并顺序				

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

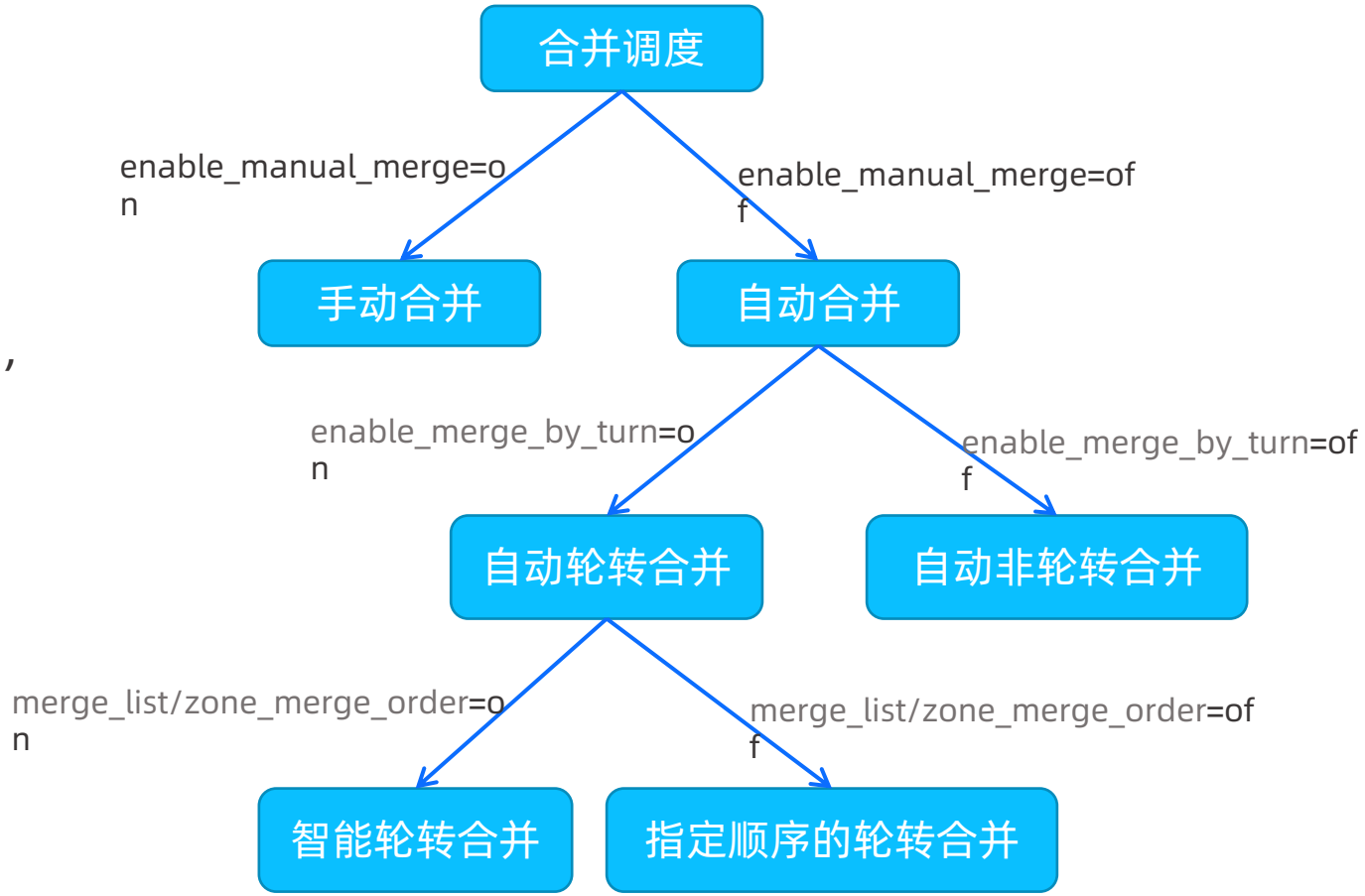
关于轮转合并的更多说明

- 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每日合并策略

可通过以下几项控制每日合并的策略

- enable_manual_merge：OB的配置项，指示是否开启手动合并
- enable_merge_by_turn：OB的配置项，指示是否开启自动轮转合并
- zone_merge_order：指定自动轮转合并的合并顺序



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
```

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

合并策略总结对比

合并策略	调度策略	如何开启	使用场景	注释
手动合并	用户通过sql命令指定zone开始合并，需要用户自己控制并发度	1.开启手动合并alter system set enable_manual_merge = true; 2.用户自主决定合并顺序和并发度，通过SQL命令调度zone合并，比如调度z1开始合并:alter system start merge zone = 'z1';	纯手工操作，一般在业务每日合并出现问题、需要人工介入的情况下使用	一旦开启，每一次合并都需要用户主动调度，除非关掉手动合并，开启自动合并
自动非轮转合并	所有zone一起开始合并，没有并发度控制	1.关闭手动合并alter system set enable_manual_merge = false; 2.关闭轮转合并alter system set enable_merge_by_turn = false;	当业务量比较小的情况下，合并中的zone也能支持业务流量时，则可以开启自动非轮转合并，这样做能够避免用户跨表join请求变成分布式跨机查询	每日合并会对业务请求产生一定的性能影响，需要业务进行确认
自动指定顺序的轮转合并	用户直接指定轮转的顺序，RS只负责并发度控制	各个版本实现有不同，具体看相关版本介绍	一种特殊的轮转合并策略，一般不会使用，只有当智能轮转合并不满足业务需求的情况下，才需要为集群定制特殊的合并调度策略	在zone成员发生变更的情况下，自动指定顺序的轮转合并会失效，退化成智能轮转合并
智能轮转合并	RS根据一定策略依次调度zone合并,并进行并发度控制	1.关闭手动合并alter system set enable_manual_merge = false; 2.开启轮转合并alter system set enable_merge_by_turn = true;	线上部署最常用的合并调度方案，轮转合并的过程中，RS会保证尽量不影响业务的请求，通过切leader的方式，将用户读写路由到不在合并的zone中;	

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

- 参数data_disk_usage_limit_percentage定义数据文件最大可以写入的百分比（超出阈值后禁止数据迁入），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
- 参数datafile_disk_percentage定义数据盘空间使用阈值（占用data_dir所在磁盘总空间百分比），默认值90
- 参数datafile_size 用于设置数据文件的大小，该配置项与 datafile_disk_percentage 同时配置时，以该配置项设置的值为准，默认值为0

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：表示自适应，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 。

合并版本

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。

```
MySQL [oceanbase]> select zone, svr_ip, major_version
    -> from __all_virtual_partition_sstable_image_info
    -> order by 1,2,3;
+-------+---------------+---------------+
| zone  | svr_ip        | major_version |
+-------+---------------+---------------+
| zone1 | [redacted]    |            29 |
| zone1 | [redacted]    |            30 |
| zone2 | [redacted]    |            29 |
| zone2 | [redacted]    |            30 |
| zone3 | [redacted]    |            29 |
| zone3 | [redacted]    |            30 |
+-------+---------------+---------------+
6 rows in set (0.17 sec)
```

```
mysql> select /*+ frozen_version(29) */ * from tmp1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
+----+------+
3 rows in set (0.01 sec)

mysql> select /*+ frozen_version(30) */ * from tmp1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |  101 |
|  2 |  202 |
|  3 |  303 |
+----+------+
3 rows in set (0.01 sec)
```

```
mysql> select * from tmp1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |  101 |
|  2 |  202 |
|  3 |  303 |
+----+------+
3 rows in set (0.01 sec)
```

查看合并记录和状态

通过__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

通过__all_zone表，查看当前合并状态：

查看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

当前状态未在合并

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

转储&合并对比

合并（Major freeze）	转储（Minor freeze）
集群级行为，产生一个全局快照，所有observer上所有租户的MemStore统一冻结。	以“租户+observer”为维度，只是MemTable的物化，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，产生新的全量数据。	转储只与相同大版本的Minor SSTable合并，产生新的Minor SSTable，所以只包含增量数据，最终被删除的行需要特殊标记，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。

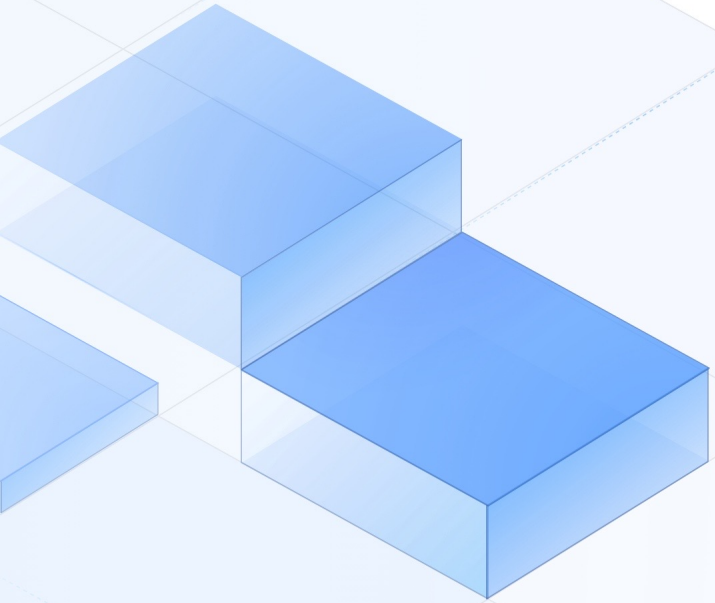
小结

- OB的LSMTree可以分为C0层（MemTable）、C1层（Minor SSTable）、C2层（Major SSTable）
- OB内存通过双索引结构和数据压缩，提高数据的查询性能
- 合并和转储之前，都需要做一次冻结，然后根据参数设置决定冻结之后是转储还是合并
- 合并可以细分为全量合并、渐进合并、增量合并三种方式，同一个数据库，这三种方式对资源的消耗程度递减
- 为了优化转储越来越慢的问题，引入了“分层转储”机制，为了提高转储速度，加快内存释放速度，被冻结的 MemTable 会直接 flush 为 Mini SSTable
- 轮转合并可以轮流为每份副本单独做合并，减少业务影响，但同时也存在合并时间变长、切主过程中影响长连接等问题
- 合并和转储特点的比较，两者互补共同组成了OB数据完整的落盘策略

思考

- 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？

感谢学习



OCEANBASE