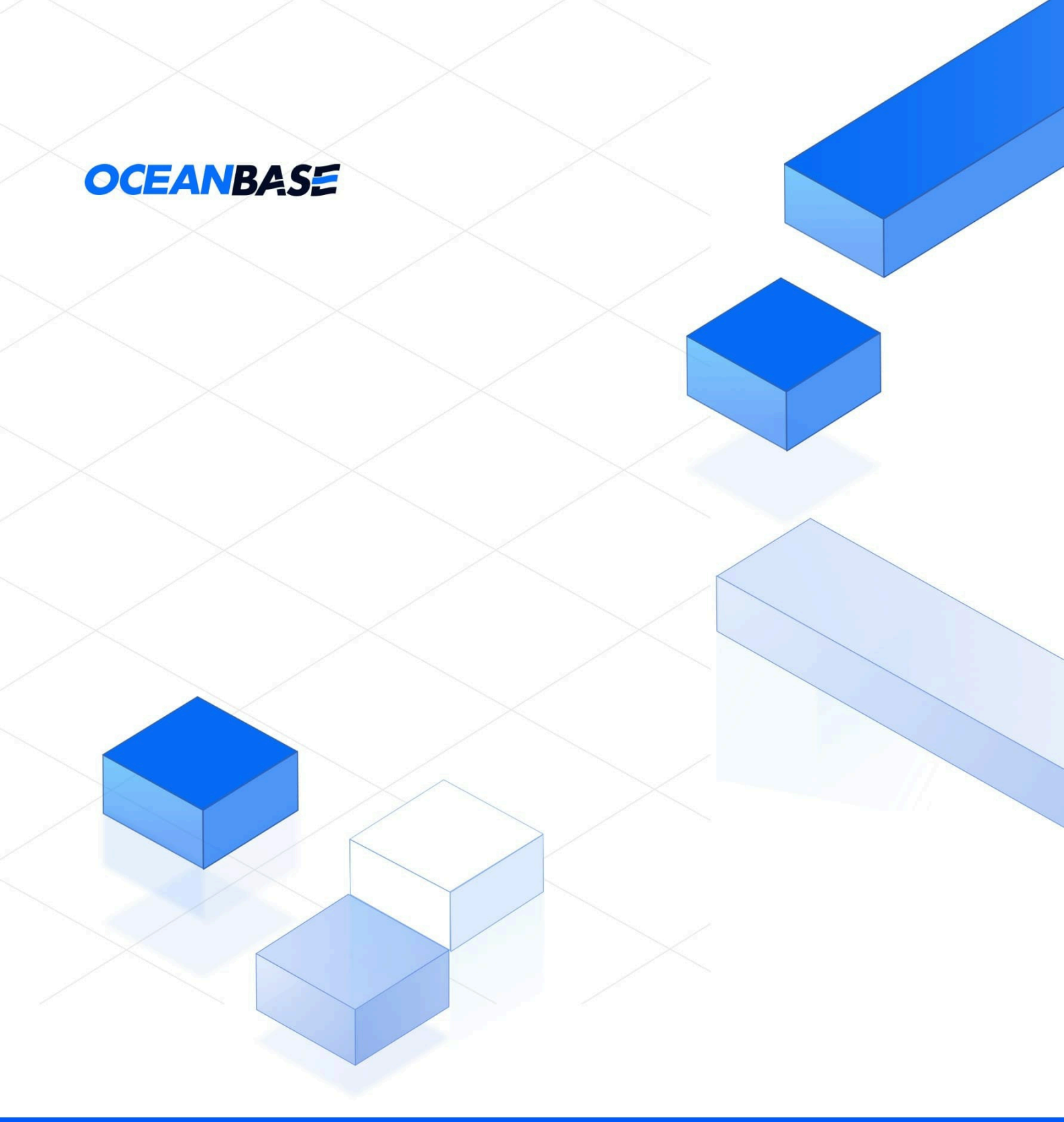


OceanBase 数据库

参考指南--系统管理

| 产品版本：V4.5.0 | 文档版本：20251219

声明

北京奥星贝斯科技有限公司版权所有©2024，并保留一切权利。

商标声明

免责声明

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	**危险** 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	**警告** 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	**注意** 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	**说明** 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1 配置管理介绍

OceanBase 数据库通过设置配置项和系统变量来完成配置管理。

1.1 配置项

配置项主要用于运维，常用于控制机器及其以上级别的系统行为。可以作为节点启动参数和租户创建参数，也可以在节点运行时进行修改以调整系统行为。 有关配置项的详细介绍请参见设置参数。

1.2 系统变量

系统变量通常和用户 Session 绑定，用于控制 Session 级别的 SQL 行为。

支持设置 Global 和 Session 级别的变量。设置 Global 级别的系统变量后，当前 Session 上不会生效，新建的任何 Session 都能读到新的变量值。设置 Session 级别的系统变量后，仅对当前 Session 生效。

有关系统变量的详细介绍请参见 设置变量。

1.3 系统配置项和系统变量对比

对比项	系统配置项	系统变量
生效范围	集群、租户、Zone、机器。	租户的 Global 级别或 Session 级别。
生效方式	• 动态生效：edit_level 为 dynamic_effective。 • 重启生效：edit_level 为 static_effective。	• 设置 Session 级别的变量仅对当前 Session 有效，对其他 Session 无效。 • 设置 Global 级别的变量对当前 Session 无效，需要重新登录建立新的 Session 才会生效。

修改方式	• 支持通过 SQL 语句修改，示例如下：
	`obclient> Alter SYSTEM SET schema_history_expire_time='1h';`
	• 支持通过启动参数修改，示例如下：
	`cd /home/admin && ./bin/observer -o "schema_history_expire_time='1h'"`
		仅支持通过 SQL 语句修改，示例如下：
		• MySQL 模式
		`obclient> SET ob_query_timeout = 20000000;`
		`obclient> SET GLOBAL ob_query_timeout = 20000000;`
		• Oracle 模式
		`obclient> SET ob_query_timeout = 20000000;`
		`obclient> SET GLOBAL ob_query_timeout = 20000000;`
		`obclient> ALTER SESSION SET ob_query_timeout = 20000000;`
		`obclient> ALTER SYSTEM SET ob_query_timeout = 20000000;`

持久化	持久化到内部表与配置文件，可以在 `/home/admin/oceanbase/etc/observer.config.bin` 与 `/home/admin/oceanbase/etc/observer.config.bin.history` 文件中查询该配置项。	仅 GLOBAL 级别的变量会持久化，SESSION 级别的变量不会进行持久化。
生命周期	长，从进程启动到退出。	短，需要租户的 Schema 创建成功以后才生效。
		可以使用 `SHOW [GLOBAL] VARIABLES` 或 `SELECT` 语句查询。 示例如下：

查询方式	可以使用 `SHOW PARAMETERS` 语句查询。示例如下：
	`obclient> SHOW PARAMETERS LIKE 'schema_history_expire_time';`
		• MySQL 模式
		`obclient> SHOW VARIABLES LIKE 'ob_query_timeout';`
		`obclient> SHOW GLOBAL VARIABLES LIKE 'ob_query_timeout';`
		`obclient> SELECT * FROM INFORMATION_SCHEMA.SESSION_VARIABLES WHERE VARIABLE_NAME = 'ob_query_timeout';`
		`obclient> SELECT * FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES WHERE VARIABLE_NAME = 'ob_query_timeout';`
		• Oracle 模式
		`obclient> SHOW VARIABLES LIKE 'ob_query_timeout';`
		`obclient> SHOW GLOBAL VARIABLES LIKE 'ob_query_timeout';`
		`obclient> SELECT * FROM SYS.TENANT_VIRTUAL_GLOBAL_VARIABLE WHERE VARIABLE_NAME = 'ob_query_timeout';`
		`obclient> SELECT * FROM SYS.TENANT_VIRTUAL_SESSION_VARIABLE WHERE VARIABLE_NAME = 'ob_query_timeout';`

2 设置参数

OceanBase 数据库的配置项分为集群级配置项和租户级配置项。您可以通过配置项的设置使 OceanBase 数据库的行为符合您业务的要求。

2.1 参数分类

OceanBase 数据库的配置项分为集群级配置项和租户级配置项。通过配置项的设置可以控制整个集群的负载均衡、合并时间、合并方式、资源分配和模块开关等功能。

- 集群级配置项：作用范围为整个集群所有 OBServer。
- 租户级配置项：作用范围为当前租户在集群内所在的 OBServer。

名称以 "_" 开头的配置项称为隐藏配置项，如：`_ob_max_thread_num`。仅供开发人员在故障排查或紧急运维时使用。本文不对隐藏配置项进行详细介绍，下文中的配置项均不包含隐藏配置项。

不同租户对配置项的查看和修改级别如下表所示：

租户类型	参数查看	参数设置
系统租户	集群级配置项、租户级配置项 #### 2.1.0.1 说明 `SHOW PARAMETERS` 语句通过指定 `TENANT` 关键字来查看指定租户的配置项信息。	集群级配置项、租户级配置项 #### 2.1.0.2 说明 系统租户下，可以通过指定 `TENANT` 关键字来修改全部或指定租户的租户级配置项。
普通租户	集群级配置项和本租户的租户级配置项	本租户的租户级配置项

当前 OceanBase 数据库中配置项的主要数据类型及其相关说明如下表所示：

数据类型	说明
BOOL	boolean 类型（布尔），支持 true/false。
CAPACITY	容量单位，支持 b （字节）、k （KB，千字节）、m （MB，百万字节）、 g （GB，10亿字节）、t （TB，万亿字节）、p（PB，千万亿字节）。单位不区分大小写字母，默认为 m。
DOUBLE	双精度浮点数，占用 64 bit 存储空间，精确到小数点后 15 位，有效位数为 16 位。
INT	int64 整型，支持正负整数和 0。
MOMENT	时刻。格式为 hh:mm（例如 02:00）；或者特殊值 disable，表示不指定时间。目前仅用于 major_freeze_duty_time 参数。
STRING	字符串。用户输入的字符串的值。
STRING_LIST	字符串列表，即以分号 （;）分隔的多个字符串。
TIME	时间类型。支持 us （微秒）、ms （毫秒）、s （秒），m （分钟）、h （小时）、d （天）等单位。如果不加后缀，默认为秒（s）。单位不区分大小写字母。

2.2 修改配置项

MySQL 模式
Oracle 模式
MySQL 模式修改配置项的 SQL 语法如下：

```
ALTER SYSTEM [SET]
parameter_name = expression [SCOPE = {SPFILE | BOTH}] [COMMENT [=] 'text']
[ TENANT [=] ALL|all_user|all_meta|tenant_name ] {SERVER [=] 'ip:port' | ZONE [=]
'zone'};
```

参数说明：

- parameter_name ： 指定要修改的配置项名称。
- expression ： 指定修改后的配置项的值。
- COMMENT 'text' ： 用于添加关于本次修改的注释。该参数为可选，建议不要省略。

- SCOPE：指定本次配置项修改的生效范围，默认为 BOTH 。
  - SPFILE：表示只修改内部表中的配置项值，当 OBServer 节点重启以后才生效。仅重启生效的配置项支持。
  - BOTH：表示既修改内部表中的配置项值，又修改内存中的配置项值，修改立即生效，且 OBServer 节点重启以后配置值仍然生效。
- TENANT：用于在系统租户下，修改所有或指定租户的租户级配置项。
  - ALL：从 V4.2.1 版本开始，tenant = all 和 tenant = all_user 具有相同的语义，都表示对所有用户租户生效。all 语法即将被废除，不再推荐使用。
  - all_user：对所有用户租户生效。
  - all_meta：对所有 META 租户生效。
  - tenant_name：指定租户。
- SERVER：只修改指定 Server 实例的某个配置项。
- ZONE：表明本配置项的修改针对指定集群的特定 Server 类型，否则，针对所有集群的特定 Server 类型。

2.2.0.1 说明

- 同时修改多个配置项时，以逗号（,）分隔。
- 集群级配置项不能通过普通租户设置，也不可以通过系统租户（即 sys 租户） 指定为普通租户设置，仅支持在系统租户下配置。例如，执行 ALTER SYSTEM SET memory_limit='100G' TENANT='test_tenant' 语句将导致报错，因为 memory_limit 是集群级配置项。
- 租户级配置项可以直接在本租户下修改，也可以在系统租户下通过指定 TENANT 关键字来修改。
- ALTER SYSTEM 语句不能同时指定 Zone 和 OBServer。并且在指定 Zone 时，仅支持指定一个 Zone；指定 OBServer 时，仅支持指定一个 OBServer。如果修改时，不指定 Zone 也不指定 OBServer，则集群级配置项会在整个集群所有 OBServer 上生效；租户级配置项会在当前租户在集群内所在的 OBServer 上生效。
- 确认一个配置项为集群级别还是租户级别，可根据 SHOW PARAMETERS 语句执行结果中的 scope 列对应的值来判断：

- scope 值为 CLUSTER 则表示为集群级配置项。
- scope 值为 TENANT 则表示为租户级配置项。

修改 log_disk_utilization_threshold 配置项的示例如下：

```
obclient> ALTER SYSTEM SET log_disk_utilization_threshold = 20;
```

修改指定 Zone 的 log_disk_utilization_threshold 配置项的示例如下：

```
obclient> ALTER SYSTEM SET log_disk_utilization_threshold = 20 ZONE='z1';
```

修改指定 OBServer 的 log_disk_utilization_threshold 配置项的示例如下：

```
obclient> ALTER SYSTEM SET log_disk_utilization_threshold = 20 SERVER='XXX.XXX.
XXX.XXX:XXXXX';
```

在系统租户下，修改所有或指定租户的租户级配置项。示例如下：

```
obclient> ALTER SYSTEM SET log_disk_utilization_threshold = 20 TENANT='ALL';


obclient> ALTER SYSTEM SET log_disk_utilization_threshold = 20 TENANT='Oracle';
```

2.2.0.2 说明

执行成功以后，所有被指定租户的配置项均会被修改。

Oracle 模式修改配置项的 SQL 语法如下：

```
ALTER SYSTEM SET parameter_name = expression
```

参数说明：

- parameter_name：指定要修改的配置项名称。
- expression：指定修改后的配置项的值。

2.2.0.3 说明

- 同时修改多个配置项时，以逗号（,）分隔。
- Oracle 模式下，只能配置租户级配置项。

- 租户级配置项可以直接在本租户下修改，也可以在系统租户下通过指定 TENANT 关键字来修改。

修改 log_disk_utilization_threshold 配置项的示例如下：

```
obclient> ALTER SYSTEM SET log_disk_utilization_threshold = 20;
```

2.3 重置配置项

MySQL 模式
Oracle 模式
MySQL 模式重置配置项的 SQL 语法如下：

```
ALTER SYSTEM [RESET]
parameter_name [SCOPE = {MEMORY | SPFILE | BOTH}] {TENANT [=] 'tenant_name'};
```

参数说明：

- RESET 用于清除当前参数的值并使用参数的默认值/启动值。
- SCOPE：指定本次配置项修改的生效范围，默认为 BOTH。
  - MEMORY：表示仅重置内存中的配置项值，修改后立即生效，且本次修改在 OBServer 节点重启以后会失效（当前暂无配置项支持该方式）。
  - SPFILE：表示只重置内部表中的配置项值，当 OBServer 节点重启以后才生效。仅重启生效的配置项支持。
  - BOTH：表示既重置内部表中的配置项值，又重置内存中的配置项值，重置立即生效，且 OBServer 节点重启以后配置值仍然生效。
- TENANT：用于在系统租户下，重置所有或指定租户的租户级配置项。
  - tenant_name：指定租户。

2.3.0.1 说明

- 同时重置多个配置项时，以逗号（,）分隔。

- 集群级配置项不能通过普通租户重置，也不可以通过系统租户（即 sys 租户） 指定为普通租户重置，仅支持在系统租户下重置。例如，执行 ALTER SYSTEM RESET memory_limit TENANT='test_tenant' 语句将导致报错，因为 memory_limit 是集群级配置项。
- 租户级配置项可以直接在本租户下修改，也可以在系统租户下通过指定 TENANT 关键字来修改。
- ALTER SYSTEM RESET 语句不支持指定 Zone 或 OBServer。
- 确认一个配置项为集群级别还是租户级别，可根据 SHOW PARAMETERS 语句执行结果中的 scope 列对应的值来判断：
  - scope 值为 CLUSTER 则表示为集群级配置项。
  - scope 值为 TENANT 则表示为租户级配置项。

重置 log_disk_utilization_threshold 配置项的示例如下：

```
obclient> ALTER SYSTEM RESET log_disk_utilization_threshold;
```

在系统租户下，重置所有或指定租户的租户级配置项。示例如下：

```
obclient> ALTER SYSTEM RESET log_disk_utilization_threshold TENANT='ALL';
obclient> ALTER SYSTEM RESET log_disk_utilization_threshold TENANT='Oracle';
```

2.3.0.2 说明

执行成功以后，所有被指定租户的配置项均会被修改。

Oracle 模式修改配置项的 SQL 语法如下：

```
ALTER SYSTEM [RESET] parameter_name = expression;
```

参数说明：

- RESET 用于清除当前参数的值并使用参数的默认值/启动值。
- parameter_name：指定要重置的配置项名称。
- expression：指定重置配置项。

2.3.0.3 说明

- 同时重置多个配置项时，以逗号（,）分隔。
- Oracle 模式下，只能重置租户级配置项。
- 租户级配置项可以直接在本租户下重置，也可以在系统租户下通过指定 `TENANT` 关键字来重置。

重置 `log_disk_utilization_threshold` 配置项的示例如下：

```
obclient> ALTER SYSTEM RESET log_disk_utilization_threshold;
```

2.4 查看配置项

查看配置项的 SQL 语法如下：

```
SHOW PARAMETERS [LIKE 'pattern' | WHERE expr] [TENANT = tenant_name]
```

2.4.0.1 说明

- 系统租户下，可以查看本租户的租户级配置项和集群级配置项信息。同时，可以通过指定 `TENANT` 关键字来查看指定租户的配置项信息。
- 普通租户下，可以查看本租户的租户级配置项以及系统租户的集群级配置项信息。
- `WHERE expr` 中可以指定的列属性与 `SHOW PARAMETERS` 返回结果中的列属性一致。

通过 `SHOW PARAMETERS` 语句查看配置项的示例如下：

```
obclient> SHOW PARAMETERS WHERE scope = 'tenant';

obclient> SHOW PARAMETERS WHERE svr_ip != 'XXX.XXX.XXX.XXX';

obclient> SHOW PARAMETERS WHERE INFO like '%ara%';

obclient> SHOW PARAMETERS LIKE 'large_query_threshold';
```

返回结果如下：

```
+-------+----------+---------------+----------+----------------------+-----------
+-------+--------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------+---------
+---------+---------+-----------------+-------------+-----------+
| zone | svr_type | svr_ip | svr_port | name | data_type | value | info | section | scope |
source | edit_level | default_value | isdefault |
+-------+----------+---------------+---------+----------------------+-----------
+-------+-----------------------------------------------------------------------------
-------------------------------------------------------------------------------+---------
+---------+---------+-----------------+-------------+-----------+
| zone1 | observer | 172.xx.xxx.xxx | 2882 | large_query_threshold | TIME | 5s | threshold
for execution time beyond which a request may be paused and rescheduled as a
\'large request\', 0ms means disable \'large request\'. Range: [0ms, +∞) | TENANT |
CLUSTER | DEFAULT | DYNAMIC_EFFECTIVE | 5s | 1 |
+-------+----------+---------------+---------+----------------------+-----------
+-------+-----------------------------------------------------------------------------
-------------------------------------------------------------------------------+---------
+---------+---------+-----------------+-------------+-----------+
1 row in set
```

返回结果中的列属性如下表所示：

列名	含义
zone	所在的 Zone
svr_type	机器类型
svr_ip	机器 IP
svr_port	机器端口
name	配置项名
data_type	配置项数据类型，包括 `NUMBER` 、`STRING` 、`CAPACITY` 等

value	配置项值 **说明** 由于在修改配置项值时，支持修改指定 Zone 或 Server 的配置项值，故不同 Zone 或 Server 对应的配置项的值可能不同。
info	配置项的说明信息
section	配置项所属的分类： ● SSTABLE：表示 SSTable 相关的配置项。 ● OBSERVER：表示 OBServer 节点相关的配置项。 ● ROOT_SERVICE：表示 Root Service 相关的配置项。 ● TENANT：表示租户相关的配置项。 ● TRANS：表示事务相关的配置项。 ● LOAD_BALANCE：表示负载均衡相关的配置项。 ● DAILY_MERGE：表示合并相关的配置项。 ● CLOG：表示 Clog 相关的配置项。 ● LOCATION_CACHE：表示 Location Cache 相关的配置项。 ● CACHE：表示缓存相关的配置项。 ● RPC：表示 RPC 相关的配置项。 ● OBPROXY：表示 OBProxy 相关的配置项。
scope	配置项范围属性： ● TENANT：表示该配置项为租户级配置项 ● CLUSTER：表示该配置项为集群级配置项

source	当前值来源： ● TENANT ● CLUSTER ● CMDLINE ● OBADMIN ● FILE ● DEFAULT
edit_level	定义该配置项的修改行为： ● READONLY：表示该参数不可修改。 ● STATIC_EFFECTIVE：表示该参数可修改但需要重启 OBServer 节点才会生效。 ● DYNAMIC_EFFECTIVE：表示该参数可修改且修改后动态生效。
default_value	配置项的默认值。
isdefault	当前值是否为默认值。 ● 0：表示当前值不是默认值。 ● 1：表示当前值是默认值。

2.5 更多信息

更多配置项参考信息，请参见 配置项和系统变量概述。

3 设置变量

您可以通过系统变量的设置使 OceanBase 数据库的行为符合您业务的要求。

3.1 变量分类

OceanBase 数据库的系统变量分为全局变量和 Session 变量：

- 全局变量：表示 Global 级别的修改，数据库同一租户内的不同用户共享全局变量。全局变量的修改不会随会话的退出而失效。此外，全局变量修改后，对当前已打开的 Session 不生效，需要重新建立 Session 才能生效。
- Session 变量：表示 Session 级别的修改。当客户端连接到数据库后，数据库会复制全局变量来自动生成 Session 变量。Session 变量的修改仅对当前 Session 生效。

名称以 "_" 开头的变量称为隐藏变量，如：`_primary_zone_entity_count`。仅供开发人员在故障排查或紧急运维时使用。

3.2 系统变量相关视图

MySQL 租户下的系统变量相关视图如下：

- `INFORMATION_SCHEMA.GLOBAL_VARIABLES`：记录当前租户下的全局变量信息。
- `INFORMATION_SCHEMA.SESSION_VARIABLES`：记录当前租户下的 Session 变量信息。
- `DBA_OB_SYS_VARIABLES`：记录当前租户下全局变量的修改记录及变量的默认值信息。

Oracle 租户下的系统变量相关视图如下：

- `SYS.TENANT_VIRTUAL_GLOBAL_VARIABLE`：记录当前租户下的全局变量信息。
- `SYS.TENANT_VIRTUAL_SESSION_VARIABLE`：记录当前租户下的 Session 变量信息。
- `DBA_OB_SYS_VARIABLES`：记录当前租户下全局变量的修改记录及变量的默认值信息。

有关视图的详细介绍，请参见《参考指南》中的 **系统视图**。

3.3 设置变量

3.3.0.1 说明

- 设置 Session 级别的变量仅对当前 Session 有效，对其他 Session无效。设置 Global 级别的变量对当前 Session 无效，需要重新登录建立新的 Session 才会生效。
- 设置 Global 级别变量要求当前用户拥有 `SUPER` 权限或 `ALTER SYSTEM` 权限。

设置 Session/Global 级别变量的 SQL 语法如下：

```
SET VARIABLE_NAME = 'VALUE'
SET GLOBAL VARIABLE_NAME = 'VALUE'
```

通过 SQL 语句设置 Session/Global 级别变量。示例如下：

```
obclient> SET ob_query_timeout = 20000000;
obclient> SET GLOBAL ob_query_timeout = 20000000;
```

对于变量值类型为 `INT`，且在 `SHOW VARIABLE` 命令中显示 ON/OFF 或者 True/False 的变量，可以通过如下任意方式设置 Session/Global 级别变量。示例如下：

```
SET foreign_key_checks = ON;
SET foreign_key_checks = 1;
SET GLOBAL foreign_key_checks = ON;
SET GLOBAL foreign_key_checks = 1;
```

对于全局变量、只读变量，我们在创建租户时也能对其进行设置，语法如下：

```
CREATE TENANT [IF NOT EXISTS] tenant_name
[tenant_characteristic_list] [opt_set_sys_var];

tenant_characteristic_list:
tenant_characteristic [, tenant_characteristic...]

tenant_characteristic:
COMMENT 'string'
| {CHARACTER SET | CHARSET} [=] charsetname
| COLLATE [=] collationname
| PRIMARY_ZONE [=] zone
| RESOURCE_POOL_LIST [=](poolname [, poolname...])
| LOCALITY [=] 'locality description'
```

```
opt_set_sys_var:
{SET | SET VARIABLES | VARIABLES} system_var_name = expr [,system_var_name =
expr] ...
```

在创建租户时，将只读变量 `ob_compatibility_mode` 初始化为 'mysql' 或者 'oracle'，将全局变量 `ob_tcp_invited_nodes` 初始化为 '%'。示例如下：

```
obclient> CREATE TENANT IF NOT EXISTS test_tenant
charset='utf8mb4', primary_zone='zone1;zone2,zone3', resource_pool_list=('pool1')
SET ob_compatibility_mode='oracle', ob_tcp_invited_nodes='%';
```

3.4 查看变量

- 通过 `SHOW VARIABLES` 语句查看。

  查看 Session/Global 级别变量的 SQL 语法如下：

  ```
  SHOW VARIABLES [LIKE 'pattern' | WHERE expr]
  SHOW GLOBAL VARIABLES [LIKE 'pattern' | WHERE expr]
  ```

  通过 SQL 语句查询 Session/Global 级别变量。示例如下：

  ```
  obclient> SHOW VARIABLES LIKE 'ob_query_timeout';
  obclient> SHOW VARIABLES WHERE VARIABLE_NAME = 'ob_query_timeout';
  obclient> SHOW GLOBAL VARIABLES LIKE 'ob_query_timeout';
  obclient> SHOW GLOBAL VARIABLES WHERE VARIABLE_NAME = 'ob_query_timeout';
  obclient> SHOW GLOBAL VARIABLES WHERE VARIABLE_NAME LIKE
  'ob_query_timeout';
  +-----------------+----------+
  | Variable_name | Value |
  +-----------------+----------+
  | ob_query_timeout | 10000000 |
  +-----------------+----------+
  1 row in set (0.00 sec)
  ```

返回结果中的列属性如下表所示：

列名	含义
Variable_name	变量名
Value	变量值

- 通过 `SELECT` 语句，在本租户相关视图中查看 Session/Global 级别变量信息。SQL 语法如下：

```
SELECT * FROM view_name WHERE VARIABLE_NAME = '[var name]'
```

在 Oracle 模式下，通过 `SELECT` 语句，在本租户相应视图中查看 Session/Global 级别变量信息。示例如下：

```
obclient> SELECT * FROM SYS.TENANT_VIRTUAL_GLOBAL_VARIABLE WHERE
VARIABLE_NAME = 'ob_query_timeout';

obclient> SELECT * FROM SYS.TENANT_VIRTUAL_SESSION_VARIABLE WHERE
VARIABLE_NAME = 'ob_query_timeout';
```

MySQL 模式下，通过 `SELECT` 语句，在本租户视图中查看 Session/Global 级别变量信息。示例如下：

```
obclient> SELECT * FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES WHERE
VARIABLE_NAME = 'ob_query_timeout';

obclient> SELECT * FROM INFORMATION_SCHEMA.SESSION_VARIABLES WHERE
VARIABLE_NAME = 'ob_query_timeout';
```

- 如果需要查询 Global 级别变量的修改记录及变量的默认值，还可以使用如下语句查询。
  - MySQL 模式

```
obclient [oceanbase]> SELECT * FROM oceanbase.DBA_OB_SYS_VARIABLES
WHERE NAME='ob_query_timeout';
```

  - Oracle 模式

```
obclient[SYS] > SELECT * FROM SYS.DBA_OB_SYS_VARIABLES WHERE
NAME='ob_query_timeout';
```

3.5 更多信息

更多变量参考信息，请参见《系统参考》中的 **系统变量**。

4 转储管理概述

本节主要介绍转储的实现原理。

OceanBase 数据库的存储引擎采用 LSM-Tree 架构，数据大体上被分为 MemTable 和 SSTable 两部分。SSTable 会继续细分为 Mini SSTable、Minor SSTable、Major SSTable 三类。

转储包含两个过程：Mini Compaction 和 Minor Compaction。当 MemTable 的内存使用达到一定阈值时，就需要将 MemTable 中的数据存储到磁盘上的 Mini SSTable 以释放内存空间，该过程称为 Mini Compaction。随着用户数据的写入，Mini SSTable 的数量越来越多，当 Mini SSTable 的数量超过一定的阈值时，后台会自动触发 Minor Compaction。在转储之前首先需要保证被转储的 MemTable 不再进行新的数据写入，该过程称为冻结（Minor Freeze），冻结会阻止当前活跃的 MemTable 再有新的写入，同时会生成新的活跃 MemTable。

4.1 SSTable 的分层策略

OceanBase 数据库在转储的实现过程中引入了 SSTable 的分层策略，在原有的基础上增加了 L0 层。

- L0 层（Mini SSTable）

  被冻结的 MemTable 会直接 Flush 为 Mini SSTable。OceanBase 数据库内部可以同时存在多个 Mini SSTable。

- L1 层（Minor SSTable）

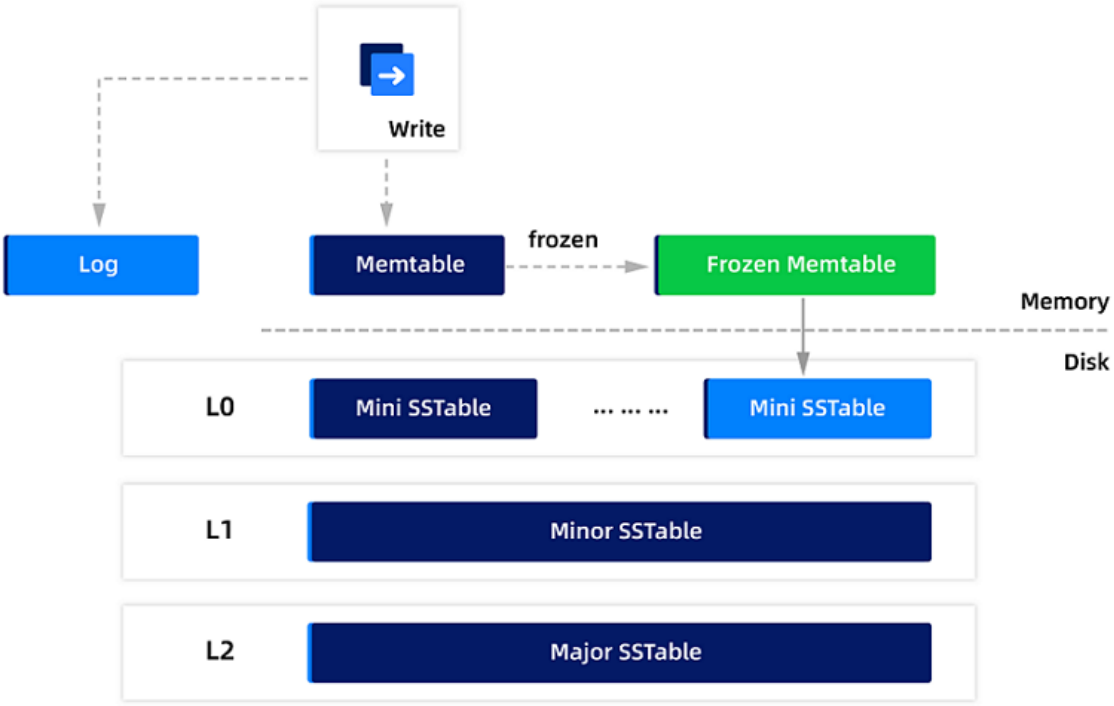
  大多数情况下仅有一个 Minor SSTable。

  由 L0 层的多个 Mini SSTable 与原有的 Minor SSTable 在 L1 层生成新的 Minor SSTable。

- L2 层（Major SSTable）

  基线数据，在合并时产生。一般情况下仅有一个。

SSTable 的分层策略的内部实现原理如下图所示。



4.2 转储触发方式

转储支持自动触发和手动触发两种方式。

5 自动触发转储

本节主要介绍自动转储的触发机制。

当一个租户 Active MemTable 内存的使用量达到 freeze_trigger_percentage * memstore_limit（其中，memstore_limit = 租户内存 * memstore_limit_percentage）时，系统就会自动触发冻结（转储的前置动作），然后再调度转储。当冻结（minor freeze）的次数达到一定的阈值时，就会触发自动合并。

更多配置项 memstore_limit_percentage 和 freeze_trigger_percentage 的说明及设置方法，请参见 修改转储配置。

5.1 相关文档

- 手动触发转储
- 查看转储信息
- 修改转储配置

6 手动触发转储

系统租户和用户租户可以通过 `ALTER SYSTEM MINOR FREEZE` 命令手动触发转储。手动转储支持租户级别、Zone 级别、Server 级别、日志流级别和分区级别。

6.1 系统租户发起转储

系统租户可以发起租户级别、Zone 级别、Server 级别、日志流级别和 Tablet 级别的转储。

1. 使用 `root` 用户登录集群的 `sys` 租户。
2. 根据业务需求，选择合适的转储操作。
   - 发起租户级别的转储

     可以对指定的一个或多个租户进行转储，SQL 语句如下。

     ```
     obclient> ALTER SYSTEM MINOR FREEZE TENANT [=] all_user | all | all_meta |
     tenant_name [, tenant_name ...];
     ```

     示例如下：

     - 系统租户对本租户发起转储

       ```
       obclient> ALTER SYSTEM MINOR FREEZE;
       ```

     - 系统租户对所有用户租户发起转储

       ```
       obclient> ALTER SYSTEM MINOR FREEZE TENANT = all_user;
       ```

       或者

       ```
       obclient> ALTER SYSTEM MINOR FREEZE TENANT = all;
       ```

6.1.0.1 说明

OceanBase 数据库从 V4.2.1 版本开始，`TENANT = all_user` 与 `TENANT = all` 语义相同，在需要生效范围为所有用户租户时，推荐使用 `TENANT = all_user`，后续 `TENANT = all` 将废弃不再使用。

- 系统租户对所有 Meta 租户发起转储

  ```
  obclient> ALTER SYSTEM MINOR FREEZE TENANT = all_meta;
  ```

- 系统租户对指定租户发起转储

```
obclient> ALTER SYSTEM MINOR FREEZE TENANT = tenant1;
```

- 发起 Zone 级别的转储

  可以对指定的 Zone 进行转储，每次仅支持指定一个 Zone，SQL 语句如下。

```
obclient> ALTER SYSTEM MINOR FREEZE ZONE [=] zone_name;
```

  示例：

```
obclient> ALTER SYSTEM MINOR FREEZE ZONE = zone1;
```

- 发起 Server 级别的转储

  可以对指定的一个或多个 OBServer 节点进行转储，SQL 语句如下。

```
obclient> ALTER SYSTEM MINOR FREEZE SERVER = ('ip:port' [, 'ip:port'...]);
```

  示例：

```
obclient> ALTER SYSTEM MINOR FREEZE SERVER = ('xx.xx.xx.xx:2882','xx.xx.xx.xx:2882');
```

- 发起日志流级别的转储

  可以对指定租户的指定日志流进行转储，SQL 语句如下。

```
ALTER SYSTEM MINOR FREEZE TENANT [=] tenant_name LS [=] ls_id;
```

  其中，`tenant_name` 为指定租户的租户名；`ls_id` 为指定租户的日志流 ID；租户的日志流 ID 可以通过查询 `oceanbase.CDB_OB_TABLE_LOCATIONS` 视图获取。

  示例：

```
obclient> ALTER SYSTEM MINOR FREEZE TENANT = t1 LS = 1001;
```

  语句执行后，系统会对指定租户的指定日志流的所有 Tablet 进行转储。

- 发起分区级别的转储

  在 OceanBase 数据库中，分区与 Tablet 一一对应。

  您可以对指定租户的指定 Tablet 进行转储，SQL 语句如下。

```
ALTER SYSTEM MINOR FREEZE TENANT [=] tenant_name TABLET_ID = tablet_id;
```

或者，也可以对指定租户指定日志流中的指定 Tablet 进行转储，SQL 语句如下。

```
ALTER SYSTEM MINOR FREEZE TENANT [=] tenant_name LS [=] ls_id TABLET_ID = tablet_id;
```

其中，tenant_name 为指定租户的租户名；ls_id 为指定租户的日志流 ID；tablet_id 为指定日志流中的 Tablet ID，租户的日志流 ID 和 Tablet ID 可以通过查询视图 CDB_OB_TABLE_LOCATIONS 来获取。有关视图 CDB_OB_TABLE_LOCATIONS 中各字段的详细介绍，请参见 oceanbase.CDB_OB_TABLE_LOCATIONS。

对指定租户的指定 Tablet 进行转储的示例如下：

```
obclient> ALTER SYSTEM MINOR FREEZE TENANT = tenant1 TABLET_ID = 200001;
```

对指定租户指定日志流中的指定 Tablet 进行转储的示例如下：

```
obclient> ALTER SYSTEM MINOR FREEZE TENANT = tenant1 LS = 1001 TABLET_ID = 200001;
```

6.2 用户租户发起转储

用户租户只能对本租户发起租户级别和分区级别的转储。

1. 租户管理员登录集群的 MySQL 租户或 Oracle 租户。
2. 根据业务需求，选择合适的转储操作。
   - 租户级别的转储

     ```
     obclient> ALTER SYSTEM MINOR FREEZE;
     ```

   - 分区级别的转储

     在 OceanBase 数据库中，分区与 Tablet 一一对应。

     您可以对本租户内指定 Tablet 发起转储，语句如下：

     ```
     ALTER SYSTEM MINOR FREEZE TABLET_ID = tablet_id;
     ```

     其中，tablet_id 可以通过查询视图 DBA_OB_TABLE_LOCATIONS 来获取。有关视图 DBA_OB_TABLE_LOCATIONS 中各字段的详细介绍，参见 DBA_OB_TABLE_LOCATIONS。

     示例：

```
obclient> ALTER SYSTEM MINOR FREEZE TABLET_ID = 200001;
```

6.3 后续操作

发起转储后，您可以查看转储情况，具体操作请参见 查看转储进度。

6.4 相关文档

- 自动触发转储
- 修改转储配置

7 查看转储信息

触发转储后，您可以通过视图查看转储进度和转储历史。

7.1 查看转储进度

1. 租户管理员登录集群的 `sys` 租户或用户租户。
2. 查看转储进度，可以查看转储未完成的数据量及预期完成时间等信息。

   `GV$OB_TABLET_COMPACTION_PROGRESS` 视图用于展示 Tablet 级别 Compaction 任务的进度信息，且仅展示正在运行的任务，任务结束后就不再展示。具体查询语句如下：

   - 系统租户查看转储进度

     ```
     obclient> SELECT * FROM oceanbase.GV$OB_TABLET_COMPACTION_PROGRESS
     WHERE TYPE='MINI_MERGE'\G
     ```

   - 用户租户查看转储进度
     - MySQL 模式

       ```
       obclient> SELECT * FROM oceanbase.GV$OB_TABLET_COMPACTION_PROGRESS
       WHERE TYPE='MINI_MERGE'\G
       ```

     - Oracle 模式

       ```
       obclient> SELECT * FROM SYS.GV$OB_TABLET_COMPACTION_PROGRESS
       WHERE TYPE='MINI_MERGE'\G
       ```

   查询结果的示例如下：

   ```
   *************************** 1. row ***************************
   SVR_IP: xx.xx.xx.xx
   SVR_PORT: 2401
   TENANT_ID: 1002
   TYPE: MINI_MERGE
   LS_ID: 1001
   TABLET_ID: 1152921504606847235
   COMPACTION_SCN: 1680514780195130031
   ```

```
TASK_ID: Y9610BA2DA3E-0005F7FD6E1FE0FF-0-0
STATUS: NODE_RUNNING
DATA_SIZE: 31890729
UNFINISHED_DATA_SIZE: 3351030
PROGRESSIVE_COMPACTION_ROUND: 1
CREATE_TIME: 2023-04-03 17:49:17.278506
START_TIME: 2023-04-03 17:51:57.953999
ESTIMATED_FINISH_TIME: 2023-04-03 23:32:25.969930
START_CG_ID: 0
END_CG_ID: 0
1 row in set
```

查询结果中的部分字段说明如下：

- `TYPE`：表示 Compaction 任务的类型。
  - `MDS_TABLE_MERGE`：将系统的元数据按照 SSTable 的格式持久化到磁盘里。
  - `MAJOR_MERGE`：租户级合并
  - `MEDIUM_MERGE`：分区级合并
  - `MINI_MERGE`：Mini Compaction，将 MemTable 转变成 Mini SSTable。
  - `MINOR_MERGE`：Minor Compaction，多个 Mini SSTable 合成一个新的 Mini SSTable 或者多个 Mini SSTable 与一个 Minor SSTable 合成一个新的 Minor SSTable。
  - `META_MAJOR_MERGE`：一种特殊的 Compaction 类型，是将某个指定时间点之前的数据合成一个 Meta Major SSTable，其数据格式与 Major SSTable 一样，不包含多版本数据和未提交事务数据。
- `STATUS`：表示任务状态。任务正在运行中时，该字段值为 `NODE_RUNNING`。
- `DATA_SIZE`：表示需要转储的总数据量。
- `UNFINISHED_DATA_SIZE`：表示未完成转储的数据量。
- `ESTIMATED_FINISH_TIME`：表示预计完成时间。

有关 GV$OB_TABLET_COMPACTION_PROGRESS 视图中字段的详细介绍，请参见 GV$OB_TABLET_COMPACTION_PROGRESS。

根据查询结果，对于未出现在该视图中的 Tablet 或长时间未完成的 Tablet，可以进一步查看诊断视图 GV$OB_COMPACTION_DIAGNOSE_INFO，确认是否有异常情况出现。

有关 GV$OB_COMPACTION_DIAGNOSE_INFO 视图中字段的详细介绍，请参见 GV$OB_COMPACTION_DIAGNOSE_INFO。

7.2 查看转储历史

1. 租户管理员登录集群的 sys 租户或用户租户。
2. 查看转储历史。

   GV$OB_TABLET_COMPACTION_HISTORY 视图用于展示 Tablet 级别 Compaction 的历史信息。具体查询语句如下：

   - 系统租户查看转储历史

     ```
     obclient> SELECT * FROM oceanbase.GV$OB_TABLET_COMPACTION_HISTORY
     WHERE TYPE='MINI_MERGE'\G
     ```

   - 用户租户查看转储历史
     - MySQL 模式

       ```
       obclient> SELECT * FROM oceanbase.GV$OB_TABLET_COMPACTION_HISTORY
       WHERE TYPE='MINI_MERGE'\G
       ```

     - Oracle 模式

       ```
       obclient> SELECT * FROM SYS.GV$OB_TABLET_COMPACTION_HISTORY WHERE
       TYPE='MINI_MERGE'\G
       ```

   查询结果的示例如下：

   ```
   *************************** 1. row ***************************
   SVR_IP: 172.xx.xxx.xxx
   SVR_PORT: 2882
   TENANT_ID: 1002
   LS_ID: 1
   ```

```
TABLET_ID: 49402
TYPE: MINI_MERGE
COMPACTION_SCN: 1747418403693944001
START_TIME: 2025-05-17 02:00:03.718154
FINISH_TIME: 2025-05-17 02:00:03.755331
TASK_ID: YB42AC1E87E0-000635386FA2C34E-0-0
OCCUPY_SIZE: 2123360
MACRO_BLOCK_COUNT: 2
MULTIPLEXED_MACRO_BLOCK_COUNT: 0
NEW_MICRO_COUNT_IN_NEW_MACRO: 103
MULTIPLEXED_MICRO_COUNT_IN_NEW_MACRO: 0
TOTAL_ROW_COUNT: 17381
INCREMENTAL_ROW_COUNT: 17381
COMPRESSION_RATIO: 1
NEW_FLUSH_DATA_RATE: 62511
PROGRESSIVE_COMPACTION_ROUND: 0
PROGRESSIVE_COMPACTION_NUM: 0
PARALLEL_DEGREE: 1
PARALLEL_INFO: -
PARTICIPANT_TABLE: table_cnt=1,start_scn=1,end_scn=1747418403693944001;
MACRO_ID_LIST: 6280,6282
COMMENTS: comment="cost_mb=5;";
START_CG_ID: 0
END_CG_ID: 0
KEPT_SNAPSHOT:
MERGE_LEVEL: MACRO_BLOCK_LEVEL
EXEC_MODE: EXEC_MODE_LOCAL
IS_FULL_MERGE: FALSE
IO_COST_TIME_PERCENTAGE: 6
```

```
MERGE_REASON:
BASE_MAJOR_STATUS:
CO_MERGE_TYPE:
MDS_FILTER_INFO:
1 row in set
```

有关 `GV$OB_TABLET_COMPACTION_HISTORY` 视图中字段的详细介绍，参见 GV$OB_TABLET_COMPACTION_HISTORY。

8 修改转储配置

本节主要介绍转储参数及其设置方法。

8.1 转储参数

配置项	含义	默认值	设定范围
minor_compact_trigger	租户级配置项，用于配置触发转储（Minor Compaction）的 SSTable 个数的阈值。	2	[0, 16]
freeze_trigger_percentage	租户级配置项，租户 MemStore 占用内存的比例阈值，达到该值则触发 Freeze。	20	(0, 100)
	集群级配置项，租户 MemStore 占租户总内存的百分比。 #### 8.1.0.1 说明 在 V4.3.x 版本中，从 V4.3.0 版本开始，还支持通过租户级隐藏配置项 `_memstore_limit_percentage` 来配置租户 MemStore 占租户总内存的百分比，除了生效范围不同，其功能及默认值与集群级配置项 `memstore_limit_percentage` 均相同。在配置这两个配置项时，需要注意以下事项：		

memstore_limit_percentage	• 如果仅配置了 _memstore_limit_percentage 或 memstore_limit_percentage 中某一个配置项的值（非默认值），则以配置的值（非默认值）为准。
	• 如果同时配置了租户级隐藏配置项 _memstore_limit_percentage（非默认值）和集群级配置项 memstore_limit_percentage（非默认值）的值，则以 _memstore_limit_percentage配置的值为准。
	• 如果两者均未配置，或均配置为默认值时，系统将采用以下自适应策略：
	  ■ 内存为 8G （真实 Memory）及以下规格的租户，其租户 Memstore 的配置比例为 40%。
	  ■ 内存为 8G 以上规格的租户，其租户 Memstore 的配置比例为 50%。
		0	[0, 100)

8.2 通过 SQL 语句修改转储配置

1. 租户管理员登录到数据库。
2. 分别执行以下语句，修改转储配置。

   示例如下：

```
obclient> ALTER SYSTEM SET minor_compact_trigger=2;

obclient> ALTER SYSTEM SET freeze_trigger_percentage=20;

obclient> ALTER SYSTEM SET memstore_limit_percentage=50;
```

3. 修改成功后，可以通过 `SHOW PARAMETERS` 语句查看是否修改成功。

   示例如下：

```
obclient> SHOW PARAMETERS LIKE 'minor_compact_trigger';

obclient> SHOW PARAMETERS LIKE 'freeze_trigger_percentage';

obclient> SHOW PARAMETERS LIKE 'memstore_limit_percentage';
```

9 合并管理概述

本节主要介绍合并的分类、状态及压缩算法。

合并操作（Major Compaction）是将动静态数据做归并，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。合并与转储的最大区别在于，合并是租户在一个统一的快照点与其对应的静态数据进行合并的行为，最终会形成一个租户级的快照。

9.1 合并分类

按照合并数据量，合并可以分为：

- 全量合并：将静态数据全部读出并和动态数据合并为最终的静态数据。合并时间长，耗费 IO 和 CPU。
- 增量合并：仅仅合并被修改过的宏块，没有改变的宏块进行复用。

  增量合并极大地减少了合并的工作量，是 OceanBase 数据库目前默认的合并算法。
- 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。
- 并行合并：将数据划分到不同线程中并行做合并。

更多合并相关的详细介绍信息，请参见 合并。

合并相关的配置项及其说明请参见 修改合并配置。

9.2 合并状态

合并的状态可以通过视图 `DBA_OB_ZONE_MAJOR_COMPACTION` 中的 `status` 列来查看。

合并状态主要有以下几种：

- `IDLE`：表示未进行合并。
- `COMPACTING`：表示正在进行合并。
- `VERIFYING`：表示正在校验 Checksum 中。

9.3 合并的压缩算法

OceanBase 数据库不会实时将小部分数据刷盘，而是通过合并的方式集中对数据进行刷盘，因此可以采用压缩的方式来写入磁盘，此时磁盘的空间利用率得到提升。在压缩算法和压缩功能的选择上，您可以根据实际情况选择高压缩率但是耗费更多 CPU 的方式，也可以选择普通的压缩方式。

您可以通过参数 default_compress_func 来配置压缩方式，默认值为 zstd_1.3.8 。其他可指定的值还有 none 、 lz4_1.0 、 snappy_1.0 、和 zstd_1.0 。

9.3.0.1 说明

更高的压缩率将更节约磁盘空间，但是也意味着性能的牺牲。例如，通常 ZSTD 比 LZ4 节省更多的空间，但是 ZSTD 合并的时间更长，同时需要 IO 查询的 RT 也更大。

如果希望为单独的表选择特别的压缩算法，您可以在创建数据表时指定压缩算法。

租户创建表并指定压缩算法的语法请参见 CREATE TABLE（MySQL 模式） 和 CREATE TABLE（Oracle 模式）。

9.4 合并的触发方法

OceanBase 数据库支持自动触发合并、定时触发合并、自适应合并和手动触发合并。

9.5 相关文档

合并

10 自动触发合并

合并可以通过自动触发来完成。

10.1 前提条件

触发合并前，需要确保全局合并开关已开启，全局合并开关由集群级配置项 `enable_major_freeze` 控制，默认值为 `True`，表示开启合并开关。

如果当前集群未开启全局合并开关，请参考以下操作开启全局合并开关。

1. 使用 `root` 用户登录集群的 `sys` 租户。
2. 确认全局合并开关是否已开启。

   ```
   obclient> SHOW PARAMETERS LIKE 'enable_major_freeze';
   ```

3. 如果显示未开启，执行以下语句，开启合并开关。

   ```
   obclient> ALTER SYSTEM SET enable_major_freeze='True';
   ```

   关于配置项 `enable_major_freeze` 的更多信息，请参见 enable_major_freeze。

10.2 自动合并的触发条件

当冻结（minor freeze）的次数达到一定的阈值时，就会自动触发合并。

10.3 相关文档

- 定时触发合并
- 手动触发合并
- 手动控制合并
- 查看合并信息

11 定时触发合并

您也可以通过指定每日合并触发时间来定时触发合并。

11.1 前提条件

触发合并前，需要确保全局合并开关已开启，全局合并开关由集群级配置项 `enable_major_freeze` 控制，默认值为 `True`，表示开启合并开关。

如果当前集群未开启全局合并开关，请参考以下操作开启全局合并开关。

1. 使用 `root` 用户登录集群的 `sys` 租户。
2. 确认全局合并开关是否已开启。

   ```
   obclient> SHOW PARAMETERS LIKE 'enable_major_freeze';
   ```

3. 如果显示未开启，执行以下语句，开启合并开关。

   ```
   obclient> ALTER SYSTEM SET enable_major_freeze='True';
   ```

   关于配置项 `enable_major_freeze` 的更多信息，请参见 enable_major_freeze。

11.2 定时合并的触发条件

开启全局合并开关后，合并的定时触发条件是，每当系统时间达到了每日合并的时间点时，就会自动触发合并。每日合并的时间点通过配置项 `major_freeze_duty_time` 来控制，您也可以在 OCP 上修改，默认是每天 `02:00` 进行合并。

`major_freeze_duty_time` 及其他合并相关参数的说明及设置请参见 修改合并配置。

11.3 通过 SQL 语句指定每日合并触发时间

1. 使用 `root` 用户登录集群的 `sys` 租户。
2. 执行以下语句，修改每日合并的触发时间。
   - 修改本租户每日合并的触发时间

     ```
     obclient> ALTER SYSTEM SET major_freeze_duty_time='01:00';
     ```

   - 修改所有用户租户每日合并的触发时间

对于租户较多的场景，当所有用户租户每次合并的触发时间时，可能会出现 CPU 利用率突然升高的问题。如果出现了 CPU 利用率突然升高的情况，建议您将各用户租户的每日合并时间设置为不同的时间。

```
obclient> ALTER SYSTEM SET major_freeze_duty_time='01:00' TENANT = all_user;
```

或者

```
obclient> ALTER SYSTEM SET major_freeze_duty_time='01:00' TENANT = all;
```

11.3.0.1 说明

OceanBase 数据库从 V4.2.1 版本开始，`TENANT = all_user` 与 `TENANT = all` 语义相同，在需要生效范围为所有用户租户时，推荐使用 `TENANT = all_user`，后续 `TENANT = all` 将废弃不再使用。

- 修改所有 Meta 租户每日合并的触发时间

  对于租户较多的场景，当所有 Meta 租户每次合并的触发时间相同时，可能会出现 CPU 利用率突然升高的问题。如果出现了 CPU 利用率突然升高的情况时，建议您将各 Meta 租户的每日合并时间设置为不同的时间。

  ```
  obclient> ALTER SYSTEM SET major_freeze_duty_time='01:00' TENANT =
  all_meta;
  ```

- 修改指定租户每日合并的触发时间

  以下语句每次仅支持指定一个租户，如果要修改多个租户，需要重复执行该语句。

  ```
  obclient> ALTER SYSTEM SET major_freeze_duty_time='01:00' TENANT = tenant1;
  ```

11.4 相关文档

- 自动触发合并
- 手动触发合并
- 手动控制合并
- 查看合并信息

12 自适应合并

OceanBase 数据库从 V4.1.0 版本开始支持自适应合并。开启自适应合并后，系统会实时采集并统计用户的查询、写入等信息，再根据这些信息来识别可能导致的查询慢的场景，自适应地调度合并任务，达到对用户无感知地解决相关场景可能导致的慢查询问题。自适应合并开启期间，系统以分区为维度进行统计信息的采集及合并任务的调度。

使用自适应合并功能，可以解决如下问题：

- 避免一部分 Buffer 表问题的出现。
- 优化导数场景下的查询性能。
- 识别并规避一部分慢查询场景。

12.1 触发规则

当分区 Leader 满足一定条件时，系统就会自动为该分区及 Follower 副本调度自适应合并。自适应合并的触发规则如下：

- 系统通过分析一段时间内分区的统计信息，获取到该分区查询和转储的次数以及增量数据中 Insert 行数的占比，确认该场景是否为导数场景。如果是导数场景，则对该分区触发自适应合并。
- 系统通过分析一段时间内分区的统计信息，获取到该分区转储的次数以及增量数据中更新和删除行的占比，确认该场景是否为导数场景。如果是导数场景，则对该分区触发自适应合并。
- 系统通过统计分区增量数据的总行数，确认该场景是否为持续写入场景，如果是持续写入场景，系统对该分区触发自适应合并。
- 系统通过查询扫描的数据量与实际有效数据的比例，确认针对该分区的查询是否为慢查询，如果为慢查询，系统对该分区触发自适应合并。

12.2 注意事项

- 自适应合并可以有效减少一部分查询慢的问题，如果您的业务对查询性能要求比较高，并且存在自适应合并所支持的场景时，建议不要关闭自适应合并功能。
- 在 OceanBase 数据库中，名称以 "_" 开头的配置项均为隐藏配置项，仅供开发人员在故障排查或紧急运维时使用。如果确认需要关闭自适应合并功能，请联系技术支持人员协助处理。

12.3 步骤一：开启自适应合并

自适应合并功能由租户级隐藏配置项 `_enable_adaptive_compaction` 进行控制，默认为开启状态。

1. 使用 root 用户登录集群的 sys 租户。
2. 查询租户级隐藏配置项 _enable_adaptive_compaction 的值。

```
obclient [oceanbase]> SELECT * FROM oceanbase.GV$OB_PARAMETERS WHERE
NAME LIKE '_enable_adaptive_compaction';
```

查询结果如下：

```
+---------------+---------+-------+-------+----------+--------------------------
+----------+-------
+-------------------------------------------------------------------------------
+---------+-----------------+---------------+----------+
| SVR_IP | SVR_PORT | ZONE | SCOPE | TENANT_ID | NAME | DATA_TYPE | VALUE | INFO
| SECTION | EDIT_LEVEL | DEFAULT_VALUE | ISDEFAULT |
+---------------+---------+-------+-------+----------+--------------------------
+----------+-------
+-------------------------------------------------------------------------------
+---------+-----------------+---------------+----------+
| 172.xx.xxx.xxx | 2882 | zone1 | TENANT | 1 | _enable_adaptive_compaction | BOOL |
True | specifies whether allow adaptive compaction schedule and information
collection | TENANT | DYNAMIC_EFFECTIVE | True | YES |
| 172.xx.xxx.xxx | 2882 | zone1 | TENANT | 1001 | _enable_adaptive_compaction |
BOOL | True | specifies whether allow adaptive compaction schedule and
information collection | TENANT | DYNAMIC_EFFECTIVE | True | YES |
| 172.xx.xxx.xxx | 2882 | zone1 | TENANT | 1002 | _enable_adaptive_compaction |
BOOL | True | specifies whether allow adaptive compaction schedule and
information collection | TENANT | DYNAMIC_EFFECTIVE | True | YES |
| 172.xx.xxx.xxx | 2882 | zone1 | TENANT | 1003 | _enable_adaptive_compaction |
BOOL | True | specifies whether allow adaptive compaction schedule and
information collection | TENANT | DYNAMIC_EFFECTIVE | True | YES |
| 172.xx.xxx.xxx | 2882 | zone1 | TENANT | 1004 | _enable_adaptive_compaction |
```

```
BOOL | True | specifies whether allow adaptive compaction schedule and
information collection | TENANT | DYNAMIC_EFFECTIVE | True | YES |
+---------------+---------+-------+--------+-----------+--------------------------
+-----------+-------
+-------------------------------------------------------------------------------
+---------+------------------+--------------+-----------+
5 rows in set
```

查询结果中，`VALUE` 列对应的值为 `True`，表示已开启自适应合并。

3. 如果未开启自适应合并，可以执行以下语句，开启自适应合并。

```
obclient [oceanbase]> ALTER SYSTEM SET _enable_adaptive_compaction = True
TENANT = tenant_name;
```

12.3.0.1 注意

由于自适应合并调度任务的执行会占用部分系统资源，为了减少系统资源的占用，您可以通过将租户级隐藏配置项 `_enable_adaptive_compaction` 的值修改为 `False` 来关闭除 extreme 模式表以外的其他自适应合并的调度。

12.4 （可选）步骤二：通过表选项 table_mode 指定自适应合并策略

开启自适应合并后，为了更加灵活地解决 Buffer 表带来的性能下降问题，OceanBase 数据库在 V4.2.3 版本开始支持表选项 `table_mode`。用户可以通过为每张表设置不同的 `table_mode` 值以便指定不同的快速冻结与自适应合并策略以应对 Buffer 表问题。

您可以在创建表或修改表时，通过表选项 table_mode 指定不同模式的自适应合并策略。表选项 table_mode 提供了以下 5 种模式来对应不同的合并策略。

模式	转储后触发合并的概率
normal（默认值）	极低
queuing	低
moderate	中
super	高
extreme	极高

如果您对业务比较熟悉和了解，您可以在创建表时根据业务情况，为表设置一个合适的模式。例如，创建一张 queuing 模式的表，示例如下。

```
obclient> CREATE TABLE tbl1 (c1 int, c2 double) table_mode = 'queuing';
```

在上述语句中，如果不显式指定 table_mode 值，则系统默认创建的是 normal 模式的表。

创建表后，当您观测到某张表的读放大现象严重，并出现了 Buffer 表现象时，您可以根据业务需要，手动修改该表的 table_mode，以便系统更频繁地触发该表的合并，加速查询。由于合并的频率越高，消耗的计算资源将会更多，建议您根据触发合并的频率从低到高逐级调整 table_mode 值。例如，先从 normal 调整到 queuing，观察一段时间后再从 queuing 调整到 moderate。调整示例如下。

```
obclient> ALTER TABLE tbl1 SET table_mode = 'moderate';
```

修改表的语句执行成功后，系统会有一定的延时（大约 2 分钟），待修改生效后，系统将以配置的合并模式调度自适应合并以解决 Buffer 表现象。

12.5 相关文档

- 自动触发合并
- 定时触发合并
- 手动触发合并
- 手动控制合并

13 手动触发合并

合并也可以通过手动触发来完成。手动合并包括租户级合并和分区级合并。

自适应合并实际调度的就是分区级合并，只是由系统适时地对有需求的分区发起分区合并任务。同时，分区级合并的使用场景与自适应合并解决的场景相同，即主要针对导数、频繁DML、查询效率低等场景，有关自适应合并的详细介绍，请参见 自适应合并。如果您因业务需要关闭了自适应合并功能，遇到查询慢等问题，您可以使用手动方式发起分区级合并来解决这些问题。

13.1 注意事项

分区与 Tablet 一一对应，在进行分区级合并时，需要注意以下事项：

- 当对应的分区正在执行租户级合并任务时，无法发起分区级合并。
- 当对应的分区正在执行自适应调度的合并任务时，无法发起分区级合并。
- 当对应分区的多副本处于不一致状态时，无法发起分区级合并。
- 当对应分区处于恢复中或 Transfer 状态时，无法发起分区级合并。
- 当合并任务被暂停时，不允许发起分区级合并。
- 分区级合并的实质为对同一分区的多个副本执行 Major Compaction 任务，会消耗 CPU 及磁盘 I/O。在执行分区级合并操作前，您需要权衡当前租户的资源占用情况，并且当分区级合并命令执行成功后，会出现 CPU、I/O 占用升高的情况。

13.2 前提条件

触发合并前，需要确保全局合并开关已开启，全局合并开关由集群级配置项 `enable_major_freeze` 控制，默认值为 `True`，表示开启合并开关。

如果当前集群未开启全局合并开关，请参考以下操作开启全局合并开关。

1. 使用 `root` 用户登录集群的 `sys` 租户。
2. 确认全局合并开关是否已开启。

   ```
   obclient> SHOW PARAMETERS LIKE 'enable_major_freeze';
   ```

3. 如果显示未开启，执行以下语句，开启合并开关。

   ```
   obclient> ALTER SYSTEM SET enable_major_freeze='True';
   ```

   关于配置项 `enable_major_freeze` 的更多信息，请参见 enable_major_freeze。

13.3 系统租户手动发起合并

系统租户可以对本租户、所有租户或指定租户发起合并操作。

具体操作如下：

1. 使用 root 用户登录集群的 sys 租户。
2. 执行以下命令，发起合并。
   - 发起租户级别的合并

     SQL 语句如下：

     ```
     ALTER SYSTEM MAJOR FREEZE TENANT [=] all_user | all | all_meta | tenant_name [, tenant_name ...];
     ```

     其中，[=] 表示等号（=）为可选。

     示例如下：

     - 系统租户对本租户发起合并

       ```
       obclient> ALTER SYSTEM MAJOR FREEZE;
       ```

     - 系统租户对所有用户租户发起合并

       ```
       obclient> ALTER SYSTEM MAJOR FREEZE TENANT = all_user;
       ```

       或者

       ```
       obclient> ALTER SYSTEM MAJOR FREEZE TENANT = all;
       ```

13.3.0.1 说明

OceanBase 数据库从 V4.2.1 版本开始，TENANT = all_user 与 TENANT = all 语义相同，在需要生效范围为所有用户租户时，推荐使用 TENANT = all_user，后续 TENANT = all 将废弃不再使用。

- 系统租户对所有 Meta 租户发起合并

  ```
  obclient> ALTER SYSTEM MAJOR FREEZE TENANT = all_meta;
  ```

- 系统租户对指定租户发起合并

  ```
  
  ```

```
obclient> ALTER SYSTEM MAJOR FREEZE TENANT = tenant1,tenant2;
```

- 发起分区级的合并

  对指定租户发起分区级的合并的 SQL 语句如下：

  ```
  ALTER SYSTEM MAJOR FREEZE TENANT [=] tenant_name TABLET_ID = tablet_id;
  ```

  其中，tablet_id 可以由系统租户通过查询视图 CDB_OB_TABLE_LOCATIONS 来获取。有关视图 CDB_OB_TABLE_LOCATIONS 中各字段的详细介绍，请参见 oceanbase.CDB_OB_TABLE_LOCATIONS。

  系统租户对租户 tenant1 发起分区级合并的示例如下：

  ```
  obclient> ALTER SYSTEM MAJOR FREEZE TENANT = tenant1 TABLET_ID = 200001;
  ```

13.3.0.2 注意

分区级合并命令与租户级合并以及自适应调度策略发起的合并均存在互斥关系，命令执行成功并不代表分区合并发起成功。您可以根据视图 GV$OB_MERGE_INFO 查看指定分区中 ACTION='MEDIUM_MERGE' 的合并信息，或者根据视图 GV$OB_TABLET_COMPACTION_HISTORY 查看指定分区中 TYPE='MEDIUM_MERGE' 的合并信息来确认合并是否成功发起。有关查看合并信息的详细操作，请参见 查看合并信息。

13.4 用户租户手动发起合并

用户租户只能对本租户发起租户级别和分区级别的合并操作。

1. 用户租户的租户管理员登录到数据库。
2. 对本租户发起租户级合并。
   - 发起本租户的合并

     ```
     obclient> ALTER SYSTEM MAJOR FREEZE;
     ```

   - 发起分区级别的合并

     语句如下：

     ```
     ALTER SYSTEM MAJOR FREEZE TABLET_ID = tablet_id;
     ```

其中，tablet_id 可以通过查询视图 DBA_OB_TABLE_LOCATIONS 来获取。有关视图 DBA_OB_TABLE_LOCATIONS 中各字段的详细介绍，参见 DBA_OB_TABLE_LOCATIONS。

示例如下：

```
obclient> ALTER SYSTEM MAJOR FREEZE TABLET_ID = 200001;
```

13.4.0.1 注意

分区级合并命令与租户级合并以及自适应调度策略发起的合并均存在互斥关系，命令执行成功并不代表分区合并发起成功。用户租户可以根据视图 GV$OB_TABLET_COMPACTION_HISTORY 查看指定分区中 TYPE='MEDIUM_MERGE' 的合并信息来确认合并是否成功发起。有关查看合并信息的详细操作，请参见 查看合并信息。

13.5 相关文档

- 自动触发合并
- 定时触发合并
- 自适应合并
- 手动控制合并
- 查看合并信息

14 手动控制合并

通过 MAJOR FREEZE 命令触发合并后，系统会自动调度合并任务。不同于 OceanBase 数据库 V2.x.x 和 V3.x.x 版本，从 OceanBase 数据库 V4.0.x 版本开始，合并将采用统一合并策略来执行合并流程，即租户的所有 Zone 同时开始执行合并，且所有 Zone 均完成合并后，才表示该租户完成合并。

合并过程中，不再需要您手动控制各个 Zone 的合并顺序等事项，也不存在轮转合并。如果在合并过程中出现错误，您可以参考本文手动控制合并。例如，Checksum 校验出错，会使得该租户当前轮次的合并无法结束，您可以暂停当前合并(suspend merge)，待 Checksum 恢复正确以后（例如，人工介入修复），再清理 Checksum 错误的标记（clear merge error），然后继续执行当前轮次的合并 (resume merge)。

14.1 前提条件

手动控制合并前，请确认已通过 MAJOR FREEZE 命令触发合并，手动触发合并的详细操作，请参见 手动触发合并。

14.2 注意事项

当合并任务被暂停时，允许发起租户级合并，但不允许发起分区级合并。对于合并任务暂停时发起的租户级合并，系统并不会立即执行本次租户级合并任务，而是在恢复合并后才执行本次租户级合并。

14.3 系统租户手动控制合并

合并开始后，系统租户可以控制所有租户或指定租户的合并操作。可以控制的合并操作包括租户级合并和分区级合并。

具体操作如下：

1. 使用 root 用户登录集群的 sys 租户。
2. 根据业务实际情况，选择合适的命令控制合并。
   - 暂停所有用户租户、所有 Meta 租户或指定租户的合并

     开始合并后，如果需要暂停所有用户租户、所有 Meta 租户或指定租户的合并，可以执行以下语句。
     - 暂停所有用户租户的合并

       ```
       ALTER SYSTEM SUSPEND MERGE TENANT = all_user;
       ```

       或者

       ```
       ```

```
ALTER SYSTEM SUSPEND MERGE TENANT = all;
```

14.3.0.1 说明

OceanBase 数据库从 V4.2.1 版本开始，`TENANT = all_user` 与 `TENANT = all` 语义相同，在需要生效范围为所有用户租户时，推荐使用 `TENANT = all_user`，后续 `TENANT = all` 将废弃不再使用。

- 暂停所有 Meta 租户的合并

```
ALTER SYSTEM SUSPEND MERGE TENANT = all_meta;
```

- 暂停指定租户的合并

```
ALTER SYSTEM SUSPEND MERGE TENANT = tenant1，tenant2;
```

其中，`=` 为可选；`tenant1`、`tenant2` 需要替换为实际的租户名。

- 恢复合并

暂停合并后，也可以恢复合并，语句如下。

- 恢复所有用户租户的合并

```
ALTER SYSTEM RESUME MERGE TENANT = all_user;
```

或者

```
ALTER SYSTEM RESUME MERGE TENANT = all;
```

14.3.0.2 说明

OceanBase 数据库从 V4.2.1 版本开始，`TENANT = all_user` 与 `TENANT = all` 语义相同，在需要生效范围为所有用户租户时，推荐使用 `TENANT = all_user`，后续 `TENANT = all` 将废弃不再使用。

- 恢复所有 Meta 租户的合并

```
ALTER SYSTEM RESUME MERGE TENANT = all_meta;
```

- 恢复指定租户的合并

```
ALTER SYSTEM RESUME MERGE TENANT = tenant1，tenant2;
```

其中，`=` 为可选；`tenant1` 、`tenant2` 需要替换为实际的租户名。

- 清理 Checksum 错误的标记

  如果在合并过程中出现了 Checksum 校验错误，且人工介入修复后，可以清理掉该 Checksum 错误的标记，以恢复合并，语句如下。

  - 清理所有用户租户的 Checksum 错误的标记

    ```
    ALTER SYSTEM CLEAR MERGE ERROR TENANT = all_user;
    ```

    或者

    ```
    ALTER SYSTEM CLEAR MERGE ERROR TENANT = all;
    ```

14.3.0.3 说明

OceanBase 数据库从 V4.2.1 版本开始，`TENANT = all_user` 与 `TENANT = all` 语义相同，在需要生效范围为所有用户租户时，推荐使用 `TENANT = all_user`，后续 `TENANT = all` 将废弃不再使用。

  - 清理所有 Meta 租户的 Checksum 错误的标记

    ```
    ALTER SYSTEM CLEAR MERGE ERROR TENANT = all_meta;
    ```

  - 清理指定租户的 Checksum 错误的标记

    ```
    ALTER SYSTEM CLEAR MERGE ERROR TENANT = tenant1，tenant2;
    ```

  其中，`=` 为可选；`tenant1` 、`tenant2` 需要替换为实际的租户名。

14.4 用户租户手动控制合并

合并开始后，用户租户只能控制本租户的合并操作。可以控制的合并操作包括租户级合并和分区级合并。

1. 用户租户的租户管理员登录到数据库。
2. 根据业务实际情况，选择合适的命令控制合并。
   - 暂停本租户的合并

     开始合并后，如果需要暂停本租户的合并，可以执行以下语句。

     ```
     obclient> ALTER SYSTEM SUSPEND MERGE;
     ```

- 恢复合并

  暂停合并后，也可以恢复合并。

  语句如下：

  ```
  obclient> ALTER SYSTEM RESUME MERGE;
  ```

- 清理 Checksum 错误的标记

  如果在合并过程中出现了 Checksum 校验错误，且人工介入修复后，可以清理掉该 Checksum 错误的标记，以恢复合并。

  语句如下：

  ```
  obclient> ALTER SYSTEM CLEAR MERGE ERROR;
  ```

14.5 相关文档

- 自动触发合并
- 定时触发合并
- 手动触发合并
- 查看合并信息

15 查看合并信息

发起合并后，您可以通过视图来查看合并信息。

合并的过程可以通过视图进行查看，通常合并的时间是取决于两次合并之间的数据变化量。两次合并之间的数据变化大，合并的时间会更长。同时，合并的线程数、合并时的集群压力等都影响合并的时间长短。

15.1 系统租户查看所有租户的合并信息

系统租户可以通过视图来观察所有租户的租户级合并信息和分区级合并信息。

1. 使用 root 用户登录集群的 sys 租户。
2. 执行以下语句，查看合并信息。
   - 通过视图 CDB_OB_ZONE_MAJOR_COMPACTION 或 CDB_OB_MAJOR_COMPACTION 查看租户级合并信息。

     视图 CDB_OB_ZONE_MAJOR_COMPACTION 展示了所有租户各个 Zone 的合并过程。

     ```
     obclient> SELECT * FROM oceanbase.CDB_OB_ZONE_MAJOR_COMPACTION;
     ```

     查询结果如下：

     ```
     +-----------+-------+--------------------+--------------------
     +---------------------------+---------------------------+--------+
     | TENANT_ID | ZONE | BROADCAST_SCN | LAST_SCN | LAST_FINISH_TIME |
     START_TIME | STATUS |
     +-----------+-------+--------------------+--------------------
     +---------------------------+---------------------------+--------+
     | 1 | zone1 | 1748368801021617000 | 1748368801021617000 | 2025-05-28 02:04:
     15.351286 | 2025-05-28 02:00:01.051592 | IDLE |
     | 1001 | zone1 | 1748368802924607000 | 1748368802924607000 | 2025-05-28 02:
     02:16.412661 | 2025-05-28 02:00:02.965810 | IDLE |
     | 1002 | zone1 | 1748368802035321000 | 1748368802035321000 | 2025-05-28 02:
     03:25.019173 | 2025-05-28 02:00:02.090687 | IDLE |
     ```

```
| 1003 | zone1 | 1748368802977311000 | 1748368802977311000 | 2025-05-28 02:02:46.565036 | 2025-05-28 02:00:03.009192 | IDLE |
| 1004 | zone1 | 1748368801786323000 | 1748368801786323000 | 2025-05-28 02:03:54.629630 | 2025-05-28 02:00:01.820863 | IDLE |
+----------+-------+---------------------+---------------------+----------------------------+----------------------------+--------+
5 rows in set
```

在执行结果中观察 status 列，部分字段说明如下表所示。

字段	说明
TENANT_ID	租户 ID。
ZONE	Zone 名称。
BROADCAST_SCN	广播的合并版本号。
LAST_SCN	上一次合并的版本号。
LAST_FINISH_TIME	上一次合并结束时间。
START_TIME	合并开始时间。
STATUS	合并状态： ■ IDLE：未在合并中 ■ COMPACTING：正在合并中 ■ VERIFYING：正在校验 Checksum 中

视图 CDB_OB_ZONE_MAJOR_COMPACTION 展示了所有租户的合并全局信息。

```
obclient> SELECT * FROM oceanbase.CDB_OB_MAJOR_COMPACTION;
```

查询结果如下：

```
+----------+---------------------+----------------------------
+---------------------+--------------------+----------------------------
+----------------------------+--------+----------+-------------+------+
| TENANT_ID | FROZEN_SCN | FROZEN_TIME | GLOBAL_BROADCAST_SCN |
LAST_SCN | LAST_FINISH_TIME | START_TIME | STATUS | IS_ERROR | IS_SUSPENDED |
```

```
INFO |
+----------+--------------------+--------------------------
+---------------------+--------------------+--------------------------
+--------------------------+--------+----------+-------------+------+
| 1 | 1748368801021617000 | 2025-05-28 02:00:01.021617 |
1748368801021617000 | 1748368801021617000 | 2025-05-28 02:04:15.380802 |
2025-05-28 02:00:01.042282 | IDLE | NO | NO | |
| 1001 | 1748368802924607000 | 2025-05-28 02:00:02.924607 |
1748368802924607000 | 1748368802924607000 | 2025-05-28 02:02:16.440351 |
2025-05-28 02:00:02.957000 | IDLE | NO | NO | |
| 1002 | 1748368802035321000 | 2025-05-28 02:00:02.035321 |
1748368802035321000 | 1748368802035321000 | 2025-05-28 02:03:25.035666 |
2025-05-28 02:00:02.075224 | IDLE | NO | NO | |
| 1003 | 1748368802977311000 | 2025-05-28 02:00:02.977311 |
1748368802977311000 | 1748368802977311000 | 2025-05-28 02:02:46.592681 |
2025-05-28 02:00:03.003544 | IDLE | NO | NO | |
| 1004 | 1748368801786323000 | 2025-05-28 02:00:01.786323 |
1748368801786323000 | 1748368801786323000 | 2025-05-28 02:03:54.645389 |
2025-05-28 02:00:01.809475 | IDLE | NO | NO | |
+----------+--------------------+--------------------------
+---------------------+--------------------+--------------------------
+--------------------------+--------+----------+-------------+------+
5 rows in set
```

在执行结果中观察 status 列，部分参数的说明如下表所示。

字段	说明
TENANT_ID	租户 ID。
FROZEN_SCN	最近一次合并的版本号
FROZEN_TIME	FROZEN_SCN 的可读时间类型。
GLOBAL_BROADCAST_SCN	全局广播的合并版本。
LAST_SCN	上一个已经完成合并的版本。
LAST_FINISH_TIME	上一次合并结束时间。
START_TIME	合并开始时间。
STATUS	合并状态： ■ IDLE：未在合并中 ■ COMPACTING：正在合并中 ■ VERIFYING：正在校验 Checksum 中
IS_ERROR	■ YES：合并过程存在错误 ■ NO： 合并过程没有报错
IS_SUSPENDED	■ YES：暂停合并 ■ NO： 没有暂停合并
INFO	展示合并信息。

- 通过视图 GV$OB_MERGE_INFO 或 GV$OB_TABLET_COMPACTION_HISTORY 查看分区级合并信息

  视图 GV$OB_MERGE_INFO 展示了 Tablet 级别的合并基础统计信息。

  ```
  obclient [oceanbase]> SELECT * FROM oceanbase.GV$OB_MERGE_INFO WHERE TENANT_ID=1002 AND TABLET_ID=200001 ORDER BY START_TIME DESC LIMIT 10;
  ```

  查询结果如下：

  ```
  +---------------+---------+----------+-------+----------+----------------
  +-------------------+-------------------------+-------------------------
  ```

```
+-----------------+----------+----------------+
| SVR_IP | SVR_PORT | TENANT_ID | LS_ID | TABLET_ID | ACTION | COMPACTION_SCN
| START_TIME | END_TIME | MACRO_BLOCK_COUNT | REUSE_PCT |
PARALLEL_DEGREE |
+---------------+---------+----------+-------+----------+----------------
+-------------------+-------------------------+-------------------------
+-----------------+----------+----------------+
| 172.xx.xxx.xxx | 2882 | 1002 | 1001 | 200001 | MAJOR_MERGE |
1706119204801812283 | 2024-01-25 02:03:03.876629 | 2024-01-25 02:03:
03.929758 | 0 | 0.00 | 1 |
| 172.xx.xxx.xxx | 2882 | 1002 | 1001 | 200001 | MDS_TABLE_MERGE |
1706086731631303497 | 2024-01-24 16:58:52.078436 | 2024-01-24 16:58:
52.080722 | 0 | 0.00 | 0 |
| 172.xx.xxx.xxx | 2882 | 1002 | 1001 | 200001 | MEDIUM_MERGE |
1706086435242718629 | 2024-01-24 16:56:57.297697 | 2024-01-24 16:56:
57.305654 | 0 | 0.00 | 1 |
| 172.xx.xxx.xxx | 2882 | 1002 | 1001 | 200001 | MDS_TABLE_MERGE |
1706086356639492056 | 2024-01-24 16:52:37.067139 | 2024-01-24 16:52:
37.069771 | 0 | 0.00 | 0 |
+---------------+---------+----------+-------+----------+----------------
+-------------------+-------------------------+-------------------------
+-----------------+----------+----------------+
4 rows in set
```

查询结果中，`ACTION` 列中各值的含义如下：

- `MDS_TABLE_MERGE`：将系统的元数据按照 SSTable 的格式持久化到磁盘里。
- `MAJOR_MERGE`：租户级合并
- `MEDIUM_MERGE`：分区级合并
- `MINI_MERGE`：Mini Compaction，将 MemTable 转变成 Mini SSTable。

- MINOR_MERGE：Minor Compaction，多个 Mini SSTable 合成一个新的 Mini SSTable 或者多个 Mini SSTable 与一个 Minor SSTable 合成一个新的 Minor SSTable。
- META_MAJOR_MERGE：一种特殊的 Compaction 类型，是将某个指定时间点之前的数据合成一个 Meta Major SSTable，其数据格式与 Major SSTable 一样，不包含多版本数据和未提交事务数据。

视图 GV$OB_MERGE_INFO 中更多字段的详细说明，请参见 GV$OB_MERGE_INFO。

视图 GV$OB_TABLET_COMPACTION_HISTORY 展示了 Tablet 级的合并的详细历史信息。

```
obclient [oceanbase]> SELECT * FROM oceanbase.
GV$OB_TABLET_COMPACTION_HISTORY WHERE TENANT_ID=1002 AND
TABLET_ID=200001 ORDER BY START_TIME DESC LIMIT 2\G
```

查询结果如下：

```
*************************** 1. row ***************************
SVR_IP: 172.xx.xxx.xxx
SVR_PORT: 2882
TENANT_ID: 1002
LS_ID: 1001
TABLET_ID: 200001
TYPE: MAJOR_MERGE
COMPACTION_SCN: 1748455203748142000
START_TIME: 2025-05-29 02:03:35.771081
FINISH_TIME: 2025-05-29 02:03:35.778086
TASK_ID: YB42AC1E87E0-000635386FA342D5-0-0
OCCUPY_SIZE: 0
MACRO_BLOCK_COUNT: 0
MULTIPLEXED_MACRO_BLOCK_COUNT: 0
NEW_MICRO_COUNT_IN_NEW_MACRO: 0
MULTIPLEXED_MICRO_COUNT_IN_NEW_MACRO: 0
```

```
TOTAL_ROW_COUNT: 0
INCREMENTAL_ROW_COUNT: 0
COMPRESSION_RATIO: 1
NEW_FLUSH_DATA_RATE: 0
PROGRESSIVE_COMPACTION_ROUND: 1
PROGRESSIVE_COMPACTION_NUM: 0
PARALLEL_DEGREE: 1
PARALLEL_INFO: -
PARTICIPANT_TABLE: table_cnt=1,[MAJOR]
snapshot_version=1748368801840485000;
MACRO_ID_LIST:
COMMENTS:
START_CG_ID: 0
END_CG_ID: 0
KEPT_SNAPSHOT: {type:"UNDO_RETENTION", snapshot:1748453603923723000}
MERGE_LEVEL: MICRO_BLOCK_LEVEL
EXEC_MODE: EXEC_MODE_LOCAL
IS_FULL_MERGE: FALSE
IO_COST_TIME_PERCENTAGE: 0
MERGE_REASON: TENANT_MAJOR
BASE_MAJOR_STATUS:
CO_MERGE_TYPE:
MDS_FILTER_INFO:
*************************** 2. row ***************************
SVR_IP: 172.xx.xxx.xxx
SVR_PORT: 2882
TENANT_ID: 1002
LS_ID: 1001
TABLET_ID: 200001
```

```
TYPE: MDS_MINOR_MERGE
COMPACTION_SCN: 1748398151630809003
START_TIME: 2025-05-28 10:09:12.139170
FINISH_TIME: 2025-05-28 10:09:12.145988
TASK_ID: YB42AC1E87E0-000635386FA3413D-0-0
OCCUPY_SIZE: 493
MACRO_BLOCK_COUNT: 1
MULTIPLEXED_MACRO_BLOCK_COUNT: 0
NEW_MICRO_COUNT_IN_NEW_MACRO: 1
MULTIPLEXED_MICRO_COUNT_IN_NEW_MACRO: 0
TOTAL_ROW_COUNT: 1
INCREMENTAL_ROW_COUNT: 1
COMPRESSION_RATIO: 1
NEW_FLUSH_DATA_RATE: 78
PROGRESSIVE_COMPACTION_ROUND: 0
PROGRESSIVE_COMPACTION_NUM: 0
PARALLEL_DEGREE: 1
PARALLEL_INFO: -
PARTICIPANT_TABLE: table_cnt=2,start_scn=1,end_scn=1748398151630809003;
MACRO_ID_LIST: 77871
COMMENTS: comment="cost_mb=2;";
START_CG_ID: 0
END_CG_ID: 0
KEPT_SNAPSHOT:
MERGE_LEVEL: MACRO_BLOCK_LEVEL
EXEC_MODE: EXEC_MODE_LOCAL
IS_FULL_MERGE: FALSE
IO_COST_TIME_PERCENTAGE: 3
MERGE_REASON:
```

```
BASE_MAJOR_STATUS:
CO_MERGE_TYPE:
MDS_FILTER_INFO:
2 rows in set
```

查询结果中，TYPE 列中各值的含义如下：

- MDS_TABLE_MERGE：将系统的元数据按照 SSTable 的格式持久化到磁盘里。
- MAJOR_MERGE：租户级合并
- MEDIUM_MERGE：分区级合并
- MINI_MERGE：Mini Compaction，将 MemTable 转变成 Mini SSTable。
- MINOR_MERGE：Minor Compaction，多个 Mini SSTable 合成一个新的 Mini SSTable 或者多个 Mini SSTable 与一个 Minor SSTable 合成一个新的 Minor SSTable。
- META_MAJOR_MERGE：一种特殊的 Compaction 类型，是将某个指定时间点之前的数据合成一个 Meta Major SSTable，其数据格式与 Major SSTable 一样，不包含多版本数据和未提交事务数据。

视图 GV$OB_TABLET_COMPACTION_HISTORY 中更多字段的详细说明，参见 GV$OB_TABLET_COMPACTION_HISTORY。

15.2 用户租户查看本租户的合并信息

用户租户也可以通过视图来查看本租户的合并信息。

1. 用户租户的租户管理员登录到数据库。
2. 执行以下语句，查看合并信息。

   MySQL 模式
   Oracle 模式

   - 通过视图 DBA_OB_ZONE_MAJOR_COMPACTION 或 DBA_OB_MAJOR_COMPACTION 查看本租户的租户级合并信息。

     视图 DBA_OB_ZONE_MAJOR_COMPACTION 展示了本租户各个 Zone 的合并信息。

     ```
     obclient [oceanbase]> SELECT * FROM oceanbase.
     DBA_OB_ZONE_MAJOR_COMPACTION;
     ```

查询结果的示例如下：

```
+-------+--------------------+--------------------+--------------------------
+---------------------------+--------+
| ZONE | BROADCAST_SCN | LAST_SCN | LAST_FINISH_TIME | START_TIME | STATUS |
+-------+--------------------+--------------------+--------------------------
+---------------------------+--------+
| zone1 | 1748455202538454000 | 1748455202538454000 | 2025-05-29 02:04:
34.174642 | 2025-05-29 02:00:02.679713 | IDLE |
+-------+--------------------+--------------------+--------------------------
+---------------------------+--------+
1 row in set
```

在执行结果中观察 status 列，部分参数的说明如下表所示。

字段	说明
ZONE	Zone 名称。
BROADCAST_SCN	广播的合并版本号。
LAST_SCN	上一次合并的版本号。
LAST_FINISH_TIME	上一次合并结束时间。
START_TIME	合并开始时间。
STATUS	合并状态： ■ IDLE：未在合并中 ■ COMPACTING：正在合并中 ■ VERIFYING：正在校验 Checksum 中

视图 DBA_OB_MAJOR_COMPACTION 展示了本租户的合并全局信息。

```
obclient [oceanbase]> SELECT * FROM oceanbase.
DBA_OB_MAJOR_COMPACTION;
```

查询结果如下：

```
```

```
+--------------------+--------------------------+---------------------
+--------------------+--------------------------+--------------------------
+--------+----------+--------------+------+
| FROZEN_SCN | FROZEN_TIME | GLOBAL_BROADCAST_SCN | LAST_SCN |
LAST_FINISH_TIME | START_TIME | STATUS | IS_ERROR | IS_SUSPENDED | INFO |
+--------------------+--------------------------+---------------------
+--------------------+--------------------------+--------------------------
+--------+----------+--------------+------+
| 1748455202538454000 | 2025-05-29 02:00:02.538454 | 1748455202538454000
| 1748455202538454000 | 2025-05-29 02:04:34.171690 | 2025-05-29 02:00:
02.670090 | IDLE | NO | NO | |
+--------------------+--------------------------+---------------------
+--------------------+--------------------------+--------------------------
+--------+----------+--------------+------+
1 row in set
```

在执行结果中观察 status 列，部分参数的说明如下表所示。

字段	说明
FROZEN_SCN	最近一次合并的版本号
FROZEN_TIME	FROZEN_SCN 的可读时间类型。
GLOBAL_BROADCAST_SCN	全局广播的合并版本。
LAST_SCN	上一个已经完成合并的版本。
LAST_FINISH_TIME	上一次合并结束时间。
START_TIME	合并开始时间。
STATUS	合并状态： ■ IDLE：未在合并中 ■ COMPACTING：正在合并中 ■ VERIFYING：正在校验 Checksum 中
IS_ERROR	■ YES：合并过程存在错误 ■ NO：合并过程没有报错
IS_SUSPENDED	■ YES：暂停合并 ■ NO：没有暂停合并
INFO	展示合并信息。

- 通过视图 GV$OB_TABLET_COMPACTION_HISTORY 查看本租户分区级合并信息。

  视图 GV$OB_TABLET_COMPACTION_HISTORY 展示了 Tablet 级的合并的详细历史信息。

  ```
  obclient [oceanbase]> SELECT * FROM oceanbase.
  GV$OB_TABLET_COMPACTION_HISTORY WHERE TABLET_ID=200001 ORDER BY
  START_TIME DESC LIMIT 2\G
  ```

  查询结果如下：

  ```
  *************************** 1. row ***************************
  SVR_IP: 172.xx.xxx.xxx
  ```

```
SVR_PORT: 2882
TENANT_ID: 1002
LS_ID: 1001
TABLET_ID: 200001
TYPE: MAJOR_MERGE
COMPACTION_SCN: 1748455203748142000
START_TIME: 2025-05-29 02:03:35.771081
FINISH_TIME: 2025-05-29 02:03:35.778086
TASK_ID: YB42AC1E87E0-000635386FA342D5-0-0
OCCUPY_SIZE: 0
MACRO_BLOCK_COUNT: 0
MULTIPLEXED_MACRO_BLOCK_COUNT: 0
NEW_MICRO_COUNT_IN_NEW_MACRO: 0
MULTIPLEXED_MICRO_COUNT_IN_NEW_MACRO: 0
TOTAL_ROW_COUNT: 0
INCREMENTAL_ROW_COUNT: 0
COMPRESSION_RATIO: 1
NEW_FLUSH_DATA_RATE: 0
PROGRESSIVE_COMPACTION_ROUND: 1
PROGRESSIVE_COMPACTION_NUM: 0
PARALLEL_DEGREE: 1
PARALLEL_INFO: -
PARTICIPANT_TABLE: table_cnt=1,[MAJOR]
snapshot_version=1748368801840485000;
MACRO_ID_LIST:
COMMENTS:
START_CG_ID: 0
END_CG_ID: 0
KEPT_SNAPSHOT: {type:"UNDO_RETENTION", snapshot:1748453603923723000}
```

```
MERGE_LEVEL: MICRO_BLOCK_LEVEL
EXEC_MODE: EXEC_MODE_LOCAL
IS_FULL_MERGE: FALSE
IO_COST_TIME_PERCENTAGE: 0
MERGE_REASON: TENANT_MAJOR
BASE_MAJOR_STATUS:
CO_MERGE_TYPE:
MDS_FILTER_INFO:
*************************** 2. row ***************************
SVR_IP: 172.xx.xxx.xxx
SVR_PORT: 2882
TENANT_ID: 1002
LS_ID: 1001
TABLET_ID: 200001
TYPE: MDS_MINOR_MERGE
COMPACTION_SCN: 1748398151630809003
START_TIME: 2025-05-28 10:09:12.139170
FINISH_TIME: 2025-05-28 10:09:12.145988
TASK_ID: YB42AC1E87E0-000635386FA3413D-0-0
OCCUPY_SIZE: 493
MACRO_BLOCK_COUNT: 1
MULTIPLEXED_MACRO_BLOCK_COUNT: 0
NEW_MICRO_COUNT_IN_NEW_MACRO: 1
MULTIPLEXED_MICRO_COUNT_IN_NEW_MACRO: 0
TOTAL_ROW_COUNT: 1
INCREMENTAL_ROW_COUNT: 1
COMPRESSION_RATIO: 1
NEW_FLUSH_DATA_RATE: 78
PROGRESSIVE_COMPACTION_ROUND: 0
```

```
PROGRESSIVE_COMPACTION_NUM: 0
PARALLEL_DEGREE: 1
PARALLEL_INFO: -
PARTICIPANT_TABLE: table_cnt=2,start_scn=1,end_scn=1748398151630809003;
MACRO_ID_LIST: 77871
COMMENTS: comment="cost_mb=2;";
START_CG_ID: 0
END_CG_ID: 0
KEPT_SNAPSHOT:
MERGE_LEVEL: MACRO_BLOCK_LEVEL
EXEC_MODE: EXEC_MODE_LOCAL
IS_FULL_MERGE: FALSE
IO_COST_TIME_PERCENTAGE: 3
MERGE_REASON:
BASE_MAJOR_STATUS:
CO_MERGE_TYPE:
MDS_FILTER_INFO:
2 rows in set
```

查询结果中，`TYPE` 列中各值的含义如下：

- `MDS_TABLE_MERGE`：将系统的元数据按照 SSTable 的格式持久化到磁盘里。
- `MAJOR_MERGE`：租户级合并
- `MEDIUM_MERGE`：分区级合并
- `MINI_MERGE`：Mini Compaction，将 MemTable 转变成 Mini SSTable。
- `MINOR_MERGE`：Minor Compaction，多个 Mini SSTable 合成一个新的 Mini SSTable 或者多个 Mini SSTable 与一个 Minor SSTable 合成一个新的 Minor SSTable。
- `META_MAJOR_MERGE`：一种特殊的 Compaction 类型，是将某个指定时间点之前的数据合成一个 Meta Major SSTable，其数据格式与 Major SSTable 一样，不包含多版本数据和未提交事务数据。

视图 GV$OB_TABLET_COMPACTION_HISTORY 中更多字段的详细说明，参见 GV$OB_TABLET_COMPACTION_HISTORY。

- 通过视图 DBA_OB_ZONE_MAJOR_COMPACTION 或 DBA_OB_MAJOR_COMPACTION 查看本租户的租户级合并信息。

  视图 DBA_OB_ZONE_MAJOR_COMPACTION 展示了本租户各个 Zone 的合并信息。

  ```
  obclient [TEST]> SELECT * FROM SYS.DBA_OB_ZONE_MAJOR_COMPACTION;
  ```

  查询结果如下：

  ```
  +-------+--------------------+----------+-----------------+-----------------
  +------------+
  | ZONE | BROADCAST_SCN | LAST_SCN | LAST_FINISH_TIME | START_TIME | STATUS |
  +-------+--------------------+----------+-----------------+-----------------
  +------------+
  | zone1 | 1748487596765126000 | 1 | 0 | 1748487596921347 | COMPACTING |
  +-------+--------------------+----------+-----------------+-----------------
  +------------+
  1 row in set
  ```

  在执行结果中观察 status 列，部分参数的说明如下表所示。

字段	说明
ZONE	Zone 名称。
BROADCAST_SCN	广播的合并版本号。
LAST_SCN	上一次合并的版本号。
LAST_FINISH_TIME	上一次合并结束时间。
START_TIME	合并开始时间。
STATUS	合并状态： ■ IDLE：未在合并中 ■ COMPACTING：正在合并中 ■ VERIFYING：正在校验 Checksum 中

视图 DBA_OB_MAJOR_COMPACTION 展示了本租户的合并全局信息。

```
obclient [TEST]> SELECT * FROM SYS.DBA_OB_MAJOR_COMPACTION;
```

查询结果如下：

```
+--------------------+---------------------+----------+-----------------
+-----------------+------------+----------+-------------+------+
| FROZEN_SCN | GLOBAL_BROADCAST_SCN | LAST_SCN | LAST_FINISH_TIME |
START_TIME | STATUS | IS_ERROR | IS_SUSPENDED | INFO |
+--------------------+---------------------+----------+-----------------
+-----------------+------------+----------+-------------+------+
| 1748487596765126000 | 1748487596765126000 | 1 | 0 | 1748487596900693 |
COMPACTING | NO | NO | NULL |
+--------------------+---------------------+----------+-----------------
+-----------------+------------+----------+-------------+------+
1 row in set
```

在执行结果中观察 status 列，部分参数的说明如下表所示。

字段	说明
FROZEN_SCN	最新的合并快照。
GLOBAL_BROADCAST_SCN	全局广播的合并版本。
LAST_SCN	上一个已经完成合并的版本。
LAST_FINISH_TIME	上一次合并结束时间。
START_TIME	合并开始时间。
STATUS	合并状态： ■ IDLE：未在合并中 ■ COMPACTING：正在合并中 ■ VERIFYING：正在校验 Checksum 中
IS_ERROR	■ YES：合并过程存在错误 ■ NO：合并过程没有报错
IS_SUSPENDED	■ YES：暂停合并 ■ NO：没有暂停合并
INFO	展示合并信息。

- 通过视图 GV$OB_TABLET_COMPACTION_HISTORY 查看本租户分区级合并信息。

  视图 GV$OB_TABLET_COMPACTION_HISTORY 展示了 Tablet 级的合并的详细历史信息。

  ```
  obclient [TEST]> SELECT * FROM SYS.GV$OB_TABLET_COMPACTION_HISTORY
  WHERE TABLET_ID=200001 AND ROWNUM<=10 ORDER BY START_TIME\G
  ```

  查询结果如下：

  ```
  *************************** 1. row ***************************
  SVR_IP: 172.xx.xxx.xxx
  SVR_PORT: 2882
  TENANT_ID: 1004
  ```

```
LS_ID: 1001
TABLET_ID: 200001
TYPE: MDS_MINOR_MERGE
COMPACTION_SCN: 1748368803414843000
START_TIME: 28-MAY-25 02.00.05.107746 AM
FINISH_TIME: 28-MAY-25 02.00.05.113086 AM
TASK_ID: YB42AC1E87E0-00063538753335FD-0-0
OCCUPY_SIZE: 630
MACRO_BLOCK_COUNT: 1
MULTIPLEXED_MACRO_BLOCK_COUNT: 0
NEW_MICRO_COUNT_IN_NEW_MACRO: 1
MULTIPLEXED_MICRO_COUNT_IN_NEW_MACRO: 0
TOTAL_ROW_COUNT: 2
INCREMENTAL_ROW_COUNT: 2
COMPRESSION_RATIO: 1
NEW_FLUSH_DATA_RATE: 191
PROGRESSIVE_COMPACTION_ROUND: 0
PROGRESSIVE_COMPACTION_NUM: 0
PARALLEL_DEGREE: 1
PARALLEL_INFO: -
PARTICIPANT_TABLE: table_cnt=2,start_scn=1,end_scn=1748368803414843000;
MACRO_ID_LIST: 75689
COMMENTS: comment="cost_mb=2;";
START_CG_ID: 0
END_CG_ID: 0
KEPT_SNAPSHOT: NULL
MERGE_LEVEL: MACRO_BLOCK_LEVEL
EXEC_MODE: EXEC_MODE_LOCAL
IS_FULL_MERGE: FALSE
```

```
IO_COST_TIME_PERCENTAGE: 2
MERGE_REASON: NULL
BASE_MAJOR_STATUS: NULL
CO_MERGE_TYPE: NULL
MDS_FILTER_INFO: NULL
1 row in set
```

查询结果中，`TYPE` 列中各值的含义如下：

- `MDS_TABLE_MERGE`：将系统的元数据按照 SSTable 的格式持久化到磁盘里。
- `MAJOR_MERGE`：租户级合并
- `MEDIUM_MERGE`：分区级合并
- `MINI_MERGE`：Mini Compaction，将 MemTable 转变成 Mini SSTable。
- `MINOR_MERGE`：Minor Compaction，多个 Mini SSTable 合成一个新的 Mini SSTable 或者多个 Mini SSTable 与一个 Minor SSTable 合成一个新的 Minor SSTable。
- `META_MAJOR_MERGE`：一种特殊的 Compaction 类型，是将某个指定时间点之前的数据合成一个 Meta Major SSTable，其数据格式与 Major SSTable 一样，不包含多版本数据和未提交事务数据。

视图 `GV$OB_TABLET_COMPACTION_HISTORY` 中更多字段的详细说明，请参见 GV$OB_TABLET_COMPACTION_HISTORY。

16 修改合并配置

本节主要介绍合并相关的参数及参数的修改方法。

16.1 合并参数

合并相关参数说明如下表所示。

配置项	描述	默认值	取值范围
enable_major_freeze	集群级配置项，表示是否开启合并。	True	• True • False
major_freeze_duty_time	租户级配置项，每天定时合并的时间。	02:00	[00:00,24:00]
major_compact_trigger	租户级配置项，冻结多少次后触发合并。	0	[0,65535]
default_progressive_merge_num	租户级配置项，建表时默认的合并行为。	0	[0, +∞) 其中： • 0 ：表示使用默认值 100 • 1：表示强制执行全量合并，不执行渐进合并 • 大于 1 ：表示发生 Schema 变更时按照指定轮次做渐进合并 #### 16.1.0.1 说明 您也可以在表创建后，通过 `ALTER TABLE table_name SET PROGRESSIVE_MERGE_NUM =0;` 语句修改表的合并行为。
merger_check_interval	租户级配置项，每个 Zone 的合并进度检查间隔。	10m	[10s, 60m]

16.2 通过 SQL 语句修改合并配置

1. 租户管理员登录到数据库。
2. 通过以下 SQL 语句修改合并配置。

   示例如下：

   ```
   obclient> ALTER SYSTEM SET major_freeze_duty_time='03:00';
   ```

16.2.0.1 注意

集群级配置项需要在 `sys` 租户下设置。

3. 修改成功后，可以通过 `SHOW PARAMETERS` 语句查看是否修改成功。

   示例如下：

   ```
   obclient>SHOW PARAMETERS LIKE 'major_freeze_duty_time';
   ```

17 数据压缩概述

OceanBase 数据库的高级压缩功能采用全新设计的行列混合存储结构，以及高效的数据编码技术与一系列综合的数据压缩算法结合的方法，实现了在使用相同后端进行压缩的场景下，存储空间大幅减少。

17.1 通用压缩

通用压缩指的是在压缩算法对数据内部的结构没有了解的情况下，直接对数据块进行压缩。这种压缩往往是根据二进制数据的特征进行编码来减少存储数据的冗余，并且压缩后的数据不能随机访问，压缩和解压都要以整个数据块为单位进行。对数据块的压缩，OceanBase 数据库支持 zlib、snappy、lz4 和 zstd 四种压缩算法，zstd 和 lz4 的压缩等级是 1，zlib 的压缩等级为 6，snappy 使用默认的压缩等级。在 OceanBase 数据库内部对默认16 KB 大小的微块进行压缩的测试中，snappy 和 lz4 压缩速度都比较快，但压缩率比较低，zlib 和 zstd 压缩率比较高但是压缩速度更慢一些。lz4 和 snappy 的压缩率相似但 lz4 压缩解压的速度会更快一些，同样，zstd 的压缩率与 zlib 相似但压缩解压速度都更快。

在 MySQL 模式下，支持用户指定单独选择 zlib、snappy、lz4 或 zstd 压缩算法；在 Oracle 模式下，兼容 Oracle 数据库的压缩选项，仅支持用户选择 lz4 或 zstd 的压缩算法。

17.2 数据编码

在通用压缩的基础上，OceanBase 数据库自研了一套对数据库进行行列混存编码的压缩方法（encoding）。和通用压缩不同，Encoding 建立在压缩算法感知数据块内部数据的格式和语义的基础上。OceanBase 数据库是一个关系型数据库，数据是以表的形式来组织的，表中的每一列都有固定的类型，这就保证了同一列数据在逻辑上存在着一定的相似性；而且在一些场景下，业务的表中相邻的行之间数据也可能会更相似，所以如果将数据按列进行压缩并存储在一起就可以带来更好的压缩效果。因此 OceanBase 数据库引入了一种 encoding 格式的微块。与所有数据逐行序列化到块中的 flat 格式的微块不同，encoding 格式的微块是行列混存的，逻辑上仍然是一组行的数据存在微块里，但微块会按列对数据进行编码，编码后的定长数据存储在微块内部的列存区，部分变长数据还是按行存储在变长区。并且在 encoding 微块中，数据是可以随机访问的，当需要读取微块中的某一行数据时，可以只对这一行数据进行解码，避免了有些解压算法读一部分数据需要解压整个数据块的计算放大；在向量化执行的过程中也可以对指定的列进行解码，降低了投影的开销。

OceanBase 数据库在对数据进行分析的过程中，会选择合适的编码算法。当前，OceanBase 数据库支持多种数据编码技术，比较常见且有效的几种编码方式如下：

- 字典编码

  将基数（Cardinality）较小的数据进行去重，再把去重后的数据建立成字典，而将原来存放数据的地方存为指向特定字典下标的引用。此外，字典中的各数据按类型排序，这样既有利于数据压缩，也可以在计算时直接将谓词下压到字典上，通过二分逻辑完成快速迭代。

- 游程编码（Run-Length Encoding）或 RLE 编码

对于连续相等的数据，例如：100，100，100，120，120，120，150，150，……，将连续的数据去重，仅保留其起始行号和值。

RLE 编码在数据库中通常用于处理基数较低的连续数据，例如，索引前缀和索引后缀等。

- 整型差值编码（Delta Encoding）

  数值型编码，适用于在一个小值域范围内分布的整数型数据，通过计算区间内的最小值和最大值，然后将数据减去最小值后，用更小的位宽进行编码。

- 常量编码

  编码识别一个最常见的数据作为常量，只记录所有不等于这个常量的异常值和它们的行号。

除此以外，OceanBase 数据库还提供了字符串前缀编码（Prefix Encoding）、十六进制编码（Hex Encoding）、列间等值编码（Column Equal Encoding）、列间子串编码（Column SubString Encoding）等多种编码方式。OceanBase 数据库会在合并时根据数据的特点选择合适的编码类型，并计算数据的压缩比，如果发现压缩比不高，会尽快回退，选择其他的编码方式，从而确保数据编码的过程不会影响正常的数据写入性能。如果您对数据的特点非常了解，您也可以在创建表时，手动指定编码方式。

17.3 更多信息

更多数据压缩与数据编码的介绍信息，请参见 压缩与编码。

18 使用数据编码和压缩

OceanBase 数据库支持 MySQL 和 Oracle 两种模式的租户，为了兼容不同类型用户的使用习惯，OceanBase 数据库在不同模式中提供了不同类型的配置方式来使用数据编码和数据压缩。

OceanBase 数据库支持通过 DDL 来对表级别的压缩或编码方式进行配置。

18.1 MySQL 模式下设置数据的压缩与编码方式

在 MySQL 模式下创建表或修改表时，您可以通过 `row_format` 和 `compression` 来设置编码格式及压缩方式。

语句如下：

- 创建表时，指定表的编码格式和压缩方式

  ```
  CREATE TABLE table_name table_definition_list
  ROW_FORMAT [=] row_format COMPRESSION [=] 'compression';
  ```

  更多 `CREATE TABLE` 语句的说明请参见 CREATE TABLE。

- 修改表时，更改表的编码格式和压缩方式

  ```
  ALTER TABLE table_name [alter_table_action_list] [SET] ROW_FORMAT [=]
  row_format COMPRESSION [=] 'compression';
  ```

  更多 `ALTER TABLE` 语句的说明请参见 ALTER TABLE。

`row_format` 的取值如下：

- `REDUNDANT` 和 `COMPACT`：表示数据不会被编码，数据以 flat 格式被保存。
- `DYNAMIC`：表示数据会被编码，数据以 encoding 格式被保存，默认取值为 `DYNAMIC`。
- `COMPRESSED`：表示数据会被编码，数据以 encoding 格式被保存。
- `CONDENSED`：表示数据会被编码，数据以 selective encoding 格式被保存。

  selective encoding 格式是 encoding 格式的子集，仅使用查询更友好的编码方式。

`compression` 的取值如下：

18.1.0.1 说明

由于 zlib 不同小版本的压缩算法可能存在对同一数据进行压缩后，出现压缩结果不一致的情况，为了避免 OceanBase 数据库在升级过程中出现无法恢复副本数据的问题，对于 V4.3.x 版本，从 V4.3.0 版本开始，OceanBase 数据库不再支持 zlib_1.0 压缩算法。

- none：表示数据不压缩。
- lz4_1.0：表示指定压缩算法为 lz4_1.0。
- snappy_1.0：表示指定压缩算法为 snappy_1.0。
- zstd_1.0：表示指定压缩算法为 zstd_1.0。
- zstd_1.3.8：表示指定压缩算法为 zstd_1.3.8。默认取值为 zstd_1.3.8 。
- lz4_1.9.1：表示指定压缩算法为 lz4_1.9.1。

示例如下：

- 创建表时指定编码格式和压缩方式

  执行以下语句，在创建表时指定表的编码格式为 encoding 格式和压缩方式为 zstd_1.0。

```
obclient> CREATE TABLE test (
id INT NOT NULL,
fname VARCHAR(30),
lname VARCHAR(30),
hired DATE NOT NULL DEFAULT '1970-01-01',
separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT NOT NULL,
store_id INT NOT NULL
) ROW_FORMAT=CONDENSED COMPRESSION='zstd_1.0';
```

- 修改表时，更改表的编码格式和压缩方式

  执行以下语句，将表的编码格式修改为 selective encoding 格式，压缩方式修改为 lz4_1.0。

```
obclient> ALTER TABLE test SET ROW_FORMAT = CONDENSED COMPRESSION = 'lz4_1.
0';
```

18.2 Oracle 模式下设置数据的压缩与编码方式

在 Oracle 模式下创建表或修改表时，也可以指定数据的编码格式和压缩方式。

语句如下：

- 创建表时，指定编码格式和压缩方式

  ```
  CREATE TABLE table_name table_definition_list compression;
  ```

  更多 `CREATE TABLE` 语句的说明请参见 CREATE TABLE。

- 修改表时，更改表的编码格式和压缩方式

  ```
  ALTER TABLE table_name [alter_table_action_list] compression;
  ```

  更多 `ALTER TABLE` 语句的说明请参见 ALTER TABLE。

其中，`compression` 的取值如下：

- `NOCOMPRESS`：表示数据不会被编码或压缩，数据以 flat 格式被保存。
- `COMPRESS BASIC`：表示数据不会被编码，数据以 flat 格式被保存，并使用 lz4_1.0 压缩。
- `COMPRESS FOR OLTP`：表示数据不会被编码，数据以 flat 格式被保存，并使用 zstd_1.3.8 压缩。
- `COMPRESS FOR QUERY`：表示数据会被编码，数据以 encoding 格式被保存，并使用 lz4_1.0 压缩。
- `COMPRESS FOR ARCHIVE`：表示数据会被编码，数据以 encoding 格式被保存，并使用 zstd_1.3.8 压缩。默认取值为 `COMPRESS FOR ARCHIVE`。
- `COMPRESS FOR ARCHIVE HIGH`：表示数据会被编码，数据以 encoding 格式被保存，并使用 zstd_1.3.8 压缩。
- `COMPRESS FOR QUERY LOW`：表示数据会被编码，数据以 selective encoding 格式被保存，并使用 lz4_1.0 压缩。

  selective encoding 格式是 encoding 格式的子集，仅使用查询更友好的编码方式。

示例如下：

- 创建表时指定编码格式和压缩方式

  执行以下语句，在创建表时指定表的编码格式为 encoding 格式，压缩方式为 lz4_1.0。

```
obclient> CREATE TABLE test (
id number NOT NULL,
fname VARCHAR2(30),
lname VARCHAR2(30),
hired DATE NOT NULL DEFAULT sysdate,
separated DATE,
job_code INT NOT NULL,
store_id INT NOT NULL
) COMPRESS FOR QUERY;
```

- 修改表时，更改表的编码格式和压缩方式

  执行以下语句，修改表的编码格式为 encoding 格式，同时压缩方式为 zstd_1.3.8。

```
obclient> ALTER TABLE test COMPRESS FOR ARCHIVE;
```

18.3 对已经生成 SSTable 的表更改压缩方式

对已经生成了 SSTable 的表进行压缩选项的变更时，为了避免给一次合并带来太大的 IO 写入压力，需要通过渐进合并的方式逐渐重写全部的微块数据，来完成压缩方式的变更。渐进合并的轮次可以通过在 `ALTER TABLE` 语句中设置 `progressive_merge_num` 的值来完成。

具体方法如下：

1. 租户管理员登录到数据库。
2. 执行以下语句，设置渐进合并轮次。

   `progressive_merge_num` 用于设置表的渐进合并的轮次，默认为 `0`，表示进行增量合并。如果值设置为 `1`，则表示进行全量合并。

   设置渐进合并轮次为 2 次的示例如下：

```
obclient> ALTER TABLE t1 SET progressive_merge_num=2;
```

   设置成功后，后续系统会在每日合并中逐渐完成数据的重写。

18.3.0.1 说明

如果希望尽快完成数据的重写，可以将 `progressive_merge_num` 的值设置为 `1` 后，手动发起一次合并，手动发起合并的操作请参见 手动触发合并。

3. 更改压缩选项。

   MySQL 模式下更改压缩选项的操作请参见本节 **MySQL 模式下设置数据的压缩与编码方式**。

   Oracle 模式下更改压缩选项的操作请参见本节 **Oracle 模式下设置数据的压缩与编码方式**。

19 内存管理概述

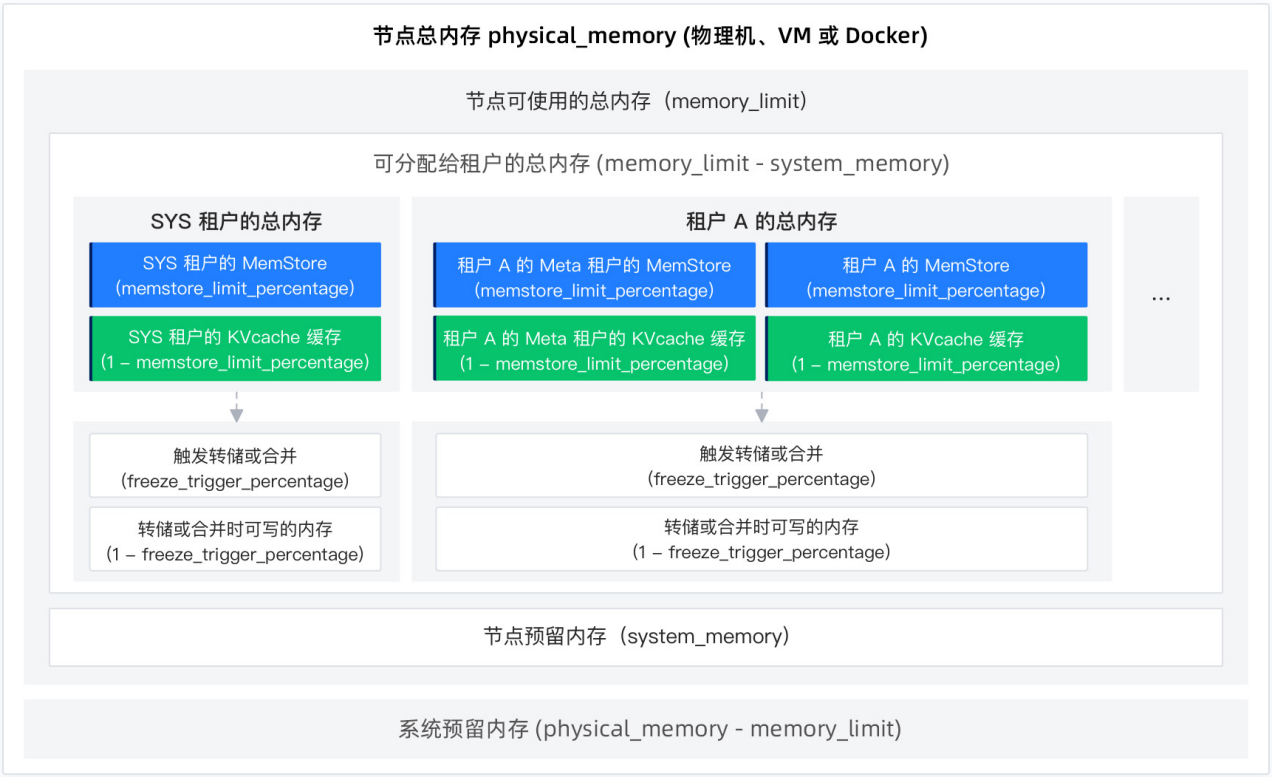
OceanBase 数据库是一个支持多租户架构的准内存级的分布式数据库，这对大容量内存的管理和使用提出了很高的要求。OceanBase 数据库采取占据服务器的大部分内存并进行统一管理的方式来管理内存。

当有新业务需要上线时为其创建一个新租户，其中内存是从租户可分配内存中划分资源，租户 CPU 和内存的大小取决于业务规模。当可分配的 CPU 和内存资源不足时，应水平扩展 OceanBase 集群，为不断扩展的业务规模提供可持续的服务能力。

本章节将分篇介绍内存管理相关的以下内容：

- 内存结构
- 数据库内存上限
- 系统内部内存管理
- 租户内部内存管理
- 执行计划缓存内存管理
- 基于常态化 Memleak 的内存泄露诊断机制
- 查看内存的使用信息
- 常见内存问题

20 内存结构



从上图可以看到，OceanBase 数据库一个节点占用的总内存大小为服务器的一个比例（也可配置为一个绝对值），这其中一部分用于自身系统运行（节点预留内存），一部分用于划分给创建的租户。每个租户等同于传统数据库的一个实例，除了系统租户，其他租户的内存模块组成是一样的。系统租户不存在对应的 Meta 租户，其内存直接分为装载增量数据的 MemStore 以及 KVCache 缓存；其他用户租户，都存在一个对应的 Meta 租户，因此用户租户的内存分为租户本身的内存和对应的 Meta 租户内存，租户本身的内存和对应的 Meta 租户内存又可以分为装载增量数据的 MemStore）以及 KVCache 缓存。

21 数据库内存上限

本文介绍了设置数据库内存上限的方式和相应的使用示例。

21.1 设置自身数据库内存上限方式

OceanBase 数据库提供以下两种方式来设置自身内存的上限：

- 按照计算机器总内存上限的百分比计算自身可以使用的总内存，由 `memory_limit_percentage` 参数配置。
- 直接设置 OceanBase 数据库可用内存的上限，由 `memory_limit` 参数配置。其中 `memory_limit` 参数值为 `0M` 时，使用百分比的配置方式，否则使用绝对值的配置方式。

21.2 使用示例

下面的示例展示了，当在一台 100 GB 的机器上启动一个 OceanBase 数据库实例时，`memory_limit_percentage` 和 `memory_limit` 参数的值是如何影响 OceanBase 数据库的内存上限的。

示例	memory_limit_percentage	memory_limit	OceanBase 数据库内存上限
示例 1	80	0M	80 GB
示例 2	80	90 GB	90 GB

- 示例 1 中，`memory_limit` 为 `0M`，表示使用百分比的配置方式，故以 `memory_limit_percentage` 为准，OceanBase 数据库内存上限为 100 GB*80%=80 GB。
- 示例 2 中，`memory_limit` 为 90 GB，表示使用绝对值的配置方式，故以 `memory_limit` 为准，OceanBase 数据库内存上限为 90 GB。

21.2.0.1 注意

目前主流的 OceanBase 数据库服务器一般内存为 384 GB 或 512 GB，384 GB 内存建议配置为使用机器内存的 80%，512 GB 内存建议配置为使用机器内存的 90%。

22 系统内部内存管理

虽然 OceanBase 数据库支持多租户架构，但是 OceanBase 数据库内存上限中配置的内存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。由于这些共享的资源或功能所使用的内存并不属于任何一个普通租户，故这类内存被归结为系统内部内存。

系统内部可使用的内存上限可通过 `system_memory` 参数来配置，该参数是对系统内部使用内存的预估，实际上限制的是租户可使用的内存上限，即 `memory_limit` - `system_memory` 。

例如，假设 OceanBase 数据库内存上限为 80 GB， `system_memory` 的值为 20 GB，那么可用于租户分配的内存就是剩下的 60 GB。

23 租户内部内存管理

23.1 内存分配

23.1.1 系统租户

系统租户是 OceanBase 数据库的集群管理员为安装 OceanBase 数据库时自动创建。V4.2.1 版本前，系统租户实际可使用的内存上限为 `sys_unit_config.memory_size` 。从 V4.2.1 版本开始，系统租户实际可使用的内存上限为：`hidden_sys_memory + sys_unit_config.memory_size` 。

其中：

- sys_unit_config.memory_size 的值由资源规格 `sys_unit_config` 中的 `memory_size` 决定。`sys_unit_config` 是系统租户默认的资源规格，其默认的内存资源为最小资源。由于系统租户也是一个正常的租户，故可以通过 `ALTER RESOURCE` 语句修改 `sys_unit_config` 中 `memory_size` 的值。
- hidden_sys_memory 的值由隐藏配置项 `_hidden_sys_tenant_memory` 控制。隐藏配置项 `_hidden_sys_tenant_memory` 的默认值为 0G，表示系统会按照一定的规则进行自适应分配 `hidden_sys_memory` 的值。不建议修改隐藏配置项 `_hidden_sys_tenant_memory` 的值。

23.1.2 用户租户

用户租户内存由租户创建 Unit 时其 `memory_size` 的大小定义的， `memory_size` 为用户租户的整体内存大小，其内存包含用户租户本身的内存和其对应 Meta 租户的内存。在创建用户租户时，其允许创建的最小内存由隐藏配置项 `__min_full_resource_pool_memory` 控制，其默认值为 5G，取值范围为 `[1073741824,+∞)` 。

用户租户本身的内存和其对应的 Meta 租户的内存可以分为装载增量数据的 MemStore 以及 KVCache 缓存。关于装载增量数据的 MemStore 的详细内容，参见 不可动态伸缩的内存管理 ，关于 KVCache 缓存的详细内容，参见 可动态伸缩的内存管理。

23.1.3 Meta 租户

Meta 租户的内存资源不支持共享，Meta 租户和用户租户的内存资源需要隔离。默认 Meta 租户占整体租户规格的 10%。为了保证 Meta 租户正常运行，Meta 租户内存资源规格最小为

512M，不设最大值。整体租户内存规格减去 Meta 租户内存规格即为用户租户的内存规格。整体租户规格最小值调整为 1G。下面举例说明：

- 租户规格大于等于 10G 时，Meta 租户和用户租户内存规格比例为 1:9。
- 租户规格大于等于 2G 时，Meta 租户的内存规格固定为 1G，剩余资源给用户租户。
- 租户规格大于等于 1G 并且小于 2G 时，Meta 租户固定分配 512 M，剩余资源给用户租户。

23.2 内存管理

OceanBase 数据库把租户内部的内存总体上分为以下两个部分：

- 不可动态伸缩的内存
- 可动态伸缩的内存——KVCache

其中，不可动态伸缩的内存主要给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（含 row_cache、schema_cache 等）进行管理。

除此之外，还有很多内存组件，包括 Plan Cache（执行计划缓存）、SQL Area（SQL 执行期内存）、选举动作等，都要占用一定量的内存。您可以通过查询 `oceanbase.GV$OB_MEMORY` 视图来获取所有内存组件的使用情况。

23.2.4 不可动态伸缩的内存管理

目前与不可动态伸缩内存相关的配置只有 `memstore_limit_percentage` 和 `ob_sql_work_area_percentage`，前者表示租户的 MemStore 部分最多占租户总内存上限的百分比，后者表示 SQL 阻塞算子工作区内存占用上限。

租户的写入或者更新会增加 MemStore 的内存使用，当租户的 MemStore 部分内存到达上限以后，后续的写入或者更新操作将会被拒绝。

OceanBase 数据库会根据 MemStore 的内存使用比例决定何时进行转储或者合并释放 MemStore 的内存，该比例由配置项 `freeze_trigger_percentage` 控制，表示当 MemStore 内存占用到达其上限的百分比后就进行冻结（转储的前置动作），默认值为租户 MemStore 内存上限的 20%。

23.2.5 可动态伸缩的内存管理

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，当租户内存满时，会优先从 KVCache 中淘汰未被引用的内存来使用。

OceanBase 数据库将绝大多数的 KV 格式的缓存统一在了 KVCache 中进行管理，KVCache 支持动态伸缩、不同类型 Cache 的优先级控制以及智能的淘汰机制。

KVCache 一般不需要配置，特殊场景下可以通过参数控制各种 Cache 的优先级，优先级高的 Cache 比优先级低的 Cache 更容易被保留在 Cache 中。

用于控制不同类型 Cache 的优先级的参数如下表所示。参数值越大表示优先级越高。

参数	含义
tablet_ls_cache_priority	tablet_ls_cache 在缓存系统中的优先级。
index_block_cache_priority	index_block_cache 在缓存系统中的优先级。
user_block_cache_priority	user_block_cache 在缓存系统中的优先级。
user_row_cache_priority	user_row_cache 在缓存系统中的优先级。
bf_cache_priority	Bloom Filter 在缓存系统中的优先级。
fuse_row_cache_priority	fuse_row_cache 在缓存系统中的优先级。
opt_tab_stat_cache_priority	opt_tab_stat_cache 在缓存系统中的优先级。

KVCache 中不同类型的 Cache 的信息可以通过查询 oceanbase.GV$OB_KVCACHE 视图获得。其中 sys 租户和普通租户的重要组成部分略有不同：

- sys 租户上常见的 Cache 种类

类别	说明
schema_cache	存放用户的 Schema 信息，用于提供 SQL 及系统正常运行所依赖的数据库对象的元信息。
tablet_table_cache	缓存 Tablet 中 schema version 和 table id 的对应关系。
vtable_cache	缓存 Table 的 Location 信息。
index_block_cache	缓存微块的 Index，加速微块数据的访问。
user_block_cache	缓存微块数据，由于微块可能通过压缩算法进行压缩，为了提升查询性能，缓存的是解压后的微块数据。
user_row_cache	缓存某一个 Table 中的热点行数据。
bf_cache	缓存为点查结果为空且超过一定次数的宏块建立的 Bloomfilter，用于提高空查询的过滤效率。
fuse_row_cache	缓存行的快照点数据，用于提高点查询性能，并且可以避免因转储、合并导致的缓存失效问题。
opt_table_stat_cache	缓存某一个分区的统计信息，例如行数、宏块数、微块数等。
opt_column_stat_cache	缓存分区中某一列的统计信息，例如空值数、非空值数、最大值、最小值等。

- 普通租户上常见的 Cache 种类

类别	说明
index_block_cache	缓存微块的 Index，加速微块数据的访问。
user_block_cache	缓存微块数据，由于微块可能通过压缩算法进行压缩，为了提升查询性能，缓存的是解压后的微块数据。
user_row_cache	缓存某一个 Table 中的热点行数据。
bf_cache	缓存为点查结果为空且超过一定次数的宏块建立的 Bloomfilter，用于提高空查询的过滤效率。
fuse_row_cache	缓存行的快照点数据，用于提高点查询性能，并且可以避免因转储、合并导致的缓存失效问题。
tmp_block_cache	缓存临时文件，作为写缓冲区以及写缓存。
tmp_page_cache	缓存临时文件，作为读缓存。
opt_table_stat_cache	缓存某一个分区的统计信息，例如行数、宏块数、微块数等。
opt_column_stat_cache	缓存分区中某一列的统计信息，例如空值数、非空值数、最大值、最小值等。

24 执行计划缓存内存管理

执行 EXPLAIN 语句看到的执行计划是预估的，在 Plan Cache 中缓存的执行计划才是真实的。Plan Cache 可以避免硬解析 SQL 语句，当同样的 SQL 请求到 OceanBase 数据库时，会从缓存中直接获取该语句对应的计划，然后直接执行该计划，从而达到加快请求处理的效果。

24.1 与 Plan Cache 相关的配置项

配置项	说明
plan_cache_evict_interval	该配置项用于设置检查执行计划是否需要淘汰的间隔时间，默认值为 5s。

24.2 与 Plan Cache 相关的系统变量

系统变量	说明
ob_plan_cache_percentage	该系统变量用于设置计划缓存可使用内存占租户内存的百分比。计划缓存最多可使用内存（内存上限绝对值）= 租户内存上限 * ob_plan_cache_percentage/100，默认值为 5。
ob_plan_cache_evict_high_percentage	该系统变量用于设置触发计划缓存淘汰的内存大小占内存上限绝对值的百分比。触发计划缓存淘汰的内存大小（淘汰计划的高水位线） = 内存上限绝对值 * ob_plan_cache_evict_high_percentage /100，默认值为 90。
ob_plan_cache_evict_low_percentage	该系统变量用于设置停止淘汰计划缓存的内存大小占内存上限绝对值的百分比。停止淘汰计划缓存的内存大小（淘汰计划的低水位线）= 内存上限绝对值 * ob_plan_cache_evict_low_percentage /100，默认值为 50。

举例来讲，假如一个租户内存大小为 10 GB，`ob_plan_cache_percentage` 的值为 10，`ob_plan_cache_evict_high_percentage` 的值为 90，`ob_plan_cache_evict_low_percentage` 的值为 50。则：

- 计划缓存内存上限绝对值 = 10 GB * 10 / 100 = 1 GB
- 淘汰计划的高水位线 = 1 GB * 90 / 100 = 0.9 GB
- 淘汰计划的低水位线 = 1 GB * 50 / 100 = 0.5 GB

当该租户在某个 OBServer 节点上的计划缓存使用超过 0.9 GB 时，会触发淘汰，且优先淘汰最久未执行的计划。当淘汰到使用内存只有 0.5 GB 时，则停止淘汰。如果淘汰速度没有新计划生成的速度快，则当计划缓存使用内存达到内存上限绝对值 1 GB 时，将不再往计划缓存中添加新计划，直到淘汰后使用的内存小于 1 GB 时才会再次添加新计划到计划缓存中。

24.3 更多信息

更多执行计划缓存的相关的内容，请参见 执行计划缓存。

25 基于常态化 Memleak 的内存泄露诊断机制

OceanBase 数据库通过 Memleak 工具来诊断内存泄漏，其工作原理是记录跟踪模块的内存分配堆栈，如果某一堆栈的累积次数在一定时段里持续增加，则该堆栈对应的内存分配上下文可能发生了内存泄漏。然而，开启 Memleak 的时机一般是内存发生泄漏之后，如果 Memleak 开启后内存不再泄漏，则无法跟踪到发生泄漏的模块，只能等待复现。因此，我们需要使用常态化的 Memleak，实时保存内存分配信息的采样，根据采样的结果分析可能发生内存泄漏的模块。

常态化 Memleak 通过配置项 `_min_malloc_sample_interval` 和 `_max_malloc_sample_interval` 来控制对内存分配信息的采样率。

配置项	描述	取值范围	默认值
_min_malloc_sample_interval	相邻两次采样之间 ob_malloc 的最小次数	[1,10000]	16
_max_malloc_sample_interval	相邻两次采样之间 ob_malloc 的最大次数	[1,10000]	256

通过设置配置项来控制对内存分配信息的采样率可分为三种模式：

- 零采样模式：该模式下，系统会停止记录内存分配信息。

```
obclient> ALTER SYSTEM SET _min_malloc_sample_interval=10000;
Query OK, 0 rows affected

obclient> ALTER SYSTEM SET _max_malloc_sample_interval=10000;
Query OK, 0 rows affected
```

- 全采样模式：该模式下，系统会记录全部的内存分配信息。在此模式下，OceanBase 数据库会出现很大程度的性能回退。

```
obclient> ALTER SYSTEM SET _min_malloc_sample_interval=1;
Query OK, 0 rows affected

obclient> ALTER SYSTEM SET _max_malloc_sample_interval=1;
Query OK, 0 rows affected
```

- 常态化模式：该模式下，系统会根据设置的采样间隔，记录内存分配信息的采样。

```
obclient> ALTER SYSTEM SET _min_malloc_sample_interval=16;
Query OK, 0 rows affected

obclient> ALTER SYSTEM SET _max_malloc_sample_interval=256;
Query OK, 0 rows affected
```

25.0.0.1 注意

- 在 OceanBase 数据库中，名称以 "_" 开头的配置项称为隐藏配置项，仅供开发人员在故障排查或紧急运维时使用。
- 在设置配置项的值时，需要满足如下要求：`_min_malloc_sample_interval <= _max_malloc_sample_interval`。
- 配置项 `_min_malloc_sample_interval` 和 `_max_malloc_sample_interval` 不建议用户随意修改，如果修改不当会导致集群性能变差。

25.1 使用常态化 Memleak 进行内存泄漏诊断

1. 使用 `root` 用户登录到集群的 `sys` 租户。
2. 通过虚拟表 `__all_virtual_malloc_sample_info` 查询怀疑发生内存泄漏的模块的内存分配信息。

   下面以模块 `glibc_malloc` 为例，查询其内存分配信息：

```
obclient [oceanbase]> SELECT * FROM __all_virtual_malloc_sample_info WHERE
mod_name='glibc_malloc' ORDER BY alloc_bytes DESC limit 10;
+-------------+----------+-----------+--------+-------------+-------------------------
-------------------------------------------------------------------------------------
-------------------------------------------------------------+----------+-------------
+-------------+
| svr_ip | svr_port | tenant_id | ctx_id | mod_name | back_trace | ctx_name |
alloc_count | alloc_bytes |
+-------------+----------+-----------+--------+-------------+-------------------------
-------------------------------------------------------------------------------------
```

```
------------------------------------------------------------+---------+------------
+------------+
| xx.xx.xx.xx | 25224 | 500 | 22 | glibc_malloc | 0x5f8bf23 0x5c5a5bd 0x5c52ec3
0x198ed1c6 0x5d788ba 0x1d8e28a5 0x1d8e2919 0x187f31a6 0x188a867a
0x188aa126 0x15c53d3f 0x16b06ae2 0x16b369ef 0x16b32d7e 0x15c8bcf8
0x15c8606a | GLIBC | 3 | 9437232 |
| xx.xx.xx.xx | 25225 | 500 | 22 | glibc_malloc | 0x5f8bf23 0x5c5a5bd 0x5c52ec3
0x198ed1c6 0x5d788ba 0x1d8e28a5 0x1d8e2919 0x187f31a6 0x188a867a
0x188aa126 0x15c53d3f 0x16b06ae2 0x16b369ef 0x16b32d7e 0x15c8bcf8
0x15c8606a | GLIBC | 3 | 9437232 |
| xx.xx.xx.xx | 25226 | 500 | 22 | glibc_malloc | 0x5f8bf23 0x5c5a5bd 0x5c52ec3
0x198ed1c6 0x5d788ba 0x1d8e28a5 0x1d8e2919 0x187f31a6 0x188a867a
0x188aa126 0x15c53d3f 0x16b06ae2 0x16b369ef 0x16b32d7e 0x15c8bcf8
0x15c8606a | GLIBC | 3 | 9437232 |
| xx.xx.xx.xx | 25224 | 500 | 22 | glibc_malloc | 0x5f8bf23 0x5c5a5bd 0x5c52ec3
0x198ed1c6 0x5d788ba 0x19dee05c 0x19e00077 0xb49139a 0x5f6d17f
0x7fc3a6c85445 0x5f6abe1 0x0 0x0 0x0 0x0 0x0 | GLIBC | 1 | 4194320 |
| xx.xx.xx.xx | 25225 | 500 | 22 | glibc_malloc | 0x5f8bf23 0x5c5a5bd 0x5c52ec3
0x198ed1c6 0x5d788ba 0x19dee05c 0x19e00077 0xb49139a 0x5f6d17f
0x7ff8c02b3445 0x5f6abe1 0x0 0x0 0x0 0x0 0x0 | GLIBC | 1 | 4194320 |
| xx.xx.xx.xx | 25226 | 500 | 22 | glibc_malloc | 0x5f8bf23 0x5c5a5bd 0x5c52ec3
0x198ed1c6 0x5d788ba 0x19dee05c 0x19e00077 0xb49139a 0x5f6d17f
0x7f06f1b63445 0x5f6abe1 0x0 0x0 0x0 0x0 0x0 | GLIBC | 1 | 4194320 |
| xx.xx.xx.xx | 25224 | 500 | 22 | glibc_malloc | 0x5f8bf23 0x5c5a5bd 0x5c52ec3
0x198ed1c6 0x5d788ba 0x1d8e28a5 0x1d8e2919 0x187f31a6 0x15c76adf
0xb49e963 0xb4915b6 0x5f6d17f 0x7fc3a6c85445 0x5f6abe1 0x0 0x0 | GLIBC | 1 |
3145744 |
| xx.xx.xx.xx | 25225 | 500 | 22 | glibc_malloc | 0x5f8bf23 0x5c5a5bd 0x5c52ec3
0x198ed1c6 0x5d788ba 0x1d8e28a5 0x1d8e2919 0x187f31a6 0x15c76adf
```

```
0xb49e963 0xb4915b6 0x5f6d17f 0x7ff8c02b3445 0x5f6abe1 0x0 0x0 | GLIBC | 1 |
3145744 |
| xx.xx.xx.xx | 25226 | 500 | 22 | glibc_malloc | 0x5f8bf23 0x5c5a5bd 0x5c52ec3
0x198ed1c6 0x5d788ba 0x1d8e28a5 0x1d8e2919 0x187f31a6 0x15c76adf
0xb49e963 0xb4915b6 0x5f6d17f 0x7f06f1b63445 0x5f6abe1 0x0 0x0 | GLIBC | 1 |
3145744 |
| xx.xx.xx.xx | 25224 | 500 | 22 | glibc_malloc | 0x5f8bf23 0x5c5a5bd 0x5c52ec3
0x198ed1c6 0x5d788ba 0x1d8e28a5 0x1d8e28d9 0x161a7a6b 0x161a7148
0x161aa227 0x64f4472 0x64f003c 0x64b9e68 0x64b1d45 0x6520720 0x651e8db |
GLIBC | 2 | 131120 |
+-------------+----------+-----------+--------+-------------+-------------------------
-------------------------------------------------------------------------------------
-------------------------------------------------------------+----------+-------------
+-------------+
10 rows in set (0.012 sec)
```

通过查询虚拟表，分析得知，`0x5f8bf23 0x5c5a5bd 0x5c52ec3 0x198ed1c6 0x5d788ba 0x1d8e28a5 0x1d8e2919 0x187f31a6 0x188a867a 0x188aa126 0x15c53d3f 0x16b06ae2 0x16b369ef 0x16b32d7e 0x15c8bcf8 0x15c8606a` 这个堆栈的 `alloc_bytes` 最大，怀疑其可能发生了内存泄漏。

字段说明：

列名	类型	描述
svr_ip	varchar: MAX_IP_ADDR_LENGTH	IP 地址
svr_port	bigint(20)	端口号
tenant_id	bigint(20)	租户 ID
ctx_id	bigint(20)	CTX ID
mod_name	varchar: OB_MAX_CHAR_LENGTH	模块名称
back_trace	varchar: DEFAULT_BUF_LENGTH	内存分配的堆栈
ctx_name	varchar: OB_MAX_CHAR_LENGTH	CTX 名称
alloc_count	bigint(20)	内存分配次数
alloc_bytes	bigint(20)	内存分配的总大小

3. 根据虚拟表查询的结果，分析 `alloc_bytes` 较大的堆栈 `back_trace`：

```
$addr2line -pCfe bin/observer 0x5f8bf23 0x5c5a5bd 0x5c52ec3 0x198ed1c6
0x5d788ba 0x1d8e28a5 0x1d8e2919 0x187f31a6 0x188a867a 0x188aa126
0x15c53d3f 0x16b06ae2 0x16b369ef 0x16b32d7e 0x15c8bcf8 0x15c8606a
void* oceanbase::lib::ObTenantCtxAllocator::common_alloc<oceanbase::lib::
ObjectMgr>(long, oceanbase::lib::ObMemAttr const&, oceanbase::lib::
ObTenantCtxAllocator&, oceanbase::lib::ObjectMgr&) at /home/distcc/tmp/.
/deps/oblib/src/lib/alloc/ob_tenant_ctx_allocator.cpp:414 (discriminator 2)
oceanbase::lib::ObTenantCtxAllocator::alloc(long, oceanbase::lib::ObMemAttr
const&) at /home/distcc/tmp/./deps/oblib/src/lib/alloc
/ob_tenant_ctx_allocator.cpp:35
oceanbase::lib::ObMallocAllocator::alloc(long, oceanbase::lib::ObMemAttr
const&) at ??:?
oceanbase::common::ob_malloc(long, oceanbase::lib::ObMemAttr const&) at .
/build_sanity/deps/oblib/src/lib/../../../../deps/oblib/src/lib/allocator
```

```
/ob_malloc.h:39
ob_malloc_hook(unsigned long, void const*) at ??:?
operator new(unsigned long) at ??:?
operator new[](unsigned long, std::nothrow_t const&) at ??:?
oceanbase::share::schema::ObSchemaMgrCache::init(long, oceanbase::share::
schema::ObSchemaMgrCache::Mode) at ./build_sanity/src/share/./src/share
/schema/ob_schema_mgr_cache.cpp:153
oceanbase::share::schema::ObSchemaStore::init(unsigned long, long, long) at .
/build_sanity/src/share/./src/share/schema/ob_schema_store.cpp:29
oceanbase::share::schema::ObSchemaStoreMap::create(unsigned long, long,
long) at ./build_sanity/src/share/./src/share/schema/ob_schema_store.cpp:131
(discriminator 2)
oceanbase::share::schema::ObMultiVersionSchemaService::
init_multi_version_schema_struct(unsigned long) at ./build_sanity/src/share/.
/src/share/schema/ob_multi_version_schema_service.cpp:218 (discriminator 52)
oceanbase::share::schema::ObServerSchemaService::update_schema_mgr
(oceanbase::common::ObISQLClient&, oceanbase::share::schema::
ObRefreshSchemaStatus const&, oceanbase::share::schema::ObSchemaMgr&,
long, oceanbase::share::schema::ObServerSchemaService::AllSchemaKeys&) at .
/build_sanity/src/share/./src/share/schema/ob_server_schema_service.cpp:
4232 (discriminator 54)
oceanbase::share::schema::ObServerSchemaService::
refresh_increment_schema(oceanbase::share::schema::
ObRefreshSchemaStatus const&) at ./build_sanity/src/share/./src/share/schema
/ob_server_schema_service.cpp:5663 (discriminator 4)
oceanbase::share::schema::ObServerSchemaService::refresh_schema
(oceanbase::share::schema::ObRefreshSchemaStatus const&) at ./build_sanity
/src/share/./src/share/schema/ob_server_schema_service.cpp:5288
oceanbase::share::schema::ObMultiVersionSchemaService::
```

```
refresh_tenant_schema(unsigned long) at ??:?
operator() at ./build_sanity/src/share/./src/share/schema
/ob_multi_version_schema_service.cpp:2328 (discriminator 4)
```

用 `addr2line` 命令解析 `alloc_bytes` 最大的堆栈，联系技术支持人员协助分析该堆栈是否发生内存泄漏。

26 查看内存的使用信息

本文介绍了使用虚拟表和日志内容查看内存使用信息的方式。

26.1 视图查询

26.1.1 查询 OBServer 节点总内存

在 sys 租户下，通过查询 oceanbase.gv$sysstat 视图可以查看指定机器的内存占用，示例如下：

```
obclient> SELECT * FROM oceanbase.gv$sysstat WHERE svr_ip = 'xx.xx.xx.xx' AND
svr_port=2882 AND con_id=1 AND name like '%observer memory%';
+--------+----------------+----------+-----------+--------------------------+-------
+-------------+------------+---------+
| CON_ID | SVR_IP | SVR_PORT | STATISTIC# | NAME | CLASS | VALUE | VALUE_TYPE |
STAT_ID |
+--------+----------------+----------+-----------+--------------------------+-------
+-------------+------------+---------+
| 1 | xx.xx.xx.xx | 2882 | 568 | observer memory used size | 64 | 12735815680 |
SET_VALUE | 140008 |
| 1 | xx.xx.xx.xx | 2882 | 569 | observer memory free size | 64 | 224395264 | SET_VALUE |
140009 |
| 1 | xx.xx.xx.xx | 2882 | 571 | observer memory hold size | 64 | 12960210944 |
SET_VALUE | 140011 |
+--------+----------------+----------+-----------+--------------------------+-------
+-------------+------------+---------+
3 rows in set
```

26.1.2 查看各个租户内存使用总量

在 sys 租户下，通过查询 oceanbase.GV$OB_TENANT_MEMORY 视图可以查看所有租户的内存使用总量。

```
obclient> SELECT tenant_id, hold FROM oceanbase.GV$OB_TENANT_MEMORY WHERE
svr_ip = 'xx.xx.xx.xx' AND svr_port=2882 GROUP BY tenant_id;
+-----------+------------+
| tenant_id | hold |
+-----------+------------+
| 1 | 1308491776 |
| 21 | 0 |
| 500 | 8969994240 |
| 506 | 2097152 |
| 507 | 2097152 |
| 508 | 18874368 |
| 509 | 2097152 |
| 510 | 2097152 |
| 512 | 2097152 |
| 999 | 2097152 |
| 1001 | 524156928 |
| 1002 | 593408000 |
| 1003 | 522059776 |
| 1004 | 771678208 |
+-----------+------------+
14 rows in set
```

26.1.3 查看指定租户的内存使用详情

在 `sys` 租户下，通过查询 `oceanbase.GV$OB_MEMORY` 视图，可以查看指定租户的内存使用详情。

```
obclient> SELECT tenant_id, CTX_NAME, hold FROM oceanbase.GV$OB_MEMORY
WHERE svr_ip = 'xx.xx.xx.xx' AND svr_port=2882 AND tenant_id = 1002 GROUP BY
CTX_NAME;
+-----------+-----------------+-----------+
```

```
| tenant_id | CTX_NAME | hold |
+-----------+-----------------+-----------+
| 1002 | APPLY_STATUS | 8128 |
| 1002 | CACHE_MAP_NODE | 16646144 |
| 1002 | ClogGe | 311296 |
| 1002 | ColUsagHashMap | 212736 |
| 1002 | DDLKvMgrObj | 524288 |
| 1002 | DeadLock | 8128 |
| 1002 | DmlStatMap | 212688 |
| 1002 | FlushLog | 65536 |
| 1002 | FlushMeta | 8128 |
| 1002 | GtsTaskQueue | 286720 |
| 1002 | HashBuckLCSta | 1589248 |
| 1002 | HashBuckPlanCac | 794624 |
| 1002 | HashBuckPsCache | 1187840 |
| 1002 | HashBuckPsInfo | 1187840 |
| 1002 | HashNode | 7968 |
| 1002 | HashNodeLCSta | 111104 |
| 1002 | HashNodePlanCac | 55552 |
| 1002 | HashNodePsCache | 7968 |
| 1002 | HashNodePsInfo | 7936 |
| 1002 | IdxBlkTreePath | 89280 |
| 1002 | LobAllocator | 65536 |
| 1002 | LockMemObj | 65536 |
| 1002 | LockWaitMgr | 8128 |
| 1002 | LogApplyStatus | 13568 |
| 1002 | LogGroupBuffer | 83927040 |
| 1002 | LogReplayStatus | 32768 |
| 1002 | LOG_HASH_MAP | 1120 |
```

| 1002 | LSMap | 4249664 |

| 1002 | LSSvr | 269248 |

| 1002 | MatchOffsetMap | 16256 |

| 1002 | Memstore | 64503808 |

| 1002 | MemtableCallbac | 260096 |

| 1002 | MemTblMgrObj | 393216 |

| 1002 | MemTblObj | 1048576 |

| 1002 | MetaAllocator | 14385792 |

| 1002 | MysqlRequesReco | 86261760 |

| 1002 | OmtTenant | 10674448 |

| 1002 | PartTranCtxMgr | 802816 |

| 1002 | PlanBaselineMgr | 96 |

| 1002 | PoolFreeList | 5250880 |

| 1002 | QueryAllocator | 16256 |

| 1002 | REPLAY_STATUS | 8128 |

| 1002 | SqlDtl | 3907584 |

| 1002 | SqlMemMgr | 24272 |

| 1002 | SqlPhyPlan | 143297472 |

| 1002 | SqlPhyPlObj | 1099840 |

| 1002 | SqlPlanCache | 7868160 |

| 1002 | SqlPlanMon | 5021696 |

| 1002 | SqlPsCache | 4144 |

| 1002 | SSTblObj | 2097152 |

| 1002 | TabletObj | 1638400 |

| 1002 | thread_factor | 32768 |

| 1002 | TxCtxMemObj | 65536 |

| 1002 | TxDataMemObj | 65536 |

| 1002 | TX_DATA_TABLE | 1815104 |

```
+----------+----------------+----------+
55 rows in set
```

26.1.4 查看某个内存模块中内存的使用详情

在 sys 租户下，通过查询 oceanbase.GV$OB_MEMORY，可以查看某个内存模块中内存的使用详情。也可以指定机器、租户等信息来查询。

```
obclient> SELECT * FROM oceanbase.GV$OB_MEMORY WHERE svr_ip = 'xx.xx.xx.xx'
AND svr_port=2882 AND tenant_id = 1002 ORDER BY hold desc limit 10;
+----------+---------------+---------+-----------------+----------------+-------
+-----------+-----------+
| TENANT_ID | SVR_IP | SVR_PORT | CTX_NAME | MOD_NAME | COUNT | HOLD | USED |
+----------+---------------+---------+-----------------+----------------+-------
+-----------+-----------+
| 1002 | xx.xx.xx.xx | 2882 | KVSTORE_CACHE_ID | KvstorCacheMb | 498 | 1044381696 | 0 |
| 1002 | xx.xx.xx.xx | 2882 | PLAN_CACHE_CTX_ID | SqlPhyPlan | 7069 | 135522752 |
120440953 |
| 1002 | xx.xx.xx.xx | 2882 | DEFAULT_CTX_ID | MysqlRequesReco | 11 | 100827136 |
100797440 |
| 1002 | xx.xx.xx.xx | 2882 | DEFAULT_CTX_ID | IoControl | 37 | 100143104 | 100086048 |
| 1002 | xx.xx.xx.xx | 2882 | DEFAULT_CTX_ID | LogGroupBuffer | 2 | 83927040 | 83886080
|
| 1002 | xx.xx.xx.xx | 2882 | CO_STACK | CoStack | 156 | 79233024 | 79082016 |
| 1002 | xx.xx.xx.xx | 2882 | DEFAULT_CTX_ID | ServerObjecPool | 1 | 63983616 |
63963136 |
| 1002 | xx.xx.xx.xx | 2882 | MEMSTORE_CTX_ID | Memstore | 21 | 43696128 | 43676136 |
| 1002 | xx.xx.xx.xx | 2882 | DEFAULT_CTX_ID | SqlDtl | 191 | 23449216 | 21915426 |
| 1002 | xx.xx.xx.xx | 2882 | DEFAULT_CTX_ID | SqlPlanManger | 54 | 23281664 |
22815744 |
+----------+---------------+---------+-----------------+----------------+-------
```

```
+-----------+-----------+
10 rows in set
```

26.2 日志查询

26.2.5 查询 OBServer 节点总内存

1. 登录 OBServer 节点所在的机器。
2. 执行以下命令，进入日志文件所在的目录。

   ```
   cd ~/oceanbase/log
   ```

3. 执行以下命令，查询 OBServer 节点总内存使用情况。

   ```
   grep CHUNK_MGR observer.log
   ```

   查询结果如下：

   ```
   [2022-07-14 09:29:26.243252] INFO [STORAGE] print_tenant_usage
   (ob_tenant_memory_printer.cpp:121) [104667][ServerGTimer][T0][Y0-
   0000000000000000-0-0] [lt=6] [CHUNK_MGR] free=42 pushes=2504440
   pops=2504398 limit= 85,899,345,920 hold= 10,254,864,384 total_hold=
   10,655,629,312 used= 10,166,784,000 freelist_hold= 88,080,384 maps= 159,816
   unmaps= 156,719 large_maps= 156,935 large_unmaps= 156,719 memalign=0
   virtual_memory_used= 11,851,849,728
   [2022-07-14 09:29:36.309803] INFO [STORAGE] print_tenant_usage
   (ob_tenant_memory_printer.cpp:121) [104667][ServerGTimer][T0][Y0-
   0000000000000000-0-0] [lt=11] [CHUNK_MGR] free=39 pushes=2504609
   pops=2504570 limit= 85,899,345,920 hold= 10,254,864,384 total_hold=
   10,655,629,312 used= 10,173,075,456 freelist_hold= 81,788,928 maps= 159,826
   unmaps= 156,729 large_maps= 156,945 large_unmaps= 156,729 memalign=0
   virtual_memory_used= 11,851,849,728
   [2022-07-14 09:29:46.376095] INFO [STORAGE] print_tenant_usage
   (ob_tenant_memory_printer.cpp:121) [104667][ServerGTimer][T0][Y0-
   ```

```
0000000000000000-0-0] [lt=6] [CHUNK_MGR] free=36 pushes=2504774
pops=2504738 limit= 85,899,345,920 hold= 10,254,864,384 total_hold=
10,655,629,312 used= 10,179,366,912 freelist_hold= 75,497,472 maps= 159,836
unmaps= 156,739 large_maps= 156,955 large_unmaps= 156,739 memalign=0
virtual_memory_used= 11,851,849,728
[2022-07-14 09:29:56.455361] INFO [STORAGE] print_tenant_usage
(ob_tenant_memory_printer.cpp:121) [104667][ServerGTimer][T0][Y0-
0000000000000000-0-0] [lt=9] [CHUNK_MGR] free=33 pushes=2504938
pops=2504905 limit= 85,899,345,920 hold= 10,254,864,384 total_hold=
10,655,629,312 used= 10,185,658,368 freelist_hold= 69,206,016 maps= 159,846
unmaps= 156,749 large_maps= 156,965 large_unmaps= 156,749 memalign=0
virtual_memory_used= 11,851,849,728
[2022-07-14 09:30:06.520032] INFO [STORAGE] print_tenant_usage
(ob_tenant_memory_printer.cpp:121) [104667][ServerGTimer][T0][Y0-
0000000000000000-0-0] [lt=7] [CHUNK_MGR] free=32 pushes=2505099
pops=2505067 limit= 85,899,345,920 hold= 10,254,864,384 total_hold=
10,655,629,312 used= 10,187,755,520 freelist_hold= 67,108,864 maps= 159,856
unmaps= 156,759 large_maps= 156,975 large_unmaps= 156,759 memalign=0
virtual_memory_used= 11,851,849,728
[2022-07-14 09:30:16.586924] INFO [STORAGE] print_tenant_usage
(ob_tenant_memory_printer.cpp:121) [104667][ServerGTimer][T0][Y0-
0000000000000000-0-0] [lt=8] [CHUNK_MGR] free=32 pushes=2505261
pops=2505229 limit= 85,899,345,920 hold= 10,254,864,384 total_hold=
10,655,629,312 used= 10,187,755,520 freelist_hold= 67,108,864 maps= 159,866
unmaps= 156,769 large_maps= 156,985 large_unmaps= 156,769 memalign=0
virtual_memory_used= 11,851,849,728
```

日志信息中的部分参数说明如下表所示。

参数	说明
limit	OBServer 节点总内存限制
hold	OBServer 节点从操作系统获取到的所有的内存
used	OBServer 节点实际使用到的内存（包括 Cache）
freelist_hold	OBServer 节点未使用的内存

26.2.6 查看各个租户的内存使用总量

1. 登录 OBServer 节点所在的机器。
2. 执行以下命令，进入日志文件所在的目录。

```
cd ~/oceanbase/log
```

3. 执行以下命令，查看各租户的内存使用总量。

```
grep ob_malloc_allocator.cpp observer.log
```

查询的部分结果如下：

```
[2022-07-14 09:29:26.241149] INFO [LIB] operator() (ob_malloc_allocator.cpp:
387) [104667][ServerGTimer][T0][Y0-0000000000000000-0-0] [lt=8] [MEMORY]
tenant: 500, limit: 9,223,372,036,854,775,807 hold: 6,765,867,008 rpc_hold: 0
cache_hold: 0 cache_used: 0 cache_item_count: 0
[2022-07-14 09:29:26.241729] INFO [LIB] operator() (ob_malloc_allocator.cpp:
387) [104667][ServerGTimer][T1][Y0-0000000000000000-0-0] [lt=10] [MEMORY]
tenant: 1, limit: 17,179,869,184 hold: 1,170,079,744 rpc_hold: 0 cache_hold:
144,703,488 cache_used: 144,703,488 cache_item_count: 69
[2022-07-14 09:29:26.242025] INFO [LIB] operator() (ob_malloc_allocator.cpp:
387) [104667][ServerGTimer][T1001][Y0-0000000000000000-0-0] [lt=5] [MEMORY]
tenant: 1001, limit: 1,073,741,824 hold: 511,574,016 rpc_hold: 0 cache_hold:
218,103,808 cache_used: 218,103,808 cache_item_count: 104
[2022-07-14 09:29:26.242321] INFO [LIB] operator() (ob_malloc_allocator.cpp:
387) [104667][ServerGTimer][T1002][Y0-0000000000000000-0-0] [lt=8] [MEMORY]
```

```
tenant: 1002, limit: 6,442,450,944 hold: 534,687,744 rpc_hold: 0 cache_hold:
44,040,192 cache_used: 44,040,192 cache_item_count: 21
[2022-07-14 09:29:26.242625] INFO [LIB] operator() (ob_malloc_allocator.cpp:
387) [104667][ServerGTimer][T1003][Y0-0000000000000000-0-0] [lt=4] [MEMORY]
tenant: 1003, limit: 1,073,741,824 hold: 511,574,016 rpc_hold: 0 cache_hold:
224,395,264 cache_used: 224,395,264 cache_item_count: 107
[2022-07-14 09:29:26.242901] INFO [LIB] operator() (ob_malloc_allocator.cpp:
387) [104667][ServerGTimer][T1004][Y0-0000000000000000-0-0] [lt=8] [MEMORY]
tenant: 1004, limit: 6,442,450,944 hold: 633,266,176 rpc_hold: 0 cache_hold:
65,011,712 cache_used: 65,011,712 cache_item_count: 31
[2022-07-14 09:29:36.307729] INFO [LIB] operator() (ob_malloc_allocator.cpp:
387) [104667][ServerGTimer][T0][Y0-0000000000000000-0-0] [lt=10] [MEMORY]
tenant: 500, limit: 9,223,372,036,854,775,807 hold: 6,765,867,008 rpc_hold: 0
cache_hold: 0 cache_used: 0 cache_item_count: 0
```

26.2.7 查看某个租户内存使用详情

1. 登录 OBServer 节点所在的机器。
2. 执行以下命令，进入日志文件所在的目录。

```
cd ~/oceanbase/log
```

3. 进行以下操作，查看 `tenant_id` 为 `1002` 的租户的内存使用总量。
   a. 使用 Vim 打开 `observer.log` 。
   b. 搜索 `ob_malloc_allocator.cpp.*1002` 。

```
[MEMORY] ctx_id= DEFAULT_CTX_ID hold_bytes= 258,838,528 limit=
9,223,372,036,854,775,807
[MEMORY] ctx_id= MEMSTORE_CTX_ID hold_bytes= 67,108,864 limit=
9,223,372,036,854,775,807
[MEMORY] ctx_id= TRANS_CTX_MGR_ID hold_bytes= 2,097,152 limit=
9,223,372,036,854,775,807
```

```
[MEMORY] ctx_id= PLAN_CACHE_CTX_ID hold_bytes= 132,120,576 limit=
9,223,372,036,854,775,807
[MEMORY] ctx_id= WORK_AREA hold_bytes= 4,194,304 limit= 322,122,545
[MEMORY] ctx_id= META_OBJ_CTX_ID hold_bytes= 22,093,824 limit= 644,245,090
[MEMORY] ctx_id= TX_CALLBACK_CTX_ID hold_bytes= 2,097,152 limit=
9,223,372,036,854,775,807
[MEMORY] ctx_id= LOB_CTX_ID hold_bytes= 2,097,152 limit=
9,223,372,036,854,775,807
```

26.2.8 查看某个内存模块的内存使用详情

1. 登录 OBServer 节点所在的机器。
2. 执行以下命令，进入日志文件所在的目录。

   ```
   cd ~/oceanbase/log
   ```

3. 进行以下操作，查看 `tenant_id` 为 `1002` 的租户的 `DEFAULT_CTX_ID` 模块的内存使用详情。
   a. 使用 Vim 打开 `observer.log` 。
   b. 搜索 `MEMORY.*1002.*DEFAULT_CTX_ID` 。

   ```
   [MEMORY] tenant_id= 1002 ctx_id= DEFAULT_CTX_ID hold= 258,838,528 used=
   216,411,808 limit= 9,223,372,036,854,775,807
   [MEMORY] idle_size= 0 free_size= 0
   [MEMORY] wash_related_chunks= 0 washed_blocks= 0 washed_size= 0
   [MEMORY] hold= 86,261,760 used= 86,239,232 count= 4 avg_used= 21,559,808
   mod=MysqlRequesReco
   [MEMORY] hold= 83,927,040 used= 83,886,080 count= 2 avg_used= 41,943,040
   mod=LogGroupBuffer
   [MEMORY] hold= 16,646,144 used= 16,637,952 count= 8 avg_used= 2,079,744
   mod=CACHE_MAP_NODE
   [MEMORY] hold= 10,674,448 used= 10,631,028 count= 42 avg_used= 253,119
   ```

```
mod=OmtTenant
[MEMORY] hold= 5,021,696 used= 4,977,920 count= 3 avg_used= 1,659,306
mod=SqlPlanMon
[MEMORY] hold= 4,249,664 used= 4,229,120 count= 2 avg_used= 2,114,560
mod=LSMap
[MEMORY] hold= 1,589,248 used= 1,573,024 count= 2 avg_used= 786,512
mod=HashBuckLCSta
[MEMORY] hold= 1,571,264 used= 1,558,656 count= 197 avg_used= 7,911
mod=TX_DATA_TABLE
[MEMORY] hold= 1,548,288 used= 1,383,795 count= 21 avg_used= 65,895
mod=SqlDtl
[MEMORY] hold= 1,187,840 used= 1,179,768 count= 1 avg_used= 1,179,768
mod=HashBuckPsInfo
[MEMORY] hold= 1,187,840 used= 1,179,768 count= 1 avg_used= 1,179,768
mod=HashBuckPsCache
[MEMORY] hold= 794,624 used= 786,512 count= 1 avg_used= 786,512
mod=HashBuckPlanCac
```

27 常见内存问题

27.1 问题一：超过租户内存限制怎么办?

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

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 MemStore 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。运行下述语句查看当前内存使用情况：

```
obclient> SELECT /*+ READ_CONSISTENCY(WEAK),query_timeout(100000000) */
TENANT_ID,SVR_IP,
round(ACTIVE_SPAN/1024/1024/1024,2) ACTIVE_GB,
round(MEMSTORE_USED/1024/1024/1024,2) TOTAL_GB,
round(FREEZE_TRIGGER/1024/1024/1024,2) FREEZE_TRIGGER_GB,
round(MEMSTORE_USED/FREEZE_TRIGGER*100,2) percent_trigger,
round(MEMSTORE_LIMIT/1024/1024/1024,2) MEM_LIMIT_GB
FROM oceanbase.GV$OB_MEMSTORE
WHERE tenant_id >1000 OR TENANT_ID=1
ORDER BY tenant_id,TOTAL_GB DESC;
```

该问题的紧急处理措施是增加租户内存，具体操作请联系技术支持人员协助处理。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；如果 MemStore 内存未超限，运行下述语句判断是哪个内存模块超限：

```
obclient> SELECT tenant_id, svr_ip, sum(hold) module_sum
FROM oceanbase.GV$OB_MEMORY
WHERE tenant_id>1000 AND hold<>0 AND
CTX_NAME NOT IN ('KVSTORE_CACHE_ID','MEMSTORE_CTX_ID')
ORDER BY tenant_id, module_sum DESC;
```

内存模块超限的判断标准是 module_sum > （租户 memory_size - 租户 MemStore）。模块内存超限，可能需要先调整单个模块的内存，例如，调整 ob_sql_work_area_percentage 的值，如果租户内存过小，也需要增加租户内存。

系统变量名	描述	默认值	生效范围
ob_sql_work_area_percentage	租户工作区内存，是 SQL 排序等阻塞性算子使用的内存。	5%	Global

27.2 问题二：PLANCACHE 命中率低于 90% 怎么办?

如果是 OLTP 系统 PLANCACHE 命中率应不低于 90%，运行下述语句查看 PLANCACHE 命中率：

```
obclient> SELECT hit_count,executions,(hit_count/executions) as hit_ratio
FROM V$OB_PLAN_CACHE_PLAN_STAT
WHERE (hit_count/executions) < 0.9;

obclient> SELECT hit_count,executions,(hit_count/executions) AS hit_ratio
FROM V$OB_PLAN_CACHE_PLAN_STAT
WHERE (hit_count/executions) < 0.9 AND executions > 1000;
```

寻找是否有相似语句，例如 in 或 not in 后面的参数个数随机，导致大量浪费；如果不是上述情况，可能是因为业务量或会话激增导致内存不足，需要调整租户内存大小。

27.3 问题三：日志中有 fail to allocate memory 或 allocate memory fail 等信息怎么办?

日志中会包含 tenant_id （租户编号）及 context （内存模块）信息，可以通过以下语句查询具体的内存模块信息：

```
obclient> SELECT * FROM oceanbase.GV$OB_MEMORY WHERE CTX_NAME =xxx AND
tenant_id = xxx;
```

从问题一中可知，如果模块内存超限，需要先调整单独模块的内存。如果租户内存过小，需要增加租户内存。具体操作请联系技术支持人员协助处理。

27.4 问题四：500 租户内存超限

`tenant_id =500` 的租户是 OB 内部租户，简称 500 租户。正常情况下，该报错会多次出现，最可能是机器系统内存被耗光或当前机器剩余内存已被预分配，用户无法再扩展。

这种报错需要查看内存被占用情况，500 租户的内存使用量没有被 `V$OB_MEMORY` 统计，需要查询 `oceanbase.GV$OB_MEMORY` 表，同时排查系统中内存的使用情况。如果是机器系统内存被耗光，判断是否为系统偶发情况。如果是，且同时没有业务影响，可等系统自行释放内存；如果有大的业务影响或系统正常进程活动导致内存被占用以及内存被预分配等情况，可考虑增加内存，具体操作请联系技术支持人员协助处理。

```
SELECT * FROM oceanbase.GV$OB_MEMORY WHERE tenant_id=500;
//排查系统内存使用命令
```

28 OceanBase 线程简介

OceanBase 数据库是单进程的，在 OceanBase 数据库中，有一部分线程是用于维持 OceanBase 数据库分布式数据库服务的系统线程，还有一部分是用来处理数据库负载的 Worker 线程。

28.1 OceanBase 数据库的线程分类

整体上，OceanBase 数据库的线程可以分成以下几类：

- 租户线程：负责租户请求处理的线程，例如，租户处理 SQL 和事务请求的线程, 命名规则为 `TNT_L<n>_<tenant id>` 。
- 系统线程：
  - net io：处理网络 IO 的线程。
  - disk io：处理磁盘 IO 的线程。
  - dag 线程：用于 Partition 的转储、合并、迁移等任务的执行。
  - clog writer：写 clog 的线程。
  - election worker：选举线程。
  - misc timer：包括多个后台定时器线程，主要负责清理资源。
  - 除此之外，还有一批 Root Server 专有的线程，这里不再做细分。
  - 最后还有一些特定用途的后台线程，在进一步深入了解 OceanBase 数据库之前，可以先忽略。

其中，租户线程是每个租户独有负责执行该租户的请求；系统线程负责保证这一节点上 observer 进程的正常运行、请求处理，是多租户共享的。

28.2 更多信息

关于 OceanBase 数据库线程的更多详情，请参见 线程简介。

29 OceanBase 数据库多租户线程

本文主要介绍了 OceanBase 数据库多租户线程的实现原理。

OceanBase 数据库是一个支持多租户架构的分布式数据库。当 OBServer 节点在处理来自不同租户的请求时，需要依靠租户独有的线程进行请求处理，最终返回给客户端（例如，OBClient 或者是由应用端通过 ODP 返回的结果）。在这个过程中，OceanBase 数据库的后台系统线程运行着系统任务和多租户共享任务。所有的线程的健康运行为 OBServer 的稳定运行以及可靠高可用的数据服务提供基础支撑。为了便于解释 OceanBase 数据库是如何在多租户线程架构下进行请求处理的，我们对 OBServer 收到的三类基本的租户请求进行了详细地介绍。

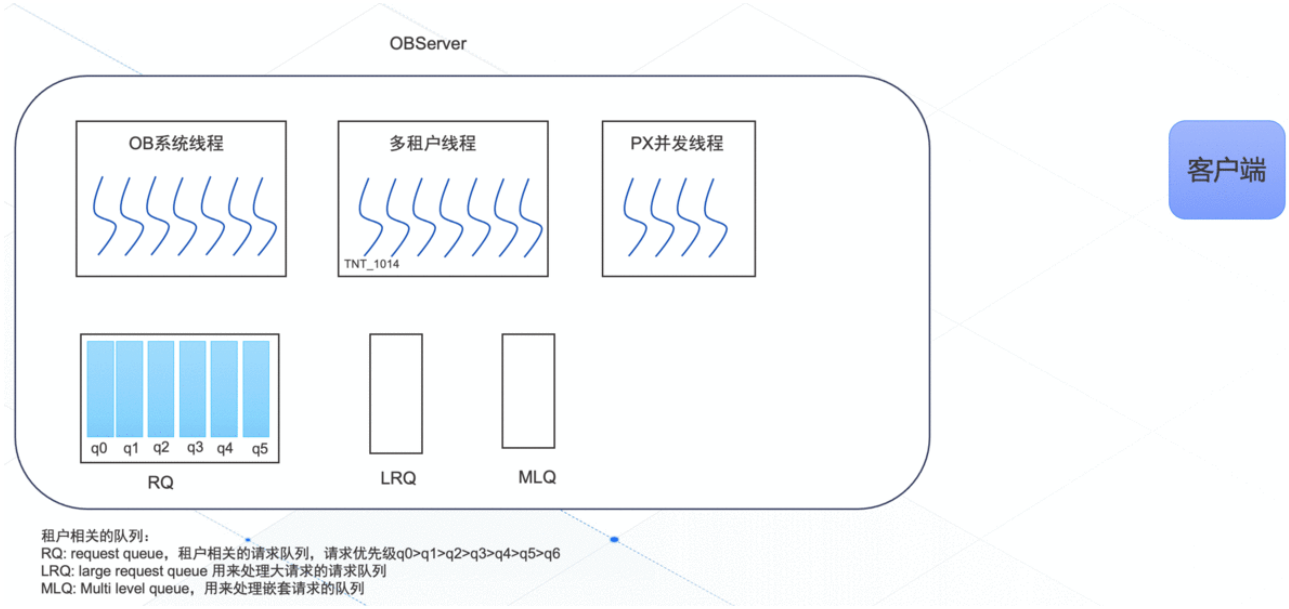
29.1 使用场景

29.1.1 场景一

当 OBServer 节点收到客户端对租户 A（租户 ID 为 `1014`）的 SQL 请求，该 SQL 本身是一个 OLTP 型 SQL，只需要由一个租户线程的本地执行就可以完成。请求被执行过程如下：

1. 客户端发起请求 req1 至 OBServer 节点。
2. req1 进入到租户请求队列，在 OBServer 节点中，此类 SQL 请求会进入到租户请求队列的 queue4。
3. 多租户的工作线程会巡检请求队列，当租户 `1014` 的租户线程（ TNT_1014）发现队列中有需要处理的新线程，将 req1 从队列中取出。
4. TNT_1014 进行对 req1 的执行。
5. TNT_1014 将执行的结果和返回值返回客户端。

OceanBase 数据库多租户线程处理本地 OLTP 型 SQL 的过程如下图所示。

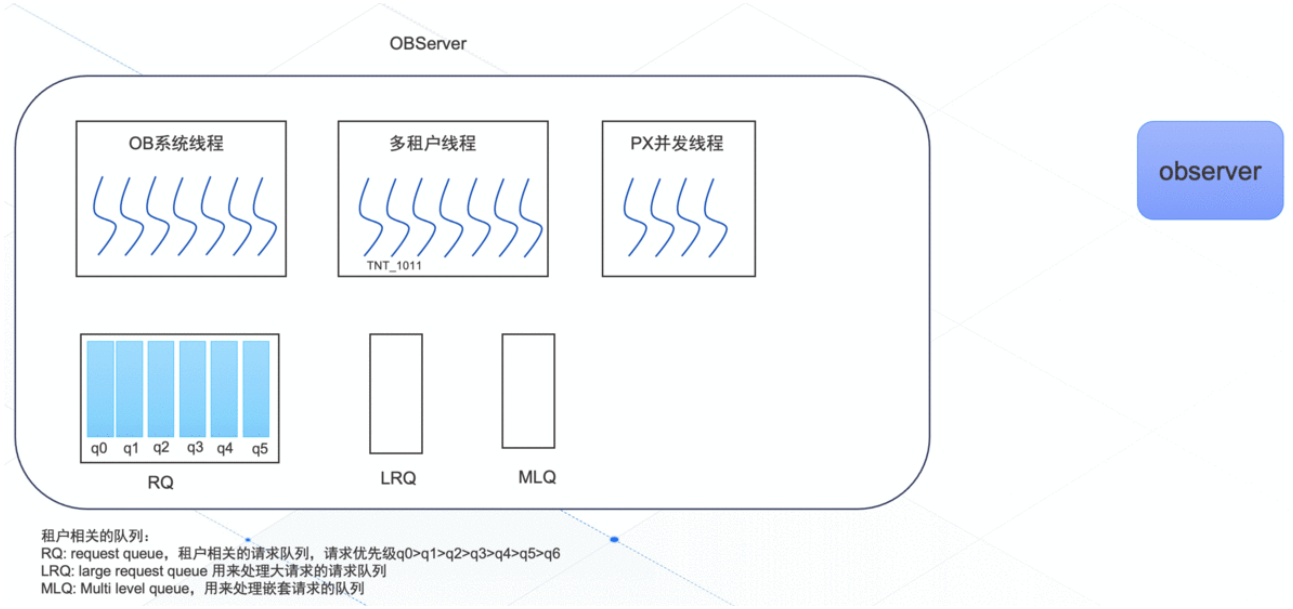


29.1.2 场景二

当 OBServer 节点收到来自租户 B（租户 ID 为 1011 ）的另一台 OBServer 节点发来的 RPC 请求（例如，分布式的 OLTP 型 SQL请求），该请求需要由该租户线程进行执行。这个 RPC 请求被执行的过程如下：

1. 租户中另一个 OBServer 节点作为 RPC 客户端发起 RPC 请求 req1 至当前的 OBServer 节点。
2. req1 是一个 OLTP 的 SQL请求，req1 请求进入到 OBServer 节点的租户请求队列 queue2。
3. 多租户的工作线程会巡检请求队列，当租户 1011 的租户线程（TNT_1011）发现队列中有需要处理的新请求，将 req1 从队列中取出。
4. TNT_1011 进行对 req1 的执行。
5. TNT_1011 将执行的结果和返回值返回给远端的 OBServer 节点。

OceanBase 数据库多租户线程处理分布式 SQL RPC 请求的过程如下图所示。

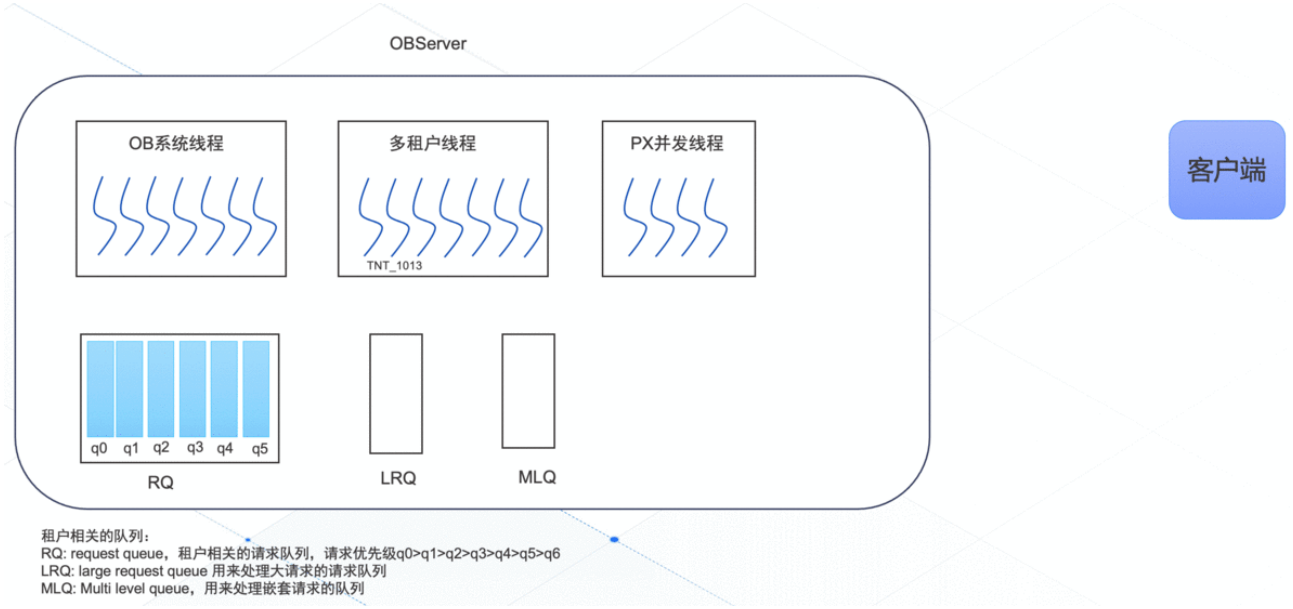


29.1.3 场景三

当 OBServer 节点收到客户端对租户 C（租户 ID 为 `1013`）的 PX SQL 请求，且所有的 SQL 访问都在本地执行。请求被执行的过程如下：

1. 客户端发起请求 req1 至 OBServer 节点。
2. req1 进入到租户请求队列，在 OceanBase 数据库中，普通 SQL 会进入到租户请求队列的 queue4。
3. 多租户的工作线程会巡检请求队列，当租户 `1013` 的租户线程（TNT_1013）发现队列中有需要处理的新请求，将 req1 从队列中取出。
4. 租户线程会作为 Local Scheduler 驱动 PX 线程执行任务。
5. 执行完后，将 req1 的执行的结果和返回值返回客户端。

OceanBase 数据库多租户线程处理本地 PX SQL 请求的过程如下图所示。



29.2 相关参数

29.2.4 cpu_quota_concurrency

Unit 的参数 `min_cpu` 和系统配置项 `cpu_quota_concurrency` 共同决定单个租户活跃线程数。

单个租户活跃线程数 = `min_cpu` * `cpu_quota_concurrency`

默认值为 4，取值范围为 [1,20]。该参数动态生效。

在标准生产部署下，建议保持默认值。

29.2.5 workers_per_cpu_quota

Unit 的参数 `max_cpu` 和系统配置项 `workers_per_cpu_quota` 共同决定单个租户可分配的线程数上限。

单个租户线程数上限 = `max_cpu` * `workers_per_cpu_quota`

默认值为 10，取值范围为 [2, 20]。该参数动态生效。

在标准生产部署下，建议保持默认值。

29.3 相关视图

视图名或表名	描述
GV$OB_TENANT_RUNTIME_INFO	OceanBase 数据库多租户线程中队列和线程限额相关的信息。

30 查看线程状态

在数据库的运行过程中，您可以根据需要查看当前集群中的线程状态，以便进行问题排查或性能诊断。

30.1 背景信息

线程可以分为工作线程和后台线程，工作线程是区分租户的，又称为租户工作线程；后台线程中既有分租户的线程又有不分租户的线程。

30.2 操作步骤

1. 管理员用户登录集群的 MySQL 租户或 Oracle 租户。
2. 执行以下命令，查看线程状态。
   - MySQL 模式

     ```
     SELECT * FROM oceanbase.GV$OB_THREAD;
     ```

     查询的部分结果的示例如下。

     ```
     +----------------+----------+-----------+------+----------------+--------
     +------------+------------+----------------------------------+
     | SVR_IP | SVR_PORT | TENANT_ID | TID | TNAME | STATUS | LATCH_WAIT |
     LATCH_HOLD | TRACE_ID |
     +----------------+----------+-----------+------+----------------+--------
     +------------+------------+----------------------------------+
     | xx.xx.xx.xx | 2882 | 1002 | 7268 | T1002_MFLaunch | Wait | | | Y0-
     0000000000000000-0-0 |
     | xx.xx.xx.xx | 2882 | 1002 | 7267 | T1002_MergeSche | Wait | | | Y0-
     0000000000000000-0-0 |
     | xx.xx.xx.xx | 2882 | 1002 | 7266 | T1002_FrzInfoDe | Wait | | | YB42AC1E87F4-
     0005FBC6A0F81772-0-0 |
     | xx.xx.xx.xx | 2882 | 1002 | 7265 | T1002_ServerPro | Wait | | | Y0-
     0000000000000000-0-0 |
     | xx.xx.xx.xx | 2882 | 1002 | 7264 | T1002_Occam | Wait | | | Y0-0000000000000000-
     ```

```
0-0 |
| xx.xx.xx.xx | 2882 | 1002 | 7263 | T1002_L0_G2 | Wait | | | Y0-0000000000000000-0-
0 |
| xx.xx.xx.xx | 2882 | 1002 | 7262 | T1002_L0_G2 | Wait | | | Y0-0000000000000000-0-
0 |
| xx.xx.xx.xx | 2882 | 1002 | 7102 | T1002_L0_G0 | Wait | | | Y0-0000000000000000-0-
0 |
| xx.xx.xx.xx | 2882 | 1002 | 7101 | T1002_L0_G0 | Wait | | | Y0-0000000000000000-0-
0 |
| xx.xx.xx.xx | 2882 | 1002 | 7100 | T1002_L0_G0 | Wait | | | Y0-0000000000000000-0-
0 |
+---------------+----------+-----------+------+----------------+--------
+------------+------------+----------------------------------+
10 rows in set
```

- Oracle 模式

```
SELECT * FROM SYS.GV$OB_THREAD;
```

查询的部分结果的示例如下。

```
+---------------+----------+-----------+------+-------------+--------+-----------
+------------+------------------------+
| SVR_IP | SVR_PORT | TENANT_ID | TID | TNAME | STATUS | LATCH_WAIT |
LATCH_HOLD | TRACE_ID |
+---------------+----------+-----------+------+-------------+--------+-----------
+------------+------------------------+
| xx.xx.xx.xx | 2882 | 1004 | 7372 | T1004_CdcSrv | Sleep | NULL | NULL | Y0-
0000000000000000-0-0 |
+---------------+----------+-----------+------+-------------+--------+-----------
+------------+------------------------+
```

```
1 row in set
```

查询结果中的每一行表示一个线程，不同时间查询的结果可能不同。每当有新的线程被创建时，系统就会在表中添加一行新的线程信息；同样的，当某个线程结束时，系统就会从表中删除该线程所在的行。

查询结果中的部分字段说明如下。

字段	描述
SVR_IP	当前线程所在的 OBServer 节点的 IP 地址
SVR_PORT	当前线程所在的 OBServer 节点的端口号
TENANT_ID	当前线程所属的租户，进程级线程的 tenant_id 为 500。
TID	线程号
TNAME	线程名
STATUS	线程状态： ● Run：表示线程处于运行状态 ● Wait：表示线程处于等待状态 ● Sleep：表示线程处于休眠状态 ● Join：表示线程在等待另一个线程结束
LATCH_WAIT	当前线程正在等待的 Latch 的地址
LATCH_HOLD	当前线程持有的 Latch 的地址，可能包含多个
TRACE_ID	当前线程上正在执行的 Trace 的 id

有关视图 GV$OB_THREAD 中字段的详细说明，参见 GV$OB_THREAD。

30.3 更多信息

关于 OceanBase 数据库线程的更多详情，请参见 线程简介。

31 OceanBase 数据库后台线程

本文列举了 OceanBase 数据库的一部分后台线程，并简单解释了这些后台线程的基本功能。在大部分场景下，用户完全不需要去关注后台线程的实现细节。

31.1 后台线程

在 OceanBase 数据库的版本迭代中，有可能会对后台线程进行持续优化和效率提高，因此有些后台线程会在版本升级的过程出现消失、合并的情况；也有可能随着版本的更新，出现新的后台线程。

线程名	归属模块	线程数量	功能
TsMgr	Trx	1	用于本地 GTS Cache 的刷新。
WeakReadSvr	Trx	1	用于计算本地 Server 级别备机读时间戳。
HAGtsSource	Trx	1	用于 ha_gts 的刷新，刷新频率由配置项 `gts_refresh_interval` 控制。
GCPartAdpt	Trx	1	定期检查 `all_tenant_gc_partition_info` 表，获取分区的 GC 情况，用于 GC 和事务 2PC 的推进。

PartWorker	Trx	1	遍历本机所有的 Partition，执行以下操作： 1. 每隔 50ms 尝试写事务层 Checkpoint Log。 2. 每隔 15s 计算迁移的快照点。 3. 每隔 10s 驱动本 Partition 上所有活跃事务，为其检查 Scheduler 的状态，用于事务上下文的 GC。 4. 每隔 10s 检查该 Partition 复制表的状态。 5. 每隔 10s 驱动刷新本 Partition 上缓存的 Clog日志。
LockWaitMgr	Trx	1	用于热点行场景下，等锁队列在特殊情况下的唤醒保障。例如，语句超时、Session 被Kill 等。
ClogAdapter	Trx	8	用于事务层异步提交 clog。
ObTransService	Trx	6	主要负责以下功能： ● 用于事务提交过程处理 Error Message。 ● 用于语句回滚失败之后的重试。 ● 用于提前解行锁场景，后继事务的唤醒操作。
GCCollector	storage	1	用于定期检查本机所有 Partition 是否能够 GC，如果可以则触发 GC 动作。
RebuildSche	storage	1	定期驱动需要 Rebuild 的 Partition 任务。

PurgeWorker	storage	1	后台 MemTable 的标记删除操作。
DAG	storage	16	Dag 的工作线程，用于 Partition 的转储、合并、迁移等任务的执行。
DagScheduler	storage	1	Dag 的调度线程。
STableChecksumUp	storage	1	用来批量提交汇报 SSTable Checksum 的任务。
PartitionScheduler	storage	2	用来调度转储和调度合并任务。
MemstoreGC	storage	1	用来回收转储预热完成之后的 Frozen MemStore。
ObStoreFile	storage	1	用来检查坏块和宏块回收。
PartSerCb	storage	40	用于分区 Leader 上任、卸任任务处理（20 个线程）。
ObPartitionSplitWorker	storage	1	用于处理分区分裂的后台任务。
FreezeTimer	storage	1	检查 MemStore 内存，触发冻结。
RebuildTask	storage	1	调度 D 副本拉取数据。
IndexSche	storage	1	调度建索引任务。
FreInfoReload	storage	1	同步内部表中 Major Freeze 相关信息。
TableMgrGC	storage	1	回收 MemTable，释放内存。
BackupInfoUpdate	storage	1	用于定时刷 Backup 的相关数据。
LogDiskMon	storage	1	用于检测系统多盘的情况。

KVCacheWash	storage	1	用于 KV Cache 的内存清洗。通过配置项 _cache_wash_interval 来控制检测周期，默认是 200ms，取值范围为 [1ms,1m]。
KVCacheRep	storage	1	用于整理 Cache 的内存碎片回收内存，通过配置项 _cache_wash_interval 来控制检测周期，默认是200ms，取值范围为 [1ms,1m]。
RSMonitor	RS	1	用来监控当前 RS 的状态。
CacheCalculator	RS	1	用来给 location_cache 和 schema_cache 预留内存。
ObDailyMergeScheduler	RS	1	用来调度每日合并。
ObEmptyServerChecker	RS	1	用来检测 OBServer 节点上是否不存在副本。
ObFetchPrimaryDDLOperator	RS	1	备库逻辑同步主库系统租户 Schema。
ObFreezeInfoUpdater	RS	1	推动冻结进度和管理快照点回收点。
ObGlobalIndexBuilder	RS	1	调度全局索引的构建。
ObGlobalMaxDecidedTransVersionMgr	RS	1	统计全局最大事务版本号。
ObLeaderCoordinator	RS	1	管理集群中副本 Leader 分布。
ObLogArchiveScheduler	RS	1	负责日志归档。
ObMajorFreezeLauncher	RS	1	负责每日发起定时合并。
ObPartitionSpliter	RS	1	负责调度分区合并。
ObRebalanceTaskMgr	RS	1	负责调度负载均衡任务的执行。

ObRootBackup	RS	1	负责调度备份任务。
ObRootBalancer	RS	1	负责生成负载均衡任务。
ObRsGtsMonitor	RS	1	监控 GTS 实例的分布情况，并基于 GTS 副本的分布，生成迁移或补 GTS 副本任务。
ObRsGtsTaskMgr	RS	1	执行 GTS 副本任务的线程。
ObHeartbeatChecker	RS	1	负责检查和 OBServer 节点之间的心跳状态。
ObStandbyClusterSchemaProcessor	RS	1	备库副本同步主库的普通租户 Schema。
ObRestoreScheduler	RS	1	负责恢复流程的调度。
ObWorkQueue	RS	4	执行 RS 的异步任务，线程数通过配置项 rootservice_async_task_thread_count 来控制，取值范围为 [1,10]。
ObAsyncTaskQueue	RS	16	执行构建索引的任务。
ObSwitchTaskQueue	RS	6	物理备库中主备切换的任务队列，线程数通过配置项 switchover_process_thread_count 来控制，取值范围为 [1,1000]。
LocalityReload	RS	1	定期刷新 Locality 信息。
RSqlPool	RS	1	备库用来定期维护主库的 Root Server 列表。
ServerTracerTimer	RS	1	用来维护和其他 Server 状态的定时任务。
LogEngine	CLOG	1	统计监控 clog 盘空间使用情况，执行 clog 文件的回收操作。
CLGWR	CLOG	1	负责 clog item 写盘。

ClogHisRep	CLOG	3	分区上下线时，更新 clog History 内部表。
BatchSubmitCtx	CLOG	1	负责拆分一阶段优化的事务。
LogScanRunnable	CLOG	4	OBServer 节点启动时，负责扫描所有 clog 文件。
LogStateDri	CLOG	1	负责 clog 模块主备角色切换、检查副本级联状态。
ObElectionGCThread	election	1	负责 Election 对象内存的回收。
Blacklist	CLOG	1	负责探测与通信目的端 Server 之间的网络是否联通。
BRPC	CLOG	5	负责后台执行 batch_rpc 聚合包发包。
CLOGReqMinor	CLOG	1	根据 clog 磁盘空间的情况触发 Minor Freeze，加快 clog 的回收。
LineCache	CLOG	1	用于优化历史数据的同步，Liboblog 场景使用。
EXTLogWash	CLOG	1	频率：100ms。
CLogFileGC	CLOG	1	用于定期回收 clog 文件。
CKPTLogRep	CLOG	1	用于日志副本的 Checkpoint操作。
RebuildRetry	CLOG	1	用于重试重建分区的任务。
ReplayEngine	CLOG	物理核	用于备机 clog 的回放。
PxPoolTh	SQL	0	并行执行的线程池。
sql_mem_timer	SQL	1	用于自动内存管理。
OmtNodeBalancer	common	1	负责后台刷新多租户信息。
MultiTenant	common	1	负责刷多租户 CPU 配比，用于资源调度。

SignalHandle	common	1	信号处理线程。
LeaseUpdate	common	3	负责 RS 与各个 Server 直接的 lease 监控。
RsDDL	common	1	负责 RS 的 DDL。
MysqlIO	common	12	负责 MySQL 的链接处理线程。
all_meta_table	common	8	用于 PG 或者分区状态的汇报。
all_pg_partition_meta_table	common	8	用于 PG 内分区的状态的汇报。
TimerMonitor	common	1	监控内部定时线程动作，对执行时间异常的任务报警。
ConfigMgr	common	1	用于配置项的刷新。
OB_ALOG	common	1	用于系统异步日志的日志落盘。
ELE_ALOG	election	1	用于选举模块异步日志的日志落盘。

32 多租户线程常见问题

32.1 1.OceanBase 数据库租户工作线程的线程名字是什么?

OceanBase 数据库支持多租户，每个租户都有一个唯一的 ID，租户 ID 是创建租户的时候分配的。每个租户的工作线程都是独立的，ID 为 1001 的租户，它的工作线程的名字是 TNT_L0_1001，其中：

- TNT：表示租户线程。
- L0：表示处理嵌套层级为 0 请求的线程。
- 1001：租户 ID。

其它租户以此类推。

32.2 2.OceanBase 数据库是如何分配工作线程的数量的?

OceanBase 数据库在启动之后就会把需要的线程创建好，之后除非调整相关配置项或租户规格，否则线程数基本不会再变化。某个租户在特定 observer 上的工作线程数是由租户的 Unit 规格决定的。

32.3 3.OceanBase 数据库的多租户架构中是如何做到 CPU 资源的有效隔离的?

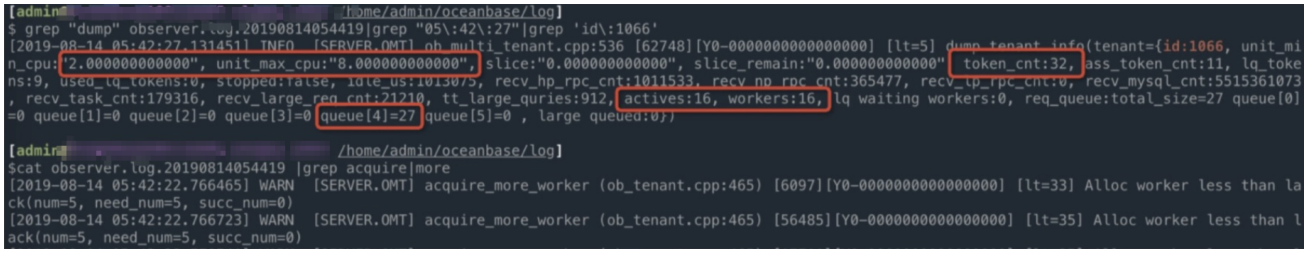
在 OceanBase 数据库目前发布的版本中，OceanBase 数据库主要依靠控制活跃线程数量（实际消耗 CPU 资源的线程）来控制 CPU 消耗。OceanBase 数据库在创建租户的时候会指定租户资源池，资源池定义 CPU 规格可以限制租户级别的 CPU 使用，从而实现租户间 CPU 使用的隔离。在 OceanBase 数据库的最新版本中，OceanBase 数据库内核实现了 cgroup 的隔离来对 CPU 进行有效地控制和限制，但这需要操作系统级别的配置搭配生效。

32.4 4.如何读取 OceanBase 数据库 Worker 线程的数量? OceanBase 数据库是否会在负载高的时候动态启动新的线程来执行负载?

在 observer.log 中，以关键字 dump tenant info 为主的日志片段里会描述现在 Worker 线程的现有值以及最大值。具体如下：

token_count = 分配给该租户的 cpu_count (>min_cpu&&<max_cpu) * cpu_quota_concurrency

OceanBase 数据库会在负载高的时候动态分配新的线程来执行负载。例如在一个特定的场景中，一台装有 OceanBase 数据库的机器上的租户 T1 配置了 unit_min_cpu = 2，unit_max_cpu=8，但是在该台机器上配置的 CPU 资源存在超卖。分配给 T1 的实际 CPU 为 5，那么 token_count = 5* cpu_quota_concurrency 。



32.5 5.OceanBase 数据库是如何对租户活跃线程和线程总数进行控制的？

OceanBase 数据库根据租户的 cpu_count * cpu_quota_concurrency 来计算活跃线程数，这些线程在租户创建的时候就会被创建好。例如，线程 A 是个活跃线程，它因为执行大查询被挂起，这时活跃线程就少了一个，作为补充，这个租户会额外再创建一个活跃线程。但是一个租户能创建的线程总数也是有限制的，总数限制为 cpu_count * workers_per_cpu_quota 。当租户线程数到达上限后，新创建线程会失败，线程 A 不会被挂起，以保持活跃线程数始终不变。例如，假设一个租户配置了 16 个 CPU，集群的 cpu_quota_concurrency 设置为 2，那么这个租户最多会有 16 * 2 = 32 个活跃的 SQL 线程，控制活跃线程数是 OceanBase 数据库的用户态线程调度器实现的。

32.6 6.大查询 CPU 资源分配原则是什么？OLAP 和 OLTP 同时存在的情况下会出现抢占 CPU 资源的情况吗？

在使用 OceanBase 数据库时，可以通过配置参数 large_query_threshold 来定义执行时间超过一定阈值的查询操作为大查询。如果系统中同时运行着大查询和小查询，OceanBase 数据库会将一部分 CPU 资源分配给大查询，并通过配置参数 large_query_worker_percentage （默认值为 30%）来限制执行大查询最多可以使用的租户活跃工作线程数。OceanBase 数据库通过限制大查询能使用的租户活跃工作线程数来约束大查询最多能够使用的 CPU 资源，以此来保证系统还会有足够的 CPU 资源来执行 OLTP（例

如，交易型的小事务）负载。通过这样的方式来保证对响应时间比较敏感的 OLTP 负载能够得到足够多的 CPU 资源尽快地被执行。另外需要注意的是，虽然 OceanBase 数据库可以做到大查询和 OLTP 资源的分配，large_query_threshold 参数也应设置在一个在合理的范围内，不应该设置成为一个过大的值。否则大查询很容易抢占系统的 CPU 资源而挤进而引发 OLTP 响应过慢甚至队列堆积的问题。

32.7 7.如何分析 OBServer 节点出现 CPU 偏高或者 OBServer 节点 Hang 住的情况?

当发现 OBServer 节点的 CPU 负载偏高已经达到了 80%- 90% 甚至以上，该 OBServer 节点已经发生了整体性能不佳甚至是 Hang 住的情况，那么可以通过 OBStack 工具（V2.2.3x 高配版本、V2.2.7x 以及 V3.x 版本均可独立安装）获取 OBStack 工具。可以执行以下命令直接执行 obstack 获取线程栈定向到问题 obstack.trc 中。

```
[root@hostname /]#obstack -p ${pid of observer} > obstack.trc
```

33 配置磁盘数据文件的动态扩容

OceanBase 数据库支持根据磁盘数据文件的实际使用情况来进行自动扩容。

33.1 背景信息

在 OceanBase 数据库中，存储数据的文件名为 block_file，block_file 文件位于 /OBServer 节点安装目录/store/sstable 目录下，该文件是在各 OBServer 节点启动后创建。且 block_file 文件的大小由配置项 datafile_size 或 datafile_disk_percentage 控制。更多 OceanBase 数据库数据文件目录的详细介绍请参见 OBServer 节点安装目录结构。

对于 OceanBase 数据库 V4.2.0 之前的版本，系统主要采用预分配的方式，将一部分磁盘空间预留给数据文件，以保证数据文件占有一段尽可能连续的磁盘空间，避免其他应用程序对磁盘的抢占引起的磁盘资源不足。但这种预分配的方式占用了很大的磁盘空间，即使磁盘空间没有被使用，也不能被释放。

为了解决预分配大块磁盘空间但不能被释放的问题，从 V4.2.0 版本开始，OceanBase 数据库引入了数据文件渐进式使用磁盘空间的用户配置选项，即系统先预分配一个合理的磁盘大小给数据文件，再根据磁盘的实际使用情况和用户的配置进行自动扩容。

33.2 相关配置项

OceanBase 数据库主要通过以下配置项配合使用来达到磁盘数据文件的自动扩容：

- datafile_size：用于设置数据文件占用磁盘可用空间的大小。默认值为 0M。有关集群级配置项 datafile_size 的详细说明，请参见 datafile_size。
- datafile_disk_percentage：磁盘数据文件占用磁盘总空间的百分比，默认值为 0。有关集群级配置项 datafile_disk_percentage 的详细说明，参见 datafile_disk_percentage。
- datafile_next：用于设置磁盘数据文件自动扩容的步长，默认值为 0，表示不启动增长，如果需要开启自动增长，则需要设置为非 0 的值。有关集群级配置项 datafile_next 的详细说明，请参见 datafile_next。
- datafile_maxsize：用于设置磁盘数据文件自动扩容的最大空间，默认值为 0，表示不启动自动扩容。有关集群级配置项 datafile_maxsize 的详细说明，参见 datafile_maxsize。

在配置上述配置项时，需要注意如下事项：

- datafile_size 与 datafile_disk_percentage 均可以用来设置数据文件占用的磁盘空间，您可以选择其中任意一项进行配置。其他情况下：
  - 如果两个配置项均已配置，即 datafile_size 与 datafile_disk_percentage 同时配置为非 0 的值，则以 datafile_size 设置的值为准。
  - 如果两个配置项均未配置，即 datafile_size 的值为 0M 且 datafile_disk_percentage 的值为 0，则系统会根据日志和数据是否共用同一磁盘来自动计算数据文件占用其所在磁盘总空间的百分比，即：
    - 如果日志和数据共用同一磁盘，则数据文件占用其所在磁盘总空间的百分比为 60%。
    - 如果日志和数据独占磁盘时，则数据文件占用其所在磁盘总空间的百分比为 90%。
- 配置 datafile_maxsize 时，其值需要大于当前数据文件占用的磁盘空间大小 datafile_size （或 datafile_disk_percentage ），如果设置的值小于当前数据文件占用的磁盘空间大小，则不会触发自动扩容。
- 如果 datafile_maxsize 的值超过了当前磁盘的最大可用空间，则以实际磁盘的可用大小作为最大值。
- 对于 datafile_next ：
  - 当 datafile_next 设置为小于或等于 1G 的值时，datafile_next 的取值为 min (1G, datafile_maxsize * 10%) 。
  - 当 datafile_next 设置为大于 1G 的值时，datafile_next 的取值为 min (datafile_next, 磁盘剩余空间) 。

33.3 注意事项及使用限制

- 配置磁盘数据文件的自动扩容功能时，建议步长 datafile_next 的初始配置设置为 datafile_maxsize 的 20% 左右，避免频繁扩容。
- 开启数据文件自动扩容后，需要做好同一机器上同时部署的其他应用程序的容量规划，避免造成实际可扩容的最大空间，小于配置所指定的 datafile_maxsize 的值的问题。
- 当前暂不支持磁盘数据文件的动态缩容。
- 如果不希望开启自动扩容功能，可以直接将 datafile_next 或 datafile_maxsize 的值设置为 0M 或者保持为默认值 0 ，系统将会按照原来的预分配的方式，将一部分磁盘空间预留给数据文件。

33.4 如何启动数据文件的自动扩展

33.4.1 步骤一：查看当前配置项的值

在配置磁盘数据文件的动态扩容前，需要先查看各配置项的值，以便了解各配置项的配置情况。

1. 使用 root 用户登录集群的 sys 租户。
2. 执行以下命令，查看各配置项的值。

```
obclient> SHOW PARAMETERS LIKE '%datafile_%';
```

33.4.2 步骤二：配置磁盘数据文件的动态扩容

获取各配置项的值后，您可以根据实际使用场景，通过修改配置项的值来实现磁盘数据文件的自动扩容。

33.4.2.1 集群启动时，已配置 datafile_size（或 datafile_disk_percentage）、datafile_next、datafile_maxsize

如果您在部署 OceanBase 数据库并启动 OceanBase 集群时，已将 datafile_size （或 datafile_disk_percentage ）、 datafile_next 及 datafile_maxsize 配置为非 0 的值，则：

- 如果 datafile_maxsize 的值大于 datafile_size 的值，或者 datafile_maxsize 的值大于根据 datafile_disk_percentage 比例换算的磁盘空间大小，则集群启动后，扩容就会自动生效。
- 如果 datafile_maxsize 的值小于 datafile_size 的值，或者 datafile_maxsize 的值小于根据 datafile_disk_percentage 比例换算的磁盘空间大小，则需要将 datafile_maxsize 的值修改为大于 datafile_size 的值后，才能开启数据文件的自动扩容。

33.4.2.2 集群启动时，仅配置了 datafile_size（或 datafile_disk_percentage）

如果您在部署 OceanBase 数据库并启动 OceanBase 集群时，仅设置了 datafile_size 或 datafile_disk_percentage 的值，您只需要手动设置 datafile_maxsize 和 datafile_next 的值，并且确保 datafile_maxsize 的值大于 datafile_size 的值，或者大于根据 datafile_disk_percentage 比例换算的磁盘空间大小即可。

33.4.2.3 集群启动时，未配置 datafile_size（或 datafile_disk_percentage）、datafile_next、datafile_maxsize

如果您在部署 OceanBase 数据库并启动 OceanBase 集群时，datafile_size（或 datafile_disk_percentage）、datafile_next、datafile_maxsize 等均未设置，您可以参考以下步骤，开启磁盘数据文件的自动扩容。

33.4.2.4 注意

集群首次启动时是否设置 datafile_size 或 datafile_disk_percentage，主要影响的是磁盘初始数据文件的大小。

具体操作如下：

1. 使用 root 用户登录集群的 sys 租户。
2. 执行以下语句，确认当前磁盘已经预分配的空间。

   ```
   obclient> SELECT data_disk_allocated/1024/1024/1024 AS datafile_G FROM oceanbase.GV$OB_SERVERS;
   ```

   查询结果的示例如下。

   ```
   +------------------+
   | datafile_G |
   +------------------+
   | 1770.000000000000 |
   +------------------+
   1 row in set
   ```

根据查询结果，当前磁盘数据文件预分配的空间大小 data_disk_allocated 为 1770 GB，约 1.7 TB。

GV$OB_SERVERS 视图的更多字段说明，请参见 GV$OB_SERVERS。

3. 根据上一步得到的结果，将 datafile_maxsize 的值设置为一个比 data_disk_allocated 大的值，同时设置 datafile_next 为一个非 0 的值。

   设置示例如下：

   ```
   ALTER SYSTEM SET datafile_maxsize='4TB';
   ```

   ```
   ALTER SYSTEM SET datafile_next='1TB';
   ```

4. 设置成功后，执行以下语句，再次确认自动扩容设置是否生效。

   ```
   SELECT data_disk_allocated/1024/1024/1024 AS datafile_G, data_disk_capacity
   /1024/1024/1024 AS datafile_max_G FROM oceanbase.GV$OB_SERVERS;
   ```

   查询结果的示例如下。

   ```
   +------------------+------------------+
   | datafile_G | datafile_max_G |
   +------------------+------------------+
   | 1770.000000000000 | 4096.000000000000 |
   +------------------+------------------+
   1 row in set
   ```

   根据查询结果，当前磁盘数据文件自动扩容的最大上限空间 data_disk_capacity 的值为设置的 datafile_maxsize 的值，且 data_disk_capacity 的值大于 data_disk_allocated 的值，说明自动扩容设置成功。

33.4.3 后续操作

配置成功后，后续系统会根据数据文件的实际使用情况，结合 datafile_next 和 datafile_maxsize 的值进行自动扩容，最大限度地降低对磁盘的分配。

33.5 相关文档

- OBServer 节点安装目录结构
- 部署 OceanBase 社区版
- 部署 OceanBase 企业版
- datafile_size
- datafile_disk_percentage
- datafile_next
- datafile_maxsize
- 修改集群参数

34 查看租户或表占用的磁盘空间

OceanBase 数据库支持通过视图来查看租户和表的磁盘空间占用情况。

34.1 查看租户所占用的磁盘空间

1. root 用户登录集群的 sys 租户。

   连接示例如下，连接数据库时请以实际环境为准。

   ```
   obclient -h10.xx.xx.xx -P2883 -uroot@sys#obdemo -p***** -A
   ```

2. 执行以下语句，查看租户在各节点上磁盘空间的占用情况。

   系统租户下，查看租户 mysql001 在各节点上不同组件分别占用的磁盘空间大小，语句如下：

   ```
   obclient [oceanbase]> SELECT * FROM oceanbase.CDB_OB_SERVER_SPACE_USAGE
   WHERE TENANT_NAME='mysql001' ;
   ```

   查询结果如下：

   ```
   +-----------+-------------+---------------+-------------+------------+------------
   +-------------+
   | TENANT_ID | TENANT_NAME | SERVER_IP | SERVER_PORT | SPACE_TYPE |
   DATA_BYTES | USAGE_BYTES |
   +-----------+-------------+---------------+-------------+------------+------------
   +-------------+
   | 1002 | mysql001 | xx.xx.xx.xx | 2882 | clog Data | 5898749661 | 5898749661 |
   | 1002 | mysql001 | xx.xx.xx.xx | 2882 | Index Data | 2793 | 24576 |
   | 1002 | mysql001 | xx.xx.xx.xx | 2882 | Meta Data | 28184576 | 28184576 |
   | 1002 | mysql001 | xx.xx.xx.xx | 2882 | slog Data | 26112000 | 26112000 |
   | 1002 | mysql001 | xx.xx.xx.xx | 2882 | Table Data | 25268 | 6438912 |
   | 1002 | mysql001 | xx.xx.xx.xx | 2882 | Tmp Data | 0 | 0 |
   | 1002 | mysql001 | xx.xx.xx.xx | 2882 | clog Data | 3348203904 | 3348203904 |
   | 1002 | mysql001 | xx.xx.xx.xx | 2882 | Index Data | 3936 | 32768 |
   ```

```
| 1002 | mysql001 | xx.xx.xx.xx | 2882 | Meta Data | 25452544 | 25452544 |
| 1002 | mysql001 | xx.xx.xx.xx | 2882 | slog Data | 39931904 | 39931904 |
| 1002 | mysql001 | xx.xx.xx.xx | 2882 | Table Data | 8056455 | 8736768 |
| 1002 | mysql001 | xx.xx.xx.xx | 2882 | Tmp Data | 0 | 0 |
+----------+------------+---------------+------------+-----------+------------
+-------------+
12 rows in set
```

返回结果中：

- SPACE_TYPE：数据的类型，包括以下几种类型：
  - Table Data：所有主表所有分区的数据，包含 Lob 分区
  - Index Data：所有索引表所有分区的数据
  - Meta Data：所有元数据
  - Tmp Data：临时文件
  - clog Data：当前节点上的 clog 总大小
  - slog Data：当前节点上的 slog 总大小
- DATA_BYTES：数据内容的大小，单位为字节。
- USAGE_BYTES：实际占用的磁盘空间大小，单位为字节。

34.2 查看表所占用的磁盘空间

1. sys 租户或用户租户的租户管理员登录到数据库。

   ## 说明

   MySQL 租户的管理员用户为 root 用户，Oracle 租户的管理员用户为 SYS 用户。

   连接示例如下，连接数据库时请以实际环境为准。

   ```
   obclient -h10.xx.xx.xx -P2883 -uroot@sys#obdemo -p***** -A
   ```

2. 执行以下语句，查看表数据的磁盘空间占用情况。
   - 系统租户

系统租户下，查询租户 mysql001 下表 t1 的磁盘空间占用情况，语句如下：

```
obclient [oceanbase]> SELECT * FROM oceanbase.CDB_OB_TABLE_SPACE_USAGE
WHERE TENANT_NAME='mysql001' AND TABLE_NAME='t1';
```

查询结果如下：

```
+-----------+----------+-------------+---------------+--------------------
+-------------+---------------+
| TENANT_ID | TABLE_ID | TENANT_NAME | DATABASE_NAME | TABLE_NAME |
OCCUPY_SIZE | REQUIRED_SIZE |
+-----------+----------+-------------+---------------+--------------------
+-------------+---------------+
| 1002 | 500051 | mysql001 | test | t1 | 1233 | 8192 |
+-----------+----------+-------------+---------------+--------------------
+-------------+---------------+
1 row in set
```

- 用户租户

  MySQL 模式
  Oracle 模式
  MySQL 租户查看本租户下各表的磁盘空间占用情况，语句如下：

  ```
  obclient [oceanbase]> SELECT * FROM oceanbase.
  DBA_OB_TABLE_SPACE_USAGE;
  ```

  Oracle 租户查看本租户下各表的磁盘空间占用情况，语句如下：

  ```
  obclient [SYS]> SELECT * FROM SYS.DBA_OB_TABLE_SPACE_USAGE;
  ```

  查询结果的示例如下：

  ```
  +----------+--------------+------------------------------+-------------
  +---------------+
  | TABLE_ID | DATABASE_NAME | TABLE_NAME | OCCUPY_SIZE | REQUIRED_SIZE |
  +----------+--------------+------------------------------+-------------
  ```

```
+---------------+
| 500011 | test | wide_table_row_storage2 | 4495925120 | 4613763072 |
| 500012 | test | wide_table_column_storage2 | 3915023145 | 4195397632 |
| 500014 | test | wide_table_row_column_storage2 | 8410964107 | 9805275136 |
+----------+--------------+-------------------------------+-------------
+---------------+
3 rows in set
```

返回结果中：

- OCCUPY_SIZE：表压缩后落盘的数据量，单位为字节。
- REQUIRED_SIZE：表压缩后落盘的数据量实际占用了多少磁盘空间，单位为字节。

从返回结果中，还可以通过 OCCUPY_SIZE / REQUIRED_SIZE 来计算磁盘空间的利用率。