常用命令

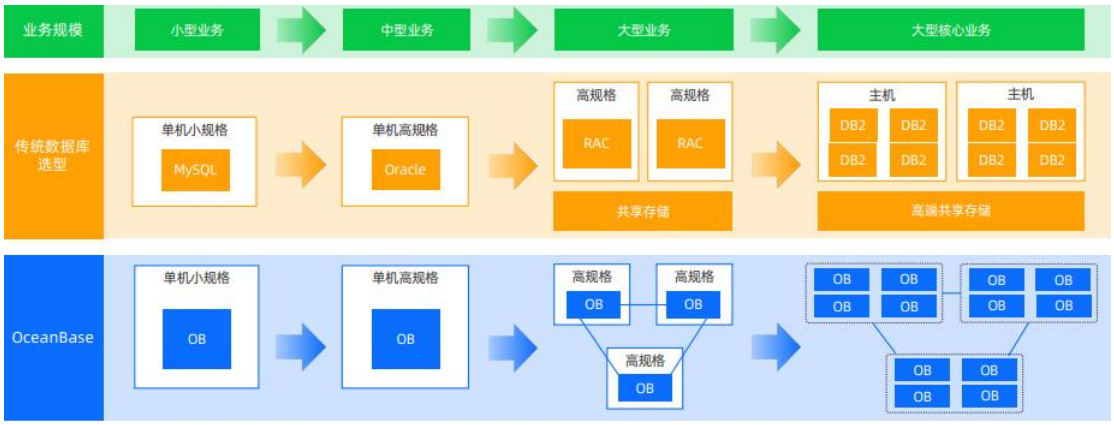
obcp 常用命令

数据库架构与原理

集群架构概述

OBV4 架构（单机分布式一体化）

单机分布式一体化的架构，既可以单机部署，也可以分布式部署；
可以从单机扩展为分布式（反之不支持）；



单机数据库优势：

单机数据库中，数据与计算都集中在同一个服务器中，不需要跨节点的数据访问和事务提交，可以获得较好的性能。但是，主备库部署的单机数据 库如果开启强同步模式，日志同步会对性能产生较大的影响。

分布式系统开销：

OceanBase 等分布式数据库通过将数据分片分布到多个服务节点来提高并发能力，采用 Paxos 等共识协议在多个副本中强同步日志流来保证高可 用能力。多副本间的强同步带来性能开销，开销的多少与日志流数量线性相关。

单机分布式一体化：

在 OceanBase V4 中，分区是数据分片的单位，但不再是日志流的单位，而是采用类似单机数据库的日志流的方式，将节点上承担 Leader 角色的 所有分区汇聚为一个日志流，实现了单机分布式一体化的架构。单机分布式一体化的架构中，参与 Paxos 同步的日志流数量指数级下降，大幅降低了分布式系统的开销，而且让 OceanBase 进行单机部署时，可以达到与单机数据库相当的性能。

一体化架构收益：

更低的资源消耗：日志流数量的指数级下降让 OceanBase 对基础资源的要求大幅降低，

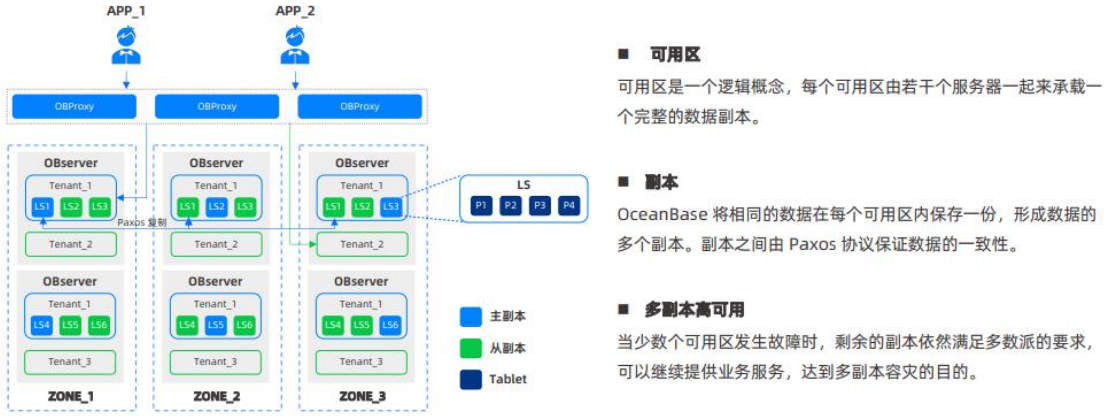
单机部署 OceanBase 的最小资源配置要求降低到 2C6G。同时，日志量同比 V3 版本减少了 30%~40%，大幅降低了分布式部署架构下日志同步的网络带宽要求；

更好的数据库性能:单机日志流带来单机性能的大幅提升，单机部署时能够获得比主流的开源单机数据库更好的性能。日志流数量的下降让单机支持的分区数量从 V3 版本的数万增长到百万；

灵活的部署形态：OceanBase V4 既支持单机部署，也可以分布式部署。因为使用了单机日志流，单机部署时 OceanBase 就是一个单机数据库。即使是分布式部署 的集群中，如果租户仅使用了单机的资源，则租户以单机模式运行，避免分布式的开销。而 OceanBase 原生的负载均衡机制可以让 OceanBase 平滑地从单机形态扩展为分布式形态，整个过程对应用透明无感。

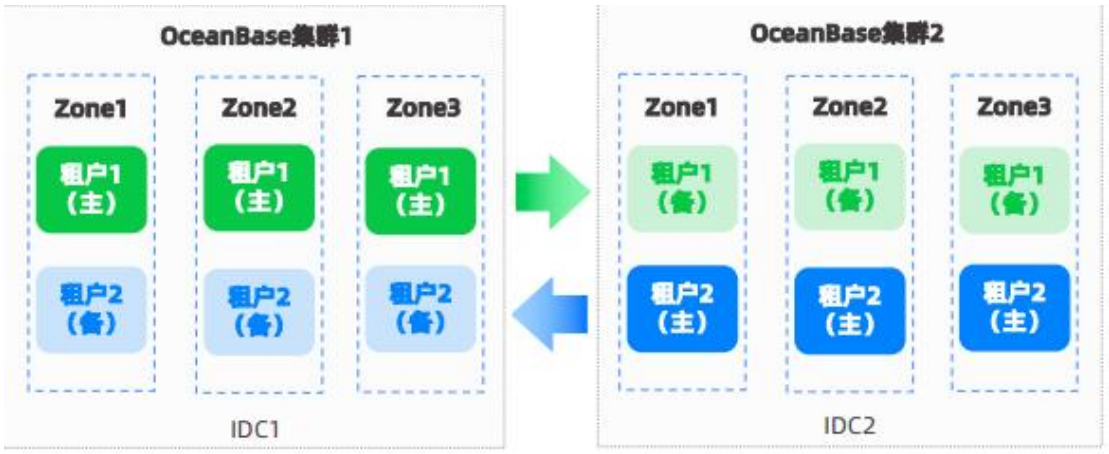
多副本高可用

OceanBase 数据库采用无共享（Shared-Nothing）分布式架构，各个节点之间完全对等，每个节点都有自己的 SQL 引擎、存储引擎、事务引擎， 运行在普通 PC 服务器组成的集群之上，具备高可扩展性、高可用性、高性能、低成本、与主流数据库高兼容等核心特性。



物理备库容灾

物理备库是 OceanBase 数据库的准实时热备份。从 V4.1 版本开始，OceanBase 按照租户粒度提供物理备库能力，即主或备的角色属于租户级别， 而不是集群级别。在主数据库无法提供服务时，物理备库可以被切换成主数据库对外提供服务，缩短服务不可用时间并减少可能的数据损失。



租户级物理备库

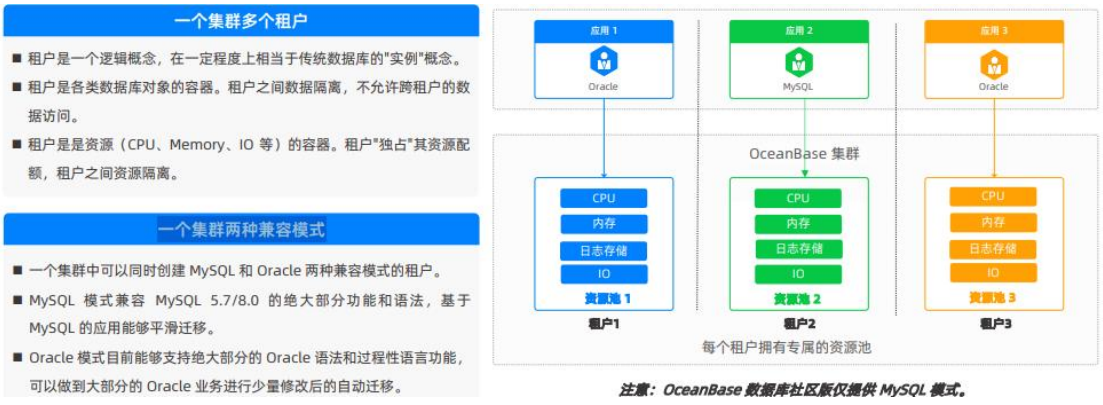
一个 OceanBase 集群中可以同时包含主租户和备 租户。集群名字不必相同，主备租户的名称也不要 求相同，租户的资源配置、副本数等配置也可不同。

主租户为业务提供读写服务并产生 Redo 日志，备租户通过同步主租户的 Redo 日志来同步主租户的数据。Redo 日志同步采用异步同步的模式。

多租户架构

OceanBase 数据库采用了单集群多租户（tenant）设计，通过租户实现资源隔离，让每个数据库服务的实例不感知其他实例的存在，并通过权 限控制确保租户数据的安全性，天然支持云数据库架构，支持公有云、私有云、混合云等多种部署形式。

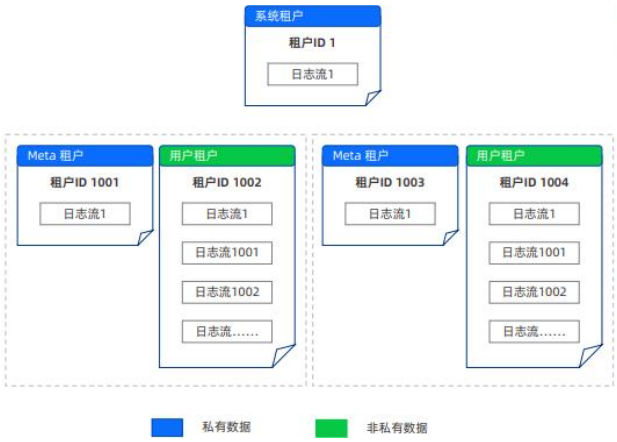
支持多租户模式，oracle 租户和 mysql 租户混部，租户间数据隔离，不允许跨租户访问。



☐ 【问题】上图中的集群架构是什么？primary_zone 设置？tenant_1 租户有几个日志流？

租户类型

系统租户、用户租户、meta 租户（用户租户 id 减 1）。



■ **系统租户**

系统租户是集群默认创建的租户，与集群生命期一致，负责管理集群和所有租户的生命周期。

■ **用户租户**

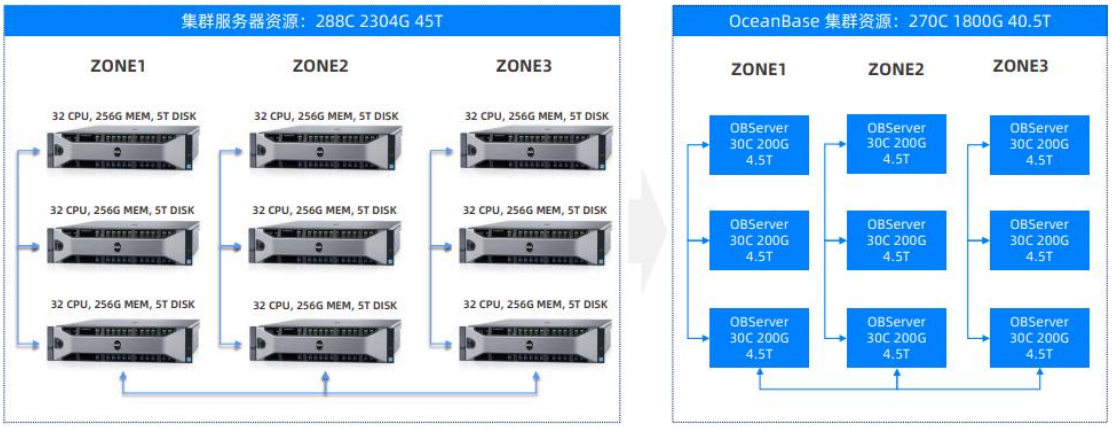
用户租户是由用户创建的租户，对外提供完整的数据库功能，支持 MySQL 和 Oracle 两种兼容模式。

■ **Meta 租户**

每创建一个用户租户，系统为该用户租户创建一个隐藏的 Meta 租户，用于存储和管理用户租户的集群私有数据。Meta 租户的生命周期与用户租户保持一致。

集群资源分配

0b 集群资源聚合



集群物理资源的逻辑表示

资源初始化

Cpu

CPU_COUNT 默认 0，代表 observer 可用的 cpu 总数，0 代表自动检测 cpu 数量；

CPU_CAPACITY_MAX 代表 observer 超卖后的 CPU 上限。

OceanBase V4 版本中，OBServer 的 CPU 资源允许 cpu 超卖，内存资源与磁盘资源不允许超卖

概念：OBServer 上所有租户的 CPU 资源之和大于 OBServer 的 CPU 资源数

控制：超卖的比例由集群配置项 resource_hard_limit 来控制

默认值为 100，表示 OBServer 最大可分配的 CPU 资源为 cpu_count*100%，即不允许超卖

当设置为大于 100 的值时，表示 OBServer 最大可分配的 CPU 资源为 cpu_count%

* resource_hard_limit，即允许超卖

MIN_CPU 之和不能超过 cpu_count 值

MAX_CPU 之和不能超过 cpu_count% * resource_hard_limit

Memory

MEMORY_LIMIT observer 的总内存大小，默认值为 0，单位为 MB；

MEM_CAPACITY observer 可供租户使用的内存上限；

memory_limit_percentage，默认值 80%，设置 OBServer 总内存占服务器内存大小的百分比,同时设置了 MEMORY_LIMIT 和 memory_limit_percentage 优先看 MEMORY_LIMIT，MEMORY_LIMIT 为默认时，取 memory_limit_percentage。

磁盘

数据磁盘

datafile_size 默认 0 OBServer 初始分配的数据文件的大小，默认单位为 MB

datafile_disk_percentage 默认 0 OBServer 初始分配的数据文件占其所在磁盘空间的百分比

日志磁盘

log_disk_size 默认 0 OBServer 的 Redo 日志总空间大小，默认单位为 MB

log_disk_percentage 默认 0 OBServer 的 Redo 日志占其所在磁盘空间的百分比

datafile_size 的优先级高于 datafile_disk_percentage，当 datafile_size 不为 0 时，系统会忽略 datafile_disk_percentage 的设置而优先使用 datafile_size 配置的值。

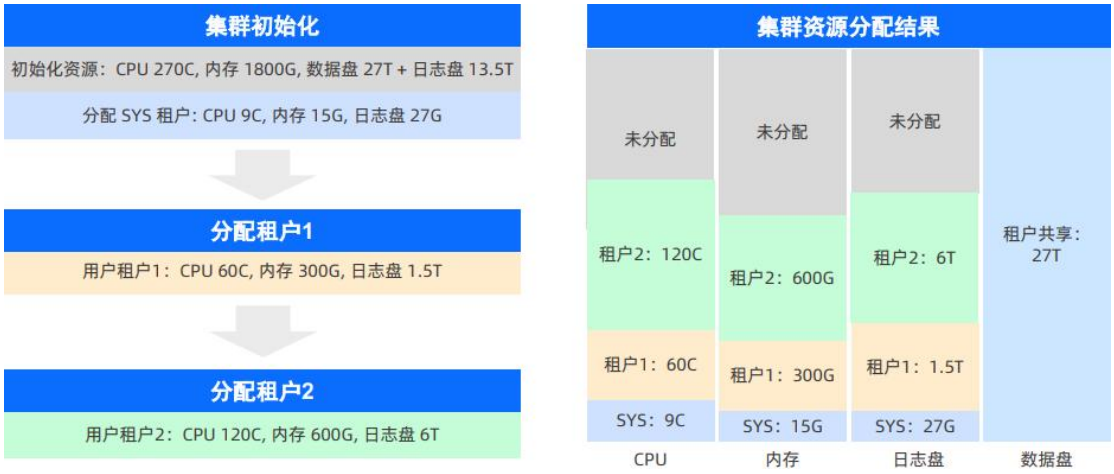
log_disk_size 的优先级高于 log_disk_percentage，当 log_disk_size 不为 0 时，系统会忽略 log_disk_percentage 的设置而优先使用 log_disk_size 配置的值。

以上配置项均为 0 时，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、Redo 日志占用其所在磁盘总空间的百分比

共用时，数据文件占用其所在磁盘总空间的 60%，Redo 日志占其所在磁盘总空间的 30%

独占时，数据文件与 Redo 日志分别占用其所在磁盘总空间的 90%

集群资源分配



租户资源分配（一般通过白屏）

一般通过白屏页面（odc）操作更简单

OceanBase 数据库是多租户的数据库系统，一个集群内可包含多个相互独立的租户，每个租户提供独立的数据库服务。在 OceanBase 数据库中，使用资源配置（Unit Config）、资源池（Resource Pool）和资源单元（Unit）三个概念，对各租户的可用资源进行管理。

■ 租户资源分配（租户创建）的通用流程

创建资源配置（一般通过白屏）

一般通过白屏页面操作

资源配置是描述资源池的配置信息，用来描述资源池中每个资源单元可用的 CPU、内存、日志存储空间和 IOPS 等的规格。

语法：

```
CREATE RESOURCE UNIT [IF NOT EXISTS] unit_name
MEMORY_SIZE [=] 'size_value',
[MIN_CPU [=] cpu_num,]
 MAX_CPU [=] cpu_num,
[MIN_IOPS [=] iops_num,]
[MAX_IOPS [=] iops_num,]
[IOPS_WEIGHT [=]iopsweight,]
[LOG_DISK_SIZE [=] 'size_value'];
```

示例：

```
CREATE RESOURCE UNIT unit1
MAX_CPU 20,
MIN_CPU 10,
MEMORY_SIZE '50G',
MAX_IOPS 128000,
MIN_IOPS 128000,
LOG_DISK_SIZE '2T' ;
```

参数说明：

必选参数

- MAX_CPU：租户的 CPU 规格上限（CPU核数）。
- MEMORY_SIZE：租户最多可以使用的内存大小。

可选参数

- MIN_CPU：租户的CPU规格下限。
  - 默认等于 MAX_CPU。
- LOG_DISK_SIZE：租户的日志磁盘大小。
  - 默认等于 3 倍的内存大小，最小为 2G。
- MAX_IOPS/MIN_IOPS：租户的 IOPS 上/下限。
  - 默认值为 INT64_MAX，即没有限制。
- IOPS_WEIGHT：租户 IOPS 的权重。
  - 默认按照 CPU 规格自动计算。

***注意：**创建资源配置时，还未真正分配资源单元，OceanBase 不会校验 OBServer 的可用资源是否满足资源配置要求。*

创建资源池（一般通过白屏）

资源池由若干个资源单元组成，通过给资源池指定资源配置，可指定资源池下各资源单元的物理资源。

语法：

```
CREATE RESOURCE POOL pool_name
UNIT [=] unit_name,
UNIT_NUM [=] unit_num,
ZONE_LIST [=] ('zone_name' [, 'zone_name' ...]);
```

示例：

```
CREATE RESOURCE POOL pool1
UNIT='unit1',
UNIT_NUM=1,
ZONE_LIST=('zone1');
```

参数说明：

- UNIT：资源池使用的资源配置的名字。
- UNIT_NUM：资源池在所在的每一个 Zone 内的资源单元个数。
  - 对于任一资源池，一台 OBServer 至多提供一个资源单元。
  - 只有当 Zone 内的 OBServer 个数大于1时，才可以指定 UNIT_NUM 大于1。
- ZONE_LIST：使用该资源池规格进行资源分配的 Zone。
  - 可以为多个 Zone 创建同一个资源池，也可以为不同的 Zone 分别创建单独的资源池。

***注意：**创建资源池时，OceanBase 会根据资源配置分配资源单元，如果 OBServer 的可用资源不足，则创建资源池失败。*

创建租户（一般通过白屏）

创建资源池后，可以在创建租户时将资源池指定给租户。一个资源池仅能属于一个租户，一个租户在同一个 Zone 上仅能有一个资源池。一个租户的所有资源池下的全部资源单元的集合描述了该租户可以使用的全部物理机资源。

语法：

```
CREATE TENANT ( {[IF NOT EXISTS] tenant_name
[tenant_option] [set_sys_var]} );

tenant_option:
    | LOCALITY = locality_description
    | PRIMARY_ZONE= primary_zone_list
    | RESOURCE_POOL_LIST [=] (pool_name [, pool_name...])
    | {CHARACTER SET | CHARSET} [=] charset_name
    | COLLATE [=] collation_name


set_sys_var:
SET var_name = var_value [,var_name =var_value...]
```

参数说明：

tenant_option

- LOCALITY：指定租户的副本都分布在哪些 Zone。
  - 默认为分布在集群的所有zone。
- PRIMARY_ZONE：指定租户提供读写服务的 Zone 列表。
  - 默认为 RANDOM，即所有的 Zone 都是 Primary Zone。
- RESOURCE_POOL_LIST：租户可以使用的资源池列表。

set_sys_var：创建租户的同时设置租户级的全局变量。

- ob_compatibility_mode：租户的兼容模式，默认为 MySQL。
  - 租户模式仅能在租户创建时指定，不允许在创建租户后修改。
- ob_tcp_invited_nodes：租户连接的白名单，即允许哪些客户端 IP 连接该租户。
  - 默认仅允许本机的 IP 登录该租户。

租户资源分配的要点

资源单元（Unit）是资源分配的最小单元
资源配置（Unit Config）定义了资源单元的大小；
同一个资源单元不能跨 OBServer 节点分配；
在一台 OBServer 上每个租户至多有一个资源单元。
租户通过资源池（Pool）分配资源
资源池定义了资源单元的大小和数量；
一个资源资源池只能属于一个租户；
租户在任一 Zone 内只能有一个资源池，租户在所有 Zone 的资源池不能出现交叠;
一个租户的多个资源池必须具有相同的资源单元个数（UNIT_NUM）。
租户资源单元的分配会自动均衡
资源单元是集群负载均衡的一个基本单位
OceanBase 在一个 Zone 内的多个 OBServer 上分配资源单元时，让每个 OBServer 的资源分配尽量均等
如果每个 Zone 内 OBServer 数量与资源配置相同，自动均衡后每个 Zone 内资源单元的分布相同

SYS 租户资源分配

SYS 租户资源分配

SYS 租户由 OceanBase 自动创建，OceanBase 根据 OBServer 的资源大小自适应地定义 SYS 租户的资源配置，用户也可以修改 SYS 租户的资源配置，集群中的 SYS 租户，租户模式为 mysql 模式。

SYS 租户下资源分配相关的视图

DBA_OB_UNIT_CONFIGS，展示资源配置的定义
DBA_OB_RESOURCE_POOLS，展示资源池的定义以及关联租户
DBA_OB_UNITS，展示资源单元的配置、分布
GV$OB_UNITS，展示资源单元的配置、分布和资源使用的信息
DBA_OB_TENANTS，展示租户的定义
GV$OB_SERVERS，展示 OBServer 的基本信息，包括资源分配和使用的信息

租户间资源隔离

CPU（隔离）
硬隔离，基于 linux cgroup 机制，v4 版本的默认隔离机制，OceanBase 为后台线程和不同租户的工作线程分配不同的 cgroup 目录，并根据租户的资源配置

（MIN_CPU/MAX_CPU ）设置 cgroup 目录内的 CPU 使用限额

当系统的 CPU 负载不高时，按照 MAX_CPU 控制每一个租户的 CPU 使用限额

当系统的 CPU 负载较高时，按照 MIN_CPU 控制每一个租户的 CPU 使用权重

租户的 CPU 使用不会超过其资源配置的上限，一个租户繁忙不会影响到集群内的其他租户

软隔离，根据租户的资源配置分配不同的工作线程数量，通过工作线程数量来控制租户的 CPU 使用

工作线程是处理 SQL 和事务提交的线程。由于 SQL 执行过程中可能会有 IO 等待、锁等待等，一个工作线程无法用满一个物理 CPU，OBServer 节点会给每个 CPU 配额启动多个工作线程

租户配置项 cpu_quota_concurrency 控制每个 CPU 配额所允许的活跃工作线程数量，默认值为 4。如果 Unit 的 MIN_CPU 配额是 10，该 Unit 内的最大活跃工作线程数是 40，是其 CPU 配额的 4 倍

因为租户的工作线程数远大于租户的 CPU 核数，所以租户可以使用超过其 CPU 限额的资源，一个租户 CPU 使用过高会影响到其他租户

CGROUP 隔离（OCP 部署 v4 时会自动配置 cgroup 隔离，命令行安装需要手工配置）

分组机制，cgroup （control groups）是 Linux 内核提供的一种机制，这种机制可以根据特定的行为，将一系列系统任务及其子任务整合（或分隔）到按资源划分等级的不同组内，从而为系统资源管理提供一个统一的框架。OceanBase 通过为每个租户创建一个子目录、租户目录内再按照线程分组建立二级子目录的方式，实现对不同租户、租户内不同线程的 CPU 资源的使用控制。

■ 按租户分组控制

```
[root@ob1 /]# ll /sys/fs/cgroup/cpu/oceanbase
        ... 省略部分文件 ...
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.cfs_burst_us
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.cfs_period_us
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.cfs_quota_us
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.qos_level
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.rt_period_us
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.rt_runtime_us
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.shares
-r--r--r--  1 admin admin 0 Jun 20 14:39 cpu.stat
drwxr-xr-x  2 admin admin 0 Jul  2 16:13 other
-rw-r--r--  1 admin admin 0 Jun 20 14:39 tasks
drwxr-xr-x 12 admin admin 0 Jul  3 02:00 tenant_0001
drwxr-xr-x  3 admin admin 0 Jul  2 16:13 tenant_0508
drwxr-xr-x  3 admin admin 0 Jul  2 16:13 tenant_0509
drwxr-xr-x  9 admin admin 0 Jul  3 02:00 tenant_1002
drwxr-xr-x  9 admin admin 0 Jul  3 02:00 tenant_1004
drwxr-xr-x 10 admin admin 0 Jul  9 16:19 tenant_1006
```

■ 按线程分组控制

```
[root@ob1 /]# ll /sys/fs/cgroup/cpu/oceanbase/tenant_1002
        ... 省略部分文件 ...
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.cfs_burst_us
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.cfs_period_us
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.cfs_quota_us
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.qos_level
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.rt_period_us
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.rt_runtime_us
-rw-r--r--  1 admin admin 0 Jun 20 14:39 cpu.shares
-r--r--r--  1 admin admin 0 Jun 20 14:39 cpu.stat
drwxr-xr-x 2 admin admin 0 Jul  2 16:13 OBCG_DEFAULT
drwxr-xr-x 2 admin admin 0 Jul  2 16:13 OBCG_ELECTION
drwxr-xr-x 2 admin admin 0 Jul  3 02:00 OBCG_LOCK
drwxr-xr-x 2 admin admin 0 Jul  2 22:00 OBCG_LQ
drwxr-xr-x 2 admin admin 0 Jul  2 16:13 OBCG_MYSQL_LOGIN
drwxr-xr-x 2 admin admin 0 Jul  2 17:14 OBCG_WR
-rw-r--r-- 1 admin admin 0 Jul  2 16:13 tasks
drwxr-xr-x 7 admin admin 0 Jul  3 02:00 tenant_1001
```

CPU 分配机制

OceanBase 成功配置 cgroup 隔离后，会根据租户的资源配置在租户的 cgroup 目录下定义其 CPU 使用控制文件

cpu.cfs_period_us：CPU 时间周期的长度，默认值 100000 (单位: us)，即 100ms。cgroup 在该周期内控制各个子目录对 CPU 的使用

cpu.cfs_quota_us：CPU 时间的最大限额，控制目录下所有线程可以使用的总的 CPU 时间，与 MAX_CPU 相对应。

cpu.cfs_quota_us = max_cpu * cpu.cfs_period_us

cpu.shares：CPU 时间的最小限额，相对权重，确定在同级目录发生 CPU 争抢时本目录可以获得的 CPU 占比，与 MIN_CPU 对应。

cpu.shares = 租户 min_cpu * 1024

租户 ID	MAX_CPU	MIN_CPU	cpu.cfs_quota_us	cpu.shares
1	3	3	-1	3072
1002	5	5	500000	5120
1004	10	5	1000000	5120

内存（隔离）

照租户的资源配置来分配和管理租户内存，租户间完全隔离

日志盘（隔离）

按照租户的资源配置来控制租户的日志盘使用，默认为租户内存的 3 倍

每个租户的日志文件和刷盘线程都是独立的

IO（隔离）闲时共享，忙时隔离

按照租户的资源配置控制不同租户的 IO 使用，通过租户资源配置项中的三个 IOPS 规格参数：MIN_IOPS、MAX_IOPS、IOPS_WEIGHT，来控制不同租户的 IOPS 使用上限。

MIN_IOPS：租户的 IOPS 最低保证，默认为不做限制

MAX_IOPS：租户的 IOPS 最大值，默认为不做限制

IOPS_WEIGHT：OBServer 上发生 IO 争抢时，OceanBase 分配给租户的 IOPS 的权重

如果 MAX_IOPS 和 MIN_IOPS 任一项被指定，则 IOPS_WEIGHT 的默认值等于 0

如果 MAX_IOPS 和 MIN_IOPS 均未指定，则 IOPS_WEIGHT 的默认值等于 MIN_CPU

OceanBase 在安装过程中会执行 IO Bench 测试，计算每个 OBServer 的 IOPS 上限

OBServer 出现 IO 争抢时，OceanBase 通过不同租户的 IOPS 设置来为计算每个租户的 IOPS 上限

数据盘（v4 版本不支持隔离）

网络（v4 版本不支持隔离）

分片与副本

分区与分片

OceanBase 数据库参考传统数据库分区表的概念，把一张表格的数据划分成不同的分区（Partition），分区的数据存储在对应的分片（Tablet）上。OceanBase 将分片均衡地分布在多个 OBServer 上，实现数据的分布式存储。

分区（partition）

分区：用户创建的逻辑对象，是 ob 进行数据分片的方式

将同一个表的多个分区分布在多个 OBServer 节点上，组成一份完整的表数据，实现数据的分布式存储

未做分区的表以单分区表的形式存在

分片（tablet）

分片是分区数据的存储对象，是数据均衡的最小单位

分片与分区一一对应，单分区表会创建一个分片，多分区表会为每个分区创建一个分片

索引的每个分区也会对应一个分片。特别地，局部索引的分片与主表的分片绑定在一起，相同分区的分片存储在一台 OBServer 上

副本

为保证数据读写服务的高可用，OceanBase 分布式数据库会将同一份数在多个可用区（Zone）内同步存储多份。每个可用区内一份完整的数据被称为副本

副本类型

全功能副本，可以提供全部的数据服务

leader， 提供写服务和强一致读服务，也可以提供弱一致性读服务

follower，提供弱一致读服务，在 Leader 故障的情况下还可以快速切换为 Leader

只读副本，提供只读服务

特性项	全功能副本	只读副本
副本名称及缩写	FULL(F)	READONLY(R)
每个可用区内的最大副本数量	1	1
是否有 Clog（Redo Log）	有	有（异步日志）
是否有 MemTable	有	有
是否有 SSTable	有	有
是否参与 Paxos 投票	参与	不参与
可否当选为 Leader	可以	不可以
是否提供读写服务	Leader 提供读写，Follower 可弱一致性读	仅可弱一致性读（Listener）
副本类型转换	可转换为只读型副本	可转换为全能型副本

注意：强、弱一致性读均读取已完成事务提交的数据，但强一致性读是读取最新的数据版本，弱一致性读不要求读取最新的数据版本。

副本分布

Locality 是租户级的属性，仅能在 CREATE TENANT 或 ALTER TENANT 时指定

语法 replicas{量词}@location

replicas：副本类型，即 FULL 或 READONLY，也可用 F 或 R 代替

量词：副本在指定可用区内的数量，固定为 1

location：可用区的名字

租户默认的 Locality 是在集群的每一个可用区内各创建一个全功能副本

可以通过 DBA_OB_TENANTS 视图的 LOCALITY 字段查看租户的副本分布

流量分布（primary zone）

OceanBase 数据库使用 Primary Zone 描述 Leader 副本的偏好位置，通过为不同的租户配置不同的 Primary Zone，可以将不同租户的业务流量集中到不同的可用区中。

Primary Zone 是租户级的属性，仅能在 CREATE TENANT 或 ALTER TENANT 时指定

用分号（;）分割表示不同的优先级，用逗号（,）分隔表示相同的优先级，逗号两侧优先级相同，分号左侧优先级高于右侧

租户默认的 Primary Zone 设置为 RANDOM，即 zone1,zone2,zone3

可以通过 DBA_OB_TENANTS 视图的 PRIMARY_ZONE 字段查看租户的副本分布

位置信息（region&idc）

Region 和 IDC 描述 Zone 的位置信息，一个 Zone 有且仅有一个 Region 和 IDC 属性。OBServer 的位置信息继承自 Zone，同一个 Zone 内的 OBServer 服务器都具有相同的 Region/IDC 属性。

地域 region

Region 是指一个地域或城市，不同 Region 通常距离较远。在 OceanBase 数据库中，一份数据可以被部署在多个 Region 中的不同副本上，以保证数据的高可用性和容错性

一个 Zone 只能属于一个 Region，一个 Region 可能包含一个或多个 Zone

互联网数据中心 idc

IDC 是指一个数据中心，在 ob 中用数据中心的一个机房（Logical Data Center，LDC）代表 IDC

一个 Zone 只能属于一个 IDC，一个 IDC 可能包含一个或多个 Zone

多副本同步与日志流

OceanBase 数据库基于 Paxos 协议实现了多副本同步与容灾方案。Leader 副本提供事务写操作，将事务日志持久化并与其他副本同步，基于 Paxos 协议保证日志数据在多数派副本持久化成功，同时通过成员变更提供容灾能力。多副本日志同步的单位不是单个数据分片，而是多个分片聚合在一起的日志流。

数据分片：数据分片的单位是分区，多个副本通过日志流同步数据

日志流：在 V3 版本，日志流的单位为数据分片；从 V4 版本开始，日志流是一个租户在同一个 OBServer 上的所有 Leader 分区的聚合，其日志持久化与同步的方式与单机数据库相似，因而被称为单机日志流。

Paxos 组：同一日志流的多个副本使用 Paxos 一致性协议保证副本的强一致，每个日志流和它的副本构成一个独立的 Paxos 组。

副本同步：Paxos 组成员通过 Redo-Log（事务日志）同步数据，事务的提交需要在多数派成员中强同步 Redo-Log，从而实现分布式集群的多副本高可用。

分布式一致性协议

OceanBase 数据库使用优化的 Multi-Paxos 协议在 Paxos 组内执行 Leader 选举和日志同步。（只读副本不是 Paxos 组成员，不参与选举投票和 Redo-Log 的多数派强同步）

Paxos 组选举

Paxos 组成员选举 Leader 副本。如果一个副本获得超过半数的 Paxos 组成员的投票，则该副本当选为 Leader

当 Leader 副本出现故障时，OceanBase 会基于 Paxos 协议在剩下的副本中选出新的 Leader。

单机日志流的架构让故障改选可在 8 秒内完成（RPO< 8 秒）

Paxos 组同步

Paxos 组的 Leader 副本对外提供写服务并产生 Redo-Log。任何时候，一个日志流仅有一个 Leader 副本

Redo-Log 的提交需要在超过半数的 Paxos 组成员中同步完成

Paxos 组的 Follower 副本通过回放 Redo-Log 而同步数据

故障改选后，新的 Leader 会确保事务状态与数据的一致性

广播日志流（配置表场景）

OceanBase V4.2.0 版本开始引入了广播日志流的概念，广播日志流是一个特殊的日志流，用于租户内所有复制表的副本同步。广播日志流会在租户内的每个 OBServer 节点上均部署一个副本，保证在理想情况下复制表可以在任意一个 OBServer 节点上提供强一致性读。

复制表

复制表会在租户的每一个 UNIT 内各创建一个副本，以满足大量并发的读请求

复制表只有一个 Leader 副本，可以接受写请求；复制表的所有“健康”副本都能接受读请求

广播日志流

一个租户只有一个广播日志流，承载租户内所有的复制表

租户在 Zone 内有多个 UNIT 时，其中一个 UNIT 按照 Locality 描述分配该 zone 的副本类型，其他 Unit 上均分配只读型副本（Listener）

广播日志流在所有非只读副本间强同步

副本与日志流相关系统视图

在 SYS 租户中使用 CDB_OB 为前缀的视图代替 DBA_OB 视图，可以来查看所有租户的信息

DBA_OB_TABLET_REPLICAS，展示本租户的所有 TABLET 副本信息

DBA_OB_TABLE_LOCATIONS，展示本租户的表或者分区所在的位置

DBA_OB_TABLET_TO_LS，展示本租户的 Tablet 和 LS 的映射信息

DBA_OB_LS，展示本租户的日志流的定义和状态信息

DBA_OB_LS_LOCATIONS，展示本租户的日志流副本的位置信息

GV$OB_LOG_STAT ，展示租户的日志流副本的同步状态信息（SYS 租户可查询所有租户的信息）

负载均衡

自动负载均衡

负载均衡机制

OceanBase 数据库通过 RootService 提供动态的负载均衡能力，提供了水平扩缩容和数据动态均衡等负载均衡能力，可以达到各个服务节点上资源分配与使用的均衡，数据分片数量与日志流副本数量的均衡，以及流量分布（Leader 副本）的均衡。

RootService(RS)： 是 SYS 租户提供的数据库集群的总控服务，主要提供资源管理、容灾、负载均衡、schema 管理等功能。

资源单元(Unit)均衡：是 OBServer 间的均衡，其目的是在 Zone 内多个 OBServer 上合理分布资源单元，达到 Zone 内多个节点之间的资源分配的均衡。在当前的算法中主要考虑两类资源（ CPU 资源和 Memory 资源）的分配和均衡。

租户内均衡：包含日志流副本的均衡和分区数据均衡两个级别。

☐ 日志流均衡：当用户对租户执行 UNIT_NUM 变更、修改 PRIMARY_ZONE、Locality 等操作时，负载均衡会通过日志流的分裂、合并等变更动作，变更日志流的数量和位置，以满足日志流数量均衡和 Leader 分布的均衡。

☐ 分区均衡：指在表和分区的个数和数据量动态变化的情况下，通过动态调整分区在多个日志流中的分布，实现多个日志流之间分区个数以及存储空间的均衡。

资源单元的均衡

RootService 模块对资源单元进行管理，并通过把资源单元在多个节点间调度，对系统资源进行有效利用。

资源单元的分配:在创建一个新的 Unit 时，RS 首先计算 Zone 内各个 OBServer 的资源占用率，然后选取资源占用率最小的那个 OBServer 节点作为新建 Unit 的宿主机，从而达到各个 OBServer 资源分配尽量均等的目的

资源单元的均衡：在集群扩缩容场景，添加、删除 OBServer 后，RS 会通过在节点间迁移 Unit 的方式，使得 Zone 内各个节点的资源占用率尽量接近，即完成了资源单元的均衡

资源单元均衡的控制

enable_rebalance：租户级配置项，是负载均衡机制的总开关，默认值为 True

当 SYS 租户的 enable_rebalance 设置为 True 时，开启租户间资源单元的均衡

当用户租户的 enable_rebalance 设置为 True 时，开启租户内的日志流均衡和分区均衡

server_balance_cpu_mem_tolerance_percent：集群级配置项，控制触发资源单元均衡的阈值，默认值为 5

当 Zone 内某些节点的资源占用率与平均的资源占有率的差值超过该配置项的值时，触发资源单元均衡

日志流均衡

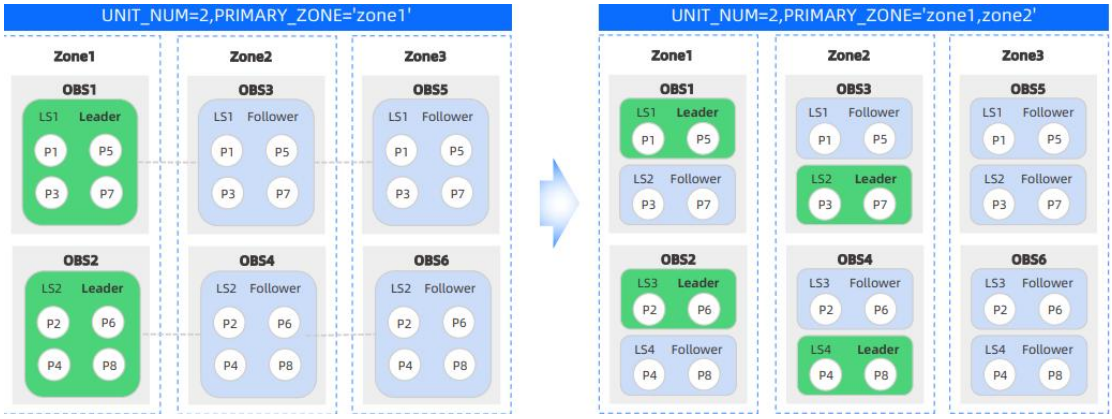
日志流数量均衡

当对租户执行扩缩容时，比如更改 UNIT_NUM、修改 PRIMARY_ZONE 的分布，负载均衡机制会对日志流就像相应的分裂、合并等操作，满足变更后的日志流的数量要求和 Leader 副本的均衡。

日志流数量均衡：日志流的数量 = 最高优先级的 Primary_Zone 数 * UNIT_NUM 数

leader 均衡

在满足日志流数量均衡的前提下，负载均衡机制会让多个日志流的 Leader 副本平均分布在 Primary Zone 内。



分区均衡

从 V4.2.1 版本开始，在日志流均衡的基础上，负载均衡模块还可以将分区 Tablet 在不同的日志流间迁移（Transfer），达到租户下的分区均衡

Transfer：将 Tablet 从一个日志流转移到另一个端日志流的操作

enable_transfer：租户级配置项，用于控制是否允许在租户内进行 Transfer，默认值为 true，即开启 Transfer 功能

分区数量均衡：分区数量均衡的目标是让租户的所有日志流上的用户表主表分区数量均匀（数量偏差不大于 1），租户内有多少个表分区，就有多少个 leader 节点，可以通过每台 observer 上 leader 节点的数量，来观察分区数量是否均衡。（CDB_OB_TABLE_LOCATIONS、DBA_OB_TABLE_LOCATIONS）

☐ 局部索引的分布跟随主表分区，其分区数量不计算在分区数量均衡之内

☐ 全局索引可以看做是一种特殊的主表，其分区数量要计算在分区数量均衡之内

磁盘负载均衡：在分区数量均衡的基础上，负载均衡模块通过尽可能地交换分区，实现各个日志流的磁盘占用均等的目的

server_balance_disk_tolerance_percent：集群级配置项，控制磁盘负载均衡的触发条件

当一个租户的所有日志流副本的磁盘使用量占比差值超过该阈值时，触发磁盘均衡

默认值为 1。为避免小数据量场景均衡过于频繁的问题，当前版本在单个日志流的磁盘占用小于 50GB 时不会触发磁盘均衡

基于均衡组的分区均衡

OceanBase 数据库采用 “均衡组” 的概念描述分区的打散关系，将需要打散分布的分区划分到同一个均衡组内，通过组内均衡和组间均衡的方式实现分区数量均衡。

在不使用表组（Table Group）的场景下，均衡组的划分与均衡策略

表类型	均衡组划分	均衡组内的均衡策略
非分区表	• 租户下所有非分区表是一个均衡组	• 均衡组内的所有非分区表平均分布到租户的所有 LS 上
一级分区表	• 每张表下所有一级分区是一个均衡组	• 均衡组内的所有一级分区平均分布到租户的所有 LS 上
二级分区表	• 单张表的每个一级分区下的所有二级分区是一个均衡组	• 均衡组内的所有二级分区平均分布到租户的所有 LS 上

基于均衡组的分区均衡分两步执行

组内均衡：先对所有均衡组进行组内均衡，即组内分区平均分布在不同的 LS（日志流）

组间均衡：从分区数最多的 LS 上 Transfer 走一部分分区，进行组间均衡，从而实现分区数量均衡

基于表组的分区均衡

表组（Table Group）是一个逻辑概念，表示一组表的集合。OceanBase 数据库依据表组的 SHARDING 属性，相应地划分表组内表和分区的均衡组，并基于均衡组进行组内均衡和组间均衡，达到不同表、分区的聚合与均衡的目的

基于表组（Table Group）的 SHARDING 属性进行组内均衡的策略

SHARDING	均衡组划分	均衡组内的均衡策略
NONE	• 表组内的所有表和分区是一个均衡组	• 所有表和分区都分布在一个 LS 上
PARTITON	• 表组内的所有表的一级分区是一个均衡组	• 第一张表的一级分区平均分布到所有的 LS 上 • 其他表的相同分区号的一级分区与第一张表聚集 • 表有二级分区时，一个一级分区下的所有二级分区聚集
ADAPTIVE	• 都是非分区表：与 SHARDING=NONE 相同 • 都是一级分区表：与 SHARDING=PARTITION 相同 • 都是二级分区表：所有表相同一级分区号下的所有二级分区是一个均衡组	都是二级分区表： • 第一张表的每一个一级分区下的所有二级分区平均分布到所有的 LS 上 • 其他表的相同分区号的二级分区与第一张表聚集

租户内均衡控制

☐ 租户内均衡包含日志流均衡和分区均衡两种，其自动均衡的行为受制于该租户的配置项 enable_rebalance 和 enable_transfer 的设置

用户租户的配置项组合	enable_transfer = true	enable_transfer = false
enable_rebalance=true	● 可以执行租户内全部类别的负载均衡。 ● 可以执行扩缩容操作，包括修改 Primary Zone 和 Locality 等。	● 无法合并、分裂日志流，采用特殊的日志流数量不变的均衡策略。 ● 系统不能发起 Transfer 操作，无法执行自动的分区均衡。 ● 在进行租户扩缩容操作时，只能在现有日志流基础上通过日志流迁移实现有限的均衡。
enable_rebalance=false	● 不能执行租户内的负载均衡。 ● 无法执行扩缩容。 ● 可以手工发起 Transfer 操作。	● 不能执行租户内任何类型的负载均衡，无法执行扩缩容。 ● 不可以手工发起 Transfer 操作。

（4.2.1 BP9 之前版本）分区均衡任务的执行还受制于配置项 partition_balance_schedule_interval 的设置

partition_balance_schedule_interval：租户级配置项，控制生成分区负载均衡任务的时间间隔

默认值为 2h。如果用户想要立刻触发分区均衡，需要将配置项调小

如果设置为 0s，表示关闭分区均衡

（4.2.1 BP9 以及之后版本）系统默认每天 24 点自动调度分区均衡任务，用户 也可以手动调整设置

手动负载均衡

手动负载调整

在自动的负载均衡机制以外，OceanBase 数据库还支持通过 ALTER SYSTEM 命令来手动进行负载的调整与均衡。手动负载均衡包括迁移资源单元、迁移日志流副本、日志流切主、迁移分区等

迁移资源单元（MIGRATE UNIT）：将资源单元从一台 OBServer 迁移到另一台 OBServer

迁移日志流副本（MIGRATE REPLICA）：将租户的某个日志流的副本从一个资源单元迁移到另一个资源单元

通常，日志流副本会随着资源单元的迁移同步完成，不需要用户手工干预

日志流切主（SWITCH REPLICA LEADER）：将某一个日志流组的 Leader 副本从一个资源单元切换到另一个资源单元

通常，用户可以通过调整租户的 Primary Zone 设置来影响日志流 Leader 副本的分布

在 V4 版本，不能对单独的分区进行切主操作，可以将分区迁移到 Leader 副本位置满足要求的日志流中，实现分区切主的目的

迁移分区（TRANSFER PARTITION）：将分区从一个日志流迁移到另一个日志流

Transfer Partition 前，需要开启 Transfer 功能，即设置租户配置项 enable_transfer=true

迁移资源单元（MIGRATE UNIT）

OceanBase 数据库管理员可以通过 ALTER SYSTEM MIGREATE UNIT 指令对资源单元进行手动迁移，实现资源分配动态调整、负载均衡的目的。手动迁移资源单元的指令需要在 SYS 租户中执行（手动迁移资源单元可能会与自动均衡策略发生冲突，建议在执行前关闭相应的自动负载均衡策略，即，在 SYS 租户中设置配置项 enable_rebalance=false）。

手动迁移资源单元：ALTER SYSTEM MIGRATE UNIT $unit_id DESTINATION '$server';

ALTER SYSTEM MIGRATE UNIT 1001 DESTINATION '10.10.10.1:2882';

取消进行中的资源单元迁移

ALTER SYSTEM CANCEL MIGRATE UNIT $unit_id

可以通过 DBA_OB_UNIT_JOBS 视图查看正在执行的迁移任务

示例一：将租户 mysql001 在 zone3 中的资源单元从 10.10.10.3:2882 迁移至 10.10.10.4:2882。

使用集群管理员用户登录 SYS 租户

obclient –h10.10.10.1 –P2883 -uroot@sys#obcluster -p'password' -D oceanbase -A -c

查看 mysql001 租户的资源单元的分布

SELECT t.TENANT_NAME,u.ZONE,u.UNIT_ID,u.SVR_IP,u.SVR_PORT FROM oceanbase.DBA_OB_TENANTS t,oceanbase.DBA_OB_UNITS u WHERE t.TENANT_ID=u.TENANT_ID AND t.TENANT_NAME='mysql001';

关闭资源单元的自动均衡

ALTER SYSTEM SET enable_rebalance=false;

示例二：

启动资源单元迁移： ALTER SYSTEM MIGRATE UNIT = 1006 DESTINATION = '10.10.10.4:2882';

在 DBA_OB_UNIT_JOBS 中查看迁移任务的执行状态

SELECT JOB_TYPE, JOB_STATUS... FROM oceanbase.DBA_OB_UNIT_JOBS WHERE JOB_TYPE = 'MIGRATE_UNIT';

查看迁移的结果

SELECT t.TENANT_NAME,u.ZONE,u.UNIT_ID,u.SVR_IP,u.SVR_PORT FROM oceanbase.DBA_OB_TENANTS t,oceanbase.DBA_OB_UNITS u WHERE t.TENANT_ID=u.TENANT_ID AND t.TENANT_NAME='mysql001';

迁移分区（TRANSFER PARTITION）

OceanBase 数据库管理员可以通过 ALTER SYSTEM TRANSFER PARTITION 指令对用户租户的表分区进行手动迁移。Transfer Partition 命令既可以在 SYS 租户执行，也可以在用户租户内执行（手动对分区分布进行调整可能会与自动均衡策略发生冲突，建议在执行前关闭相应的自动负载均衡策略，即，设置相应用户租户的配置项 enable_rebalance=false）

Transfer Partition 语法（ALTER SYSTEM TRANSFER PARTITION TABLE_ID [=] $table_id, OBJECT_ID [=] $object_id TO LS $ls_id [TENANT [=]'$tenant_name'];）

table_id：待迁移的表的 table_id，可以从系统视图 DBA_OB_TABLE_LOCATIONS 中查到

object_id：待迁移的分区的 object_id，object_id 是租户为对象分配的租户唯一的识别号，可以从系统视图 DBA_OB_TABLE_LOCATIONS 中查到

ls_id：要迁往的目的日志流 ID，可以从系统视图 DBA_OB_LS_LOCATIONS 中查到

tenant_name：在 SYS 租户中执行 Transfer 操作时，需要指定对应的用户租户的名称

存储架构原理

存储架构原理

LSM TREE 存储架构

LSM-Tree 的核心思想源自日志结构的存储方式。在传统的数据库系统中，数据更新操

作通常涉及磁盘上的随机读写，这就会导致频繁的磁盘寻道和旋转，从而严重影响性能。而 LSM-Tree 将整个磁盘视为一个日志文件，数据写入时总是追加到日志的末尾，这样可以将众多小文件的分散存储转换成大批量的顺序传输，极大地降低了磁盘寻道的开销，提高了系统效率。

从结构上来看，LSM-Tree 并不是一种严格的树结构，而是一种由内存和磁盘组成的多层存储结构。通常，LSM-Tree 包括以下几个关键组件：

Memtable：位于内存中的数据结构，用于保存最新的数据，支持高速的增删改查操作。MemTable 通常是有序的，常见的实现方式包括跳表（Skip List）和红黑树；

ImmuTable：当 MemTable 达到一定大小时，会被冻结成不可变的 Immutable MemTable，同时创建一个新的 MemTable 继续服务写操作。Immutable MemTable 随后会被异步写入磁盘；

SSTable:LSM-Tree 在磁盘上的持久化存储数据结构，由 Immutable MemTable 转换而来，是一个有序的键值对集合。SSTable 内部包含一系列可配置大小的 Block 块，这些 Block 的索引存储在 SSTable 的尾部，用于快速查找特定的 Block;

WAL：预写日志，用于保证数据的可靠性。在数据写入 MemTable 之前，会先记录到 WAL 中，这样即使系统崩溃，也能通过 WAL 恢复数据。

数据更新与合并

LSM-Tree 的数据更新操作采用日志式的方式，即修改数据时不是直接在原地修改，而是追加一条新记录（为被修改数据创建一个新版本）。因此，在不同的 SSTable 中可能存在相同的 Key 记录，最新的数据总是先写入 MemTable。读取数据时，首先从 MemTable 开始查询，然后从新到旧搜索 SSTable，找到的第一个版本就是该 Key 的最新版本。随着不断的写入，SSTable 的数量会越来越多，这会导致读取数据时需要搜索的 SSTable 也越来越多，同时对于某个 Key 而言，只有最新版本的数据是有效的，其它记录都是冗余的。因此，对 SSTable 进行合并和压缩（Compact）就显得尤为重要。合并操作可以清除冗余的记录，优化存储空间，同时保持层内有序，减少读放大.

LSMTree 工作机制

（一）写入流程

1、数据准备写入：写入操作加入到写前日志（WAL）中，同时数据写入到 MemTable 中。WAL 用于系统崩溃重启时重放操作，保证 MemTable 和 Immutable MemTable 中未持久化到磁盘中的数据不会丢失。

2、MemTable 写满：当 MemTable 达到一定大小时，会被切换为 Immutable MemTable，同时创建一个新的 MemTable 接收写请求。Immutable MemTable 随后会被异步写入磁盘的 Level 0 层。

3、合并操作：当 Level 0 层的 SSTable 数量或大小超过阈值时，会触发合并操作。合并操作将 Level 0 层的 SSTable 与 Level 1 层中有交集的 SSTable 合并，生成新的 SSTable 放入 Level 1 层。这个过程会重复进行，直到达到最高层。

（二）读取流程

1、内存查询：首先在 MemTable 中查询，如果找到则返回结果。

2、磁盘查询：如果 MemTable 中未找到，则从磁盘的各级 SSTable 中依次查询，直到找到结果。查询顺序通常是 MemTable -> Immutable MemTable -> Level 0 SSTable -> Level 1 SSTable -> ... -> 最高层 SSTable。每层先查新生成的 SSTable，每个 SSTable 从后向前查。

（三）合并和压缩策略

LSM-Tree 中的合并与压缩策略是关键，常见的策略包括 Size Tiered Compaction Strategy（STCS）和 Leveled Compaction Strategy（LCS）：

STCS：保证每层 SSTable 的大小相近，同时限制每一层 SSTable 的数量。当某一层 SSTable 数量达到阈值后，将它们合并为一个大的 SSTable 放入下一层。STCS 实现简单且 SSTable 数量较少，但容易出现巨大的 SSTable，同时读放大和空间放大较严重。LCS：限制每一层总文件的大小，并将每一层切分成多个大小相近的 SSTable。SSTable 在层内是有序的，一个 Key 在每一层至多只有一条记录，不存在冗余记录。LCS 层内不存在冗余，所以空间放大较小；因为层内有序，所以在读取时每一层最多读取一个 SSTable，读放大较小。在读取和更改较多的场景下，LCS 压缩策略有着显著优势。

OceanBase 是一款“准内存数据库”，基于 LSM-Tree 的存储架构将数据分为静态基线数据和动态增量数据两部分，并采用读写分离的架构设计。

写数据

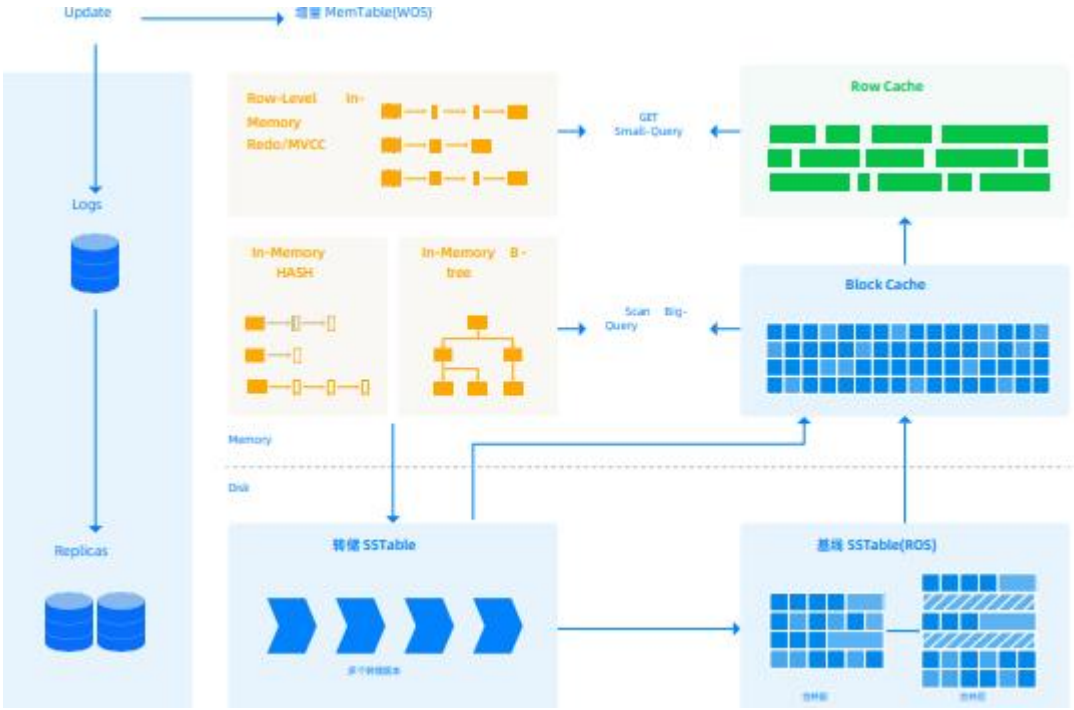
OceanBase 使用 LSMTree 作为存储架构，采取追加更新的策略（append write），不对数据的原始版本进行覆盖更新

写数据时，将数据写入到写内存区（MemStore），通过转储机制将数据落盘，后台线程负责将增量数据与基线数据进行合并。有效避免了写放大和随机写

读数据

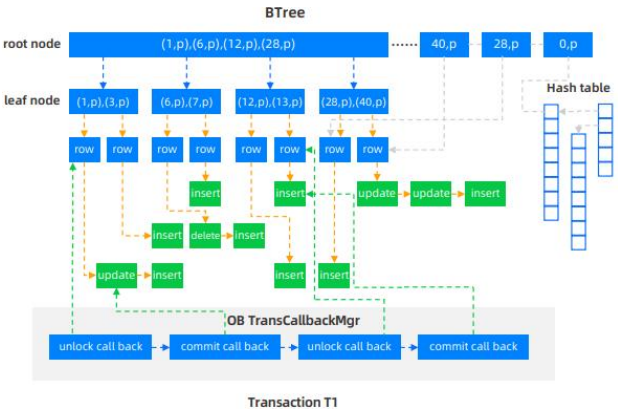
LSMTree 转储会保留数据的多个版本，读取数据时可能需要整合多个版本的数据，读取路径较长

为了解决读数据的性能问题，OceanBase 参考了内存数据库的设计，使用多级读缓存（KVCache）将热点数据尽量缓存在内存中，极大地提升了 LSMTree 的读性能



动态数据（MEMTABLE）

MemTable 是 LSMTree 架构中增量数据在写内存中的组织形式，也叫做动态数据。每一个分区对应一个 MemTable，MemTable 由 BTree 和 HashTable 组成，在插入/更新/删除数据时，数据被写入内存块，在 HashTable 和 BTree 中存储的均为指向对应数据的指针



数据结构	优点	缺点
BTree	BTree 中的数据是有序的，适合范围查找	单行的查找，也需要进行大量的主键比较，理论上性能比 HashTable 慢很多。
Hash Table	可以根据主键快速定位到行，适合单行查找	不适合对范围查询

静态数据（SSTABLE）

SSTable 是 LSMTree 架构中表数据在磁盘中的存储形式。当数据从内存落盘，写入到 SSTable 后，应用程序对数据的修改将不会改变已经落盘的数据版本，因此我们称 SSTable 中的数据为静态数据，SSTable 是由数据宏块和微块组成。

数据宏块（Macro Block）

每个 SSTable 就由若干个宏块构成，宏块是磁盘上大小为 2MB 的定长数据块

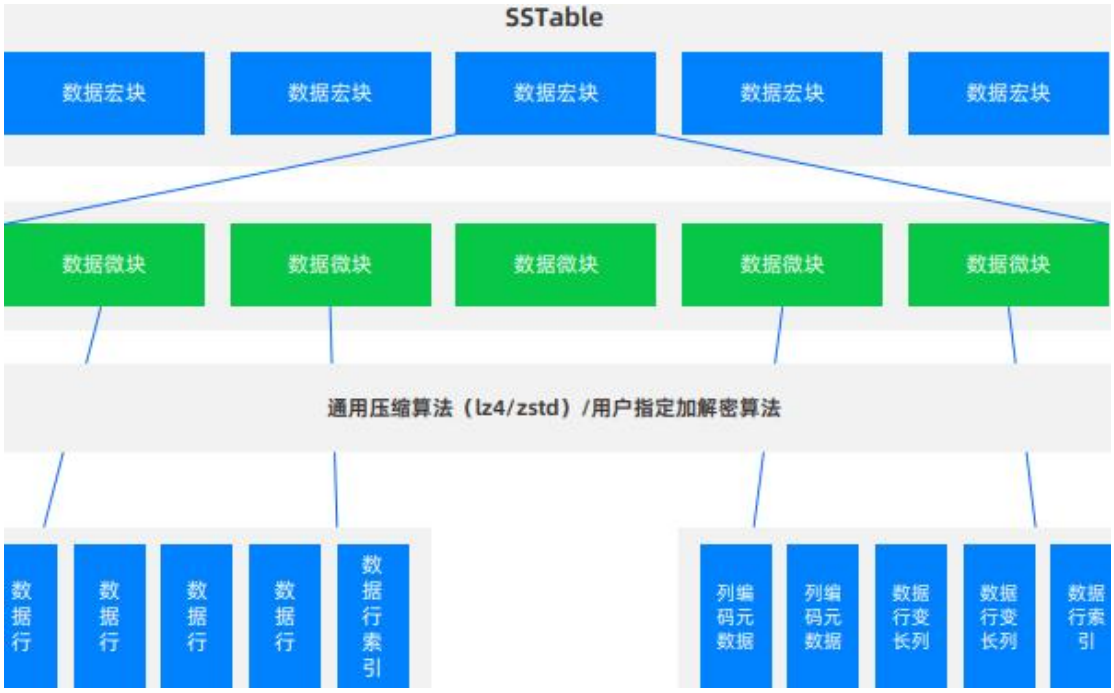
宏块是数据文件写 IO 的基本单位

微块（Micro Block）

每一个宏块内部都是由若干微块组成，微块是大小约 16KB 的变长数据块，其中包含若干数据行（Row）

微块在构建时会根据用户指定的压缩算法进行压缩，因此宏块上存储的实际是压缩后的数据微块

微块是数据文件读 IO 的最小单位



分层转储机制

将 MemTable 中的数据转存到 SSTable 中的过程叫转储。如果 SSTable 只有一层，每次转储都需要将增量数据与静态数据进行归并，转储的性能会随着静态数据量的增多而越来越慢，从而导致 MemTable 内存用满的问题。从 V2.2 版本开始，OceanBase 数据库引入了分层转储策略

L0 层（ Mini SSTable）

内存中的 MemTable 写入磁盘生成 Mini SSTable ，即 L0 层。这个过程叫做 Mini Compaction（转储）

每次转储都生成新的 Mini SSTable，L0 层可以同时存在多个 Mini SSTable

L1 层（Minor SSTable）

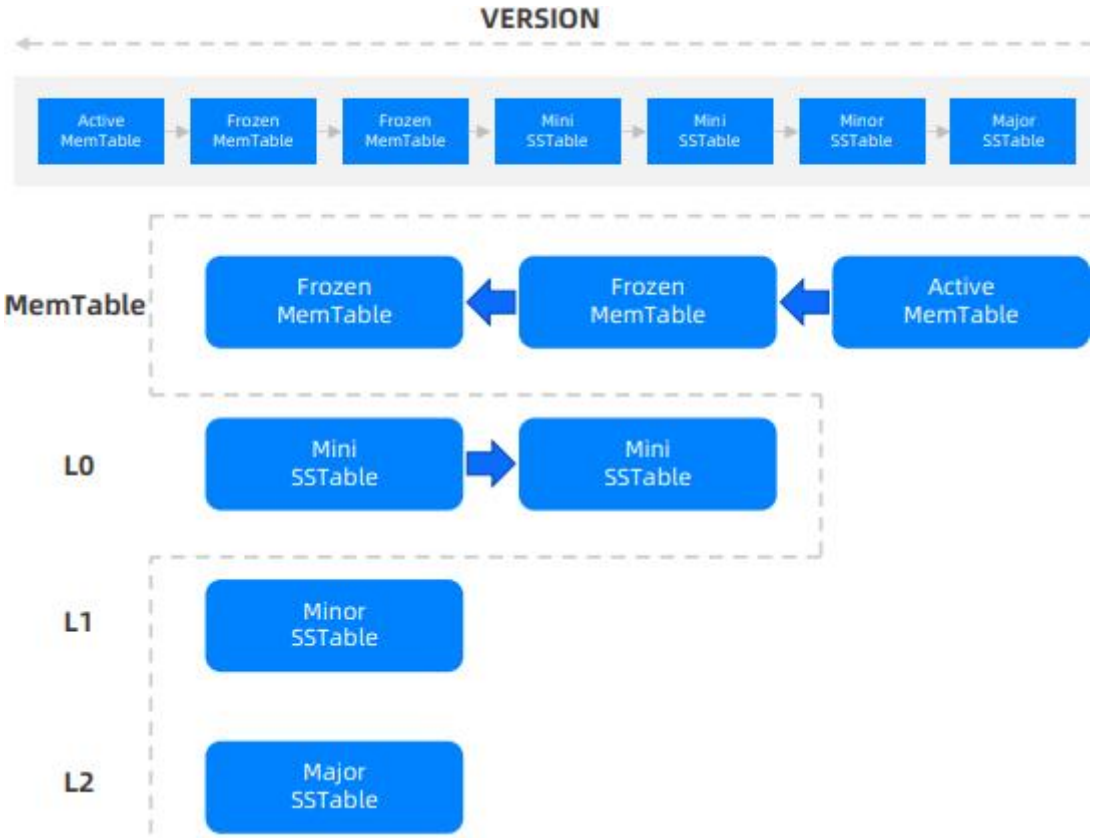
L0 层的多个 Mini SSTable 归并成一个 Minor SSTable，即 L1 层。这个过程叫 Minor Compaction

通常，L1 层只会有一个 Minor SSTable

L2 层（Major SSTable）

L2 层存放基线数据，基线数据由 Major Compaction （合并）产生

通常，L2 层只会有一个 Major SSTable



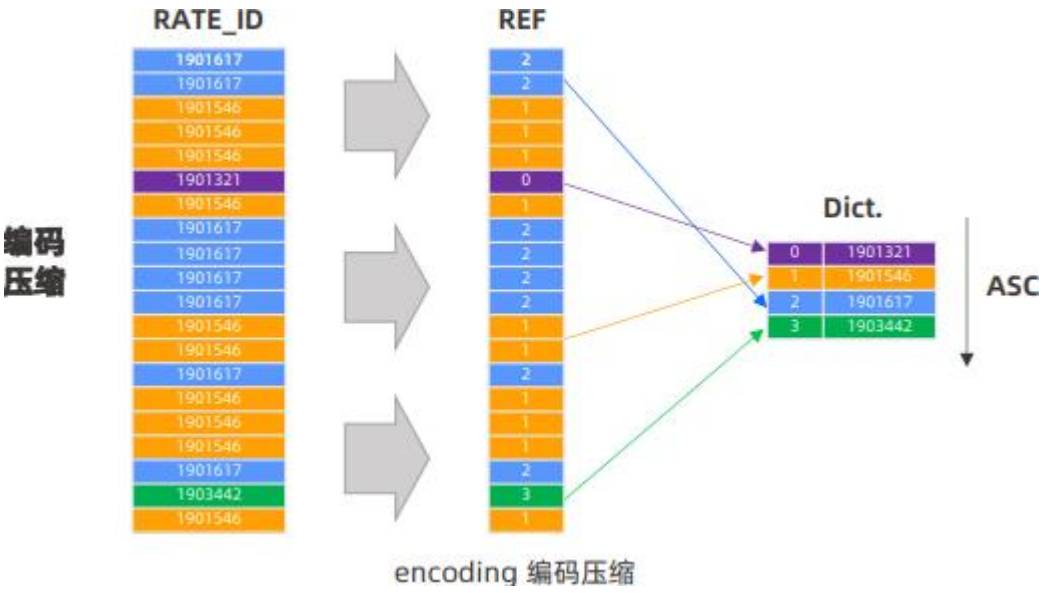
数据压缩存储

OceanBase 数据库默认会对 SSTable 中的数据进行压缩，以节省磁盘存储空间。依托 LSMTree 的存储架构和自研的编码算法，OceanBase 可以实现很好的压缩率，以很低的性能代价实现了存储成本的大幅下降

第一次压缩是编码压缩（encoding），使用了 OceanBase 自研的行列混存编码的压缩方法。仅在合并时可以使用编码压缩

第二次压缩是通用压缩，对 encoding 之后的数据再做一次瘦身。OceanBase 支持 snappy、lz4、lzo，zstd 等压缩算法，允许用户在压缩率和解压缩时间上做各自的权衡

在相同块微块大小（16KB）、相同压缩算法、相同数据的情况下，在 OceanBase 中可以实现远高于传统关系型数据库中的压缩比



Compaction 机制

	Mini Compaction	Minor Compaction	Major Compaction	Adaptive Compaction
级别与范围	• 租户级别（自动触发） • 分区级别（手动触发）	• 分区级别	• 租户级别：所有节点、所有分区同时执行合并	• 分区级别
触发机制	• 自动：根据 MemStore 内存使用情况自动触发 • 手动：通过命令	• 自动：根据 Mini SSTable 的个数与数据量来触发 • 手动：不可手动	• 自动：转储次数达到阈值 • 定时：每天在固定的时间 • 手动：通过命令	• 自动：根据分区的数据读写行为自适应触发 • 手动：不可手动
全局快照	不产生全局快照	不产生全局快照	产生租户级的全局快照	产生分区级的全局快照
多版本数据	不回收多版本数据	回收多版本数据	回收多版本数据	回收多版本数据
数据压缩	通用压缩	通用压缩	encoding 编码压缩 + 通用压缩	encoding 编码压缩 + 通用压缩

Mini Compaction（转储）

当 MemTable 的大小超过一定的阈值时，OceanBase 数据库自动执行转储（Mini Compaction，也叫做 Mini Merge），将数据落盘并释放 MemTable 的内存

自动转储机制

租户级配置项 freeze_trigger_percentage 控制转储的触发时机。当一个租户的 MemTable 内存的使用量与 MemStore 内存上限的比例超过配置设定的值时，该租户即会触发转储

freeze_trigger_percentage 的默认值为 20%，每个租户可以设置不同的阈值

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，这个过程称为冻结（Freeze），冻结会阻止当前活跃的 MemTable 再有新的事务写入，同时会生成新的活跃 MemTable 以供新的事务使用

OceanBase 自动发起转储操作时，会将 MemStore 中的所有分区的 MemTable 均冻结

和转储

OceanBase 在所有冻结的 MemTable 都转储成功后统一释放这些 MemTable 占用的内存

自动转储的特点

转储是租户级别的，不是集群级别的。每个租户根据自己的 MemStore 使用情况单独触发转储

转储是 OBServer 级别的，不是全局级别的。每个 OBServer 根据租户在该节点的 MemStore 使用情况单独触发转储

可以对未提交事务的更新数据做转储，转储成功后即可对相关的事务日志做回收

基于转储的速度考虑，转储时不会对多版本数据进行回收处理，不做 encoding 编码压缩，但会使用通用压缩算法对数据压缩存储

关于转储频率的思考

提高转储次数

优点

宕机恢复时间短：增量数据更快落盘，避免宕机恢复时读取大量的 Redo 日志来恢复数据，提升宕机恢复的速度

避免 MemStore 用满：提升转储的次数，降低 MemStore 的内存使用量，避免在批量更新数据时 MemStore 内存被用满或限速的发生

避免 IO 洪峰：降低每次转储的数据量，避免一次转储大量的数据对系统 IO 造成压力

适用场景

对于数据更新量较高的小内存租户，通过提高转储频率，避免内存用满

对于大量数据更新的场景，通过增加转储次数、降低单次转储的数据量，来提升转储的速度，避免内存用满

减少转储次数

优点

缓存热点数据：将更新的数据更久地保留在 MemStore 中，查询可以从 MemStore 中快速读到数据，避免去磁盘上读取，提升查询性能

缩短查询路径：通过减少转储次数，减少同一条记录在多级 SSTable 中的“碎片”数，缩短查询路径，提升查询性能

避免 IO 压力：通过减少转储次数，减少转储对系统 IO 的使用，避免转储对系统 IO 的持续影响

适用场景

对于数据更新不频繁的租户，通过降低转储频率，提升数据读取的速度

对于转储频繁导致磁盘 IO 一直繁忙的租户，通过降低转储频率，缓解 IO 压力

手动转储

在自动转储机制之外，OceanBase 数据库还支持通过 ALTER SYSTEM MINOR FREEZE 命令来手动进行转储。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。转储命令既可以在 SYS 租户执行，也可以在用户租户执行

MINOR FREEZE 语法

ALTER SYSTEM MINOR FREEZE [tenant_list] [server_list] [ZONE = zone_name] [LS = ls_id] [TABLET_ID = tablet_id]

tenant_list:TENANT [=] { all | all_user | all_meta } | tenant_name_list

tenant_name_list:tenant_name [, tenant_name ...]

server_list:SERVER [=] ('ip:port' [, 'ip:port'...])

tenant_list：在 SYS 租户中对用户租户执行转储时，需要指定对应的用户租户的名称可以同时指定多个租户，例如，TENANT=tenant1,tenant2

TENANT=all_user 代表所有用户租户，TENANT=all_meta 代表所有 Meta 租户

server_list：执行转储的 OBServer。可以同时指定多个 OBServer，例如，SERVER=('10.10.10.1:2882',' 10.10.10.2:2882 ')

zone_name：执行转储的 Zone，一次仅支持指定一个 Zone

ls_id：执行转储的日志流 ID，需要同时指定 TENANT 名字

tablet_id：执行转储的分区所对应的 Tablet ID，需要同时指定 TENANT 名字

手动转储的示例

示例：在 SYS 租户执行转储操作

转储命令	说明
ALTER SYSTEM MINOR FREEZE;	系统租户对本租户发起转储
ALTER SYSTEM MINOR FREEZE TENANT = all_user;	对所有用户租户的所有 OBServer 同时执行转储
ALTER SYSTEM MINOR FREEZE TENANT = obmysql,oboracle;	对指定租户在其所有的 OBServer 上执行转储
ALTER SYSTEM MINOR FREEZE TENANT = obmysql ZONE = zone1;	对指定租户在指定 Zone 内的所有资源单元（OBServer）上执行转储
ALTER SYSTEM MINOR FREEZE TENANT = obmysql SERVER=('10.10.10.1:2882',' 10.10.10.2:2882 ')	对指定租户在指定的 OBServer上执行转储
ALTER SYSTEM MINOR FREEZE TENANT =obmysql LS = 1001;	对指定租户的特定日志流所在的所有 OBServer 上执行转储
ALTER SYSTEM MINOR FREEZE TENANT = obmysql TABLET_ID = 200001;	对指定租户内的特定分区在所有相关的 OBServer 上执行转储

***注意**：在 4.3 版本前，用户租户仅可对本租户发起**租户级别**的转储，在用户租户中执行转储命令时不能指定 TENANT。*

手动转储的场景

虽然 OceanBase 数据库的自动化管理程度较高，能够自动处理很多维护任务以保证数据库的高效运行，但在某些特定场景下，可能需要手动发起转储操作

内存使用紧张：当系统内存使用率很高，而自动转储由于某些原因未及时执行，此时可以手动发起转储操作，避免内存资源耗尽

日志盘高水位：在转储前，系统无法回收未提交事务的日志。通过主动转储，推动日志回收，避免日志盘被用满

业务负载高峰前：在业务需求预期增加的高峰期来临前，提前手动触发转储操作可以减少高峰期的系统负载，避免在关键业务时间段内触发自动转储带来的性能波动

重启 OBServer：在重启 OBServer 前主动转储，将内存中的增量数据写入磁盘保存，以缩短 OBServer 重启时的数据恢复时间

系统维护和变更：进行系统维护升级、扩容缩容或配置变更前主动转储，确保数据的一致性和完整性，降低数据丢失的风险

数据物理备份：在执行物理备份前主动转储，将最新的数据写入磁盘，保证数据的最新状态被安全备份

Minor Compaction

随着转储次数的增加，L0 层的 Mini SSTable 的数量和数据量越来越多，查询的效率会变差。OceanBase 数据库在满足一定条件后会自动将 L0 层的多个 Mini SSTable 合并，或与 L1 层的 Minor SSTable 合并成一个新的 Minor SSTable，这个过程叫做 Minor Compaction，也叫做 Minor Merge。

Minor Compaction 的作用

减少 SSTable 个数：将 L0 层的多个 Mini SSTable 合并为 1 个，或者更进一步与 L1 层的 Minor SSTable 合并。

回收多版本数据：对多版本数据进行事务状态的回填，并根据多版本数据的保留要求对过期数据进行回收，减少总的数据量。

通过减少 SSTable 的个数以及多版本数据的数据量，达到提升查询性能的目的

Minor Compaction 的特点

由 OceanBase 数据库自动触发，无法手动执行

是分区级别的操作，由 OceanBase 根据各个分区的 SSTable 的特征自动触发，不是租户或者集群级别的统一操作

基于效率的考虑，不做 encoding 编码压缩，但会使用通用压缩算法对数据压缩存储

Minor Compaction 的触发机制

OceanBase 数据库根据 L0 层 Mini SSTable 的数量以及其数据总量与 L1 层 Minor SSTable 的数据量的占比动态执行两种 Compaction 策略。

两种 Compaction 策略：

第一种（Mini Minor Compaction）：当 L0 层 Mini SSTable 的个数超过配置项 minor_compact_trigger 的值时，将 L0 层的 MiniSSTable 合并为一个大的 Mini SSTable

minor_compact_trigger 是租户级配置项，默认值为 2

第二种（Minor Compaction）：在满足第一种策略的同时，且 L0 层 Mini SSTable 的记录总数超过 L1 层 Minor SSTable 记录数的 25% 时，L0 层的多个 Mini SSTable 与 L1 层的 Minor SSTable 直接合并为一个新的 Minor SSTable

Major Compaction（合并）

Major Compaction（合并，也叫做 Major Merge）是将所有的动、静态数据（MemTable，L0，L1，L2）做归并，形成一个全新的版本一致的基线数据（Major SSTable）的过程。

合并的作用

合并会删除失效的多版本数据，生成按照主键排序的新的基线 SSTable，有利于提升查询的性能

合并过程中，OceanBase 会对数据进行校验，并执行 Schema 变更对应的数据变更

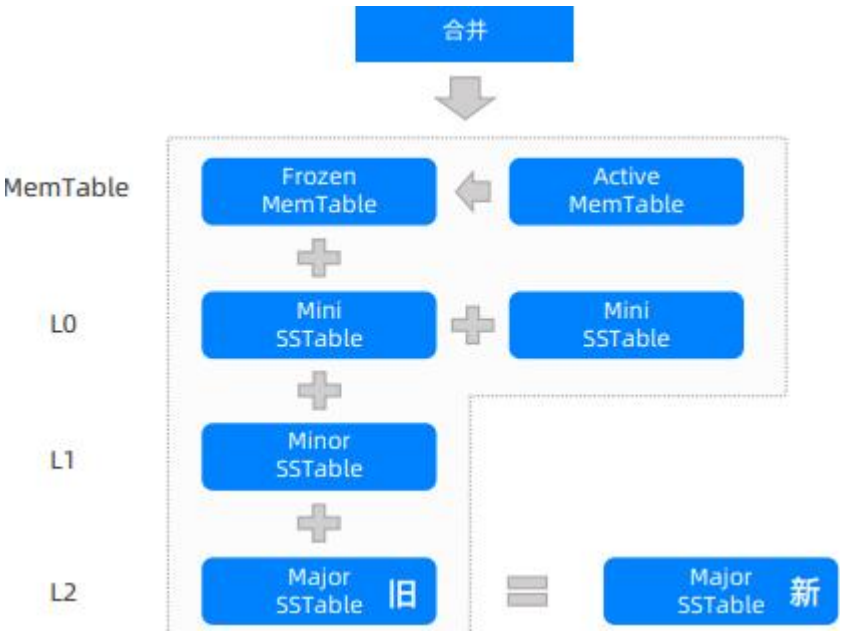
合并会对数据进行编码压缩和通用压缩，降低存储空间的占用

合并的特点

在 V4 版本中，不是集群级别的，每个租户按照各自的设定单独触发

合并是全局级别的，所有副本需要同时执行合并操作

与转储相比，合并是一个重资源消耗的操作，要尽量在业务低估期进行



合并的执行方式

合并在执行时，按照处理的宏块的不同，可以细化为全量合并、增量合并，渐进合并三种方式。

全量合并

合并在执行时，按照处理的宏块的不同，可以细化为全量合并、增量合并，渐进合并三种方式。

全量合并会极大的耗费磁盘 IO 和空间，OceanBase 通常不会主动采取这种合并方式。

增量合并

只会读取被修改过的宏块数据，和动态数据归并，并写入磁盘，对于未修改过的宏块，则直接重用。

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

每次只合并一部分需要更新的宏块数据，经过若干轮次后将全部数据重写一遍。

主要用于 Online DDL 操作导致的数据变更，比如加减列。通过将 DDL 变更造成的数据重写分散到多次每日合并中去做，减小每日合并的工作量。

控制渐进合并

OceanBase 数据库分别提供了租户级和表级的渐进合并控制选项，可以控制表上渐进合并的轮次。

租户级配置项 default_progressive_merge_num 用于设置新建一张表时默认的渐进合并次数，其默认值为 0

值为 0 时，表示使用默认值 100

值为 1 时，表示强制执行全量合并，不执行渐进合并

值为 X 时，表示 X 轮合并重写完全部数据，每轮合并重写 1/X

在创建表或修改表时，可以指定表的属性 progressive_merge_num 来指定表上渐进合并的发生轮次

表属性 progressive_merge_num 的取值规则与租户配置项 default_progressive_merge_num 一致

如果表属性 progressive_merge_num 未被设置，则使用租户配置项 default_progressive_merge_num 值

示例

示例 1：ALTER SYSTEM SET default_progressive_merge_num=30; #设置本租户的默认渐进合并轮次为 50

示例 2：CREATE TABLE tbl1 (C1 VARCHAR(100)); #表在创建时继承租户的渐进合并轮次为 50

示例 3：ALTER TABLE tbl1 SET progressive_merge_num=1; #修改表的渐进合并轮次为 1，即下次合并执行全量合并

合并触发机制

在 OceanBase V4.2 版本中，合并的触发方法包含：自动触发合并、定时触发合并、手动触发合并，以及自适应合并。

触发机制

自动触发

合并范围：租户级全局合并

触发条件

集群级全局合并开关 enable_major_freeze = true 且

租户的转储次数已达到了租户级配置项 major_compact_trigger 设置的上限时，下一次触发转储时直接进行合并

默认值为 0，表示无论转储多少次都不会自动触发合并

定时触发

合并范围：租户级全局合并

触发条件

集群级全局合并开关 enable_major_freeze = true 且

系统时间达到了租户级配置项 major_freeze_duty_time 设置的每日合并的时间点时，就会自动触发合并

默认值为 02:00，表示每日凌晨 02:00 自动触发合并

手动触发

合并范围：租户级全局合并

触发条件

集群级全局合并开关 enable_major_freeze = true，且

手动执行合并命令：ALTER SYSTEM MAJOR FREEZE

手动合并

OceanBase 数据库支持通过 ALTER SYSTEM MAJOR FREEZE 命令来手动执行合并。手动合并是租户级的全局合并，既可以在 SYS 租户执行，也可以在用户租户执行（用户租户仅可对本租户发起合并，在用户租户中执行合并命令时不能指定 TENANT）

MINOR FREEZE 语法：ALTER SYSTEM MAJOR FREEZE [tenant_list]

tenant_list：在 SYS 租户中对用户租户执行合并时，需要指定对应的用户租户的名称

可以同时指定多个租户，例如，TENANT=tenant1,tenant2

TENANT=all_user 代表所有用户租户，TENANT=all_meta 代表所有 Meta 租户

ALTER SYSTEM MAJOR FREEZE; 系统租户对本租户发起合并

ALTER SYSTEM MAJOR FREEZE TENANT = all_user；对所有用户租户同时发起合并

ALTER SYSTEM MAJOR FREEZE TENANT = obmysql,oboracle; 对指定租户发起合并

手动合并的场景

合并通常是一个比较耗时的操作，会占用大量的系统资源（IO、CPU、内存等），对正在执行的业务影响较大，因此 OceanBase V4.2 版本默认只开启每日定时合并以及分区级的自适应合并。但在以下场景下，我们需要主动执行合并

大量数据变更：当表数据频繁变更（如大量插入、更新或删除操作），产生大量的多版本数据，通过合并消除无效的多版本数据，形成统一版本的基线数据，提升查询的性能

表结构变动频繁：当表结构频繁变动后，磁盘存储的数据可能对应多个 Schema 版本，通过全量合并形成统一版本的基线数据，提升查询的性能

业务峰值负载前：在业务需求预期增加的高峰期来临前，提前手动触发合并操作可以减少高峰期的系统负载，避免在关键业务时间段内触发自动合并带来的性能波动

存储空间紧张：合并操作会进行数据压缩和整理，能够释放部分存储空间。如果发现存储空间紧张，可以通过手动触发合并操作来优化存储空间使用

系统维护和变更：在进行系统维护升级、扩容缩容或配置变更前，手动触发合并操作可以确保数据的持久化状态，减少维护过程中意外情况带来的风险

数据物理备份：在执行物理备份前主动合并，确保数据的最新状态被安全备份并减小数据备份的空间占用

Adaptive Compaction（自适应合并）

OceanBase 数据库从 V4.2.0 版本开始支持 Adaptive Compaction（自适应合并，也叫 Medium Merge）。开启自适应合并后，系统会根据分区上的读、写行为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。

自适应合并的特点

自适应合并是分区级的

自适应合并是自动触发，不能手动发起

自适应合并可以利用 encoding 编码压缩来压缩数据

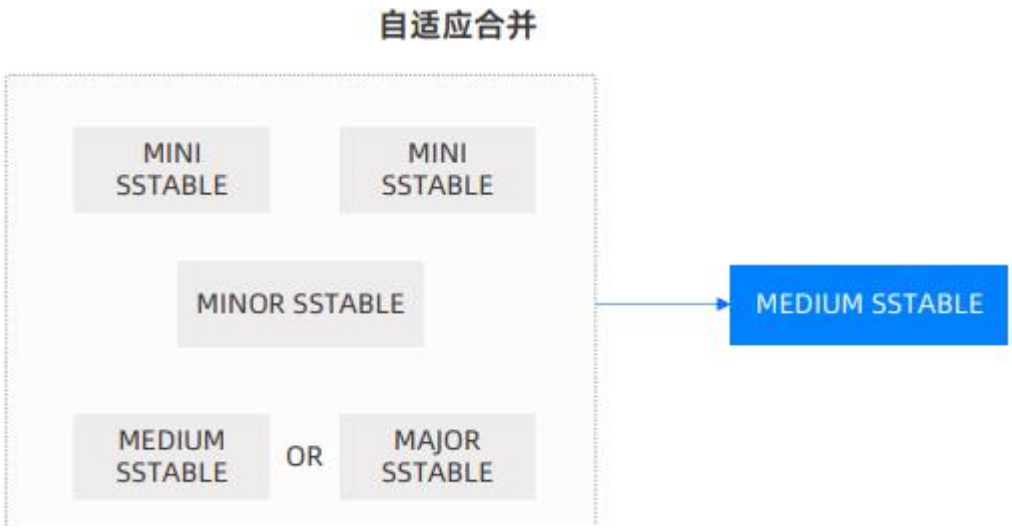
自适应合并只保留合并时的数据版本，没有多版本数据

自适应合并解决的问题

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



Medium SSTable：自适应合并产生的归一 SSTable，可以取代合并产生的 Major SSTable

Buffer 表的自适应合并策略

对于 V4.2.1 版本，从 V4.2.1 BP5 版本开始支持通过表选项 table_mode 指定自适应合并策略的功能，以更好地解决 Buffer 表的性能问题

Queuing 表效应：当用户在某张表上频繁地执行插入并且同时进行批量删除，或者有大量的并发更新操作时，可能会遇到一种现象：表中的数据行数并不大，但是查询和更新的性能出现明显下降。这种现象在 OceanBase 数据库中称为 Queuing 表（Buffer 表）效应

问题原因：Buffer 表是 LSM-Tree 架构数据库都要面对的一类问题。当增量数据中存在大量标记删除的数据时，物理行数量将远多于逻辑行，从而造成严重的读放大现象，并影响优化器执行方案的生成

解决方案：在自适应合并机制的基础上，为了更加灵活地解决 Buffer 表带来的性能下

降问题，OceanBase 数据库支持通过表选项 table_mode 为每张表设置不同的自适应合并策略，来应对 Buffer 表引起的读放大现象

table_mode	触发自适应合并的概率
normal（默认值）	极低
queuing	低
moderate	中
super	高
extreme	极高

指定表的自适应合并策略

如果您对业务比较熟悉和了解，您可以在创建表或修改表时根据业务情况，为表设置一个合适的 table_mode

创建一张 queuing 模式的表

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

调整表的 table_mode 属性

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

修改表的自适应合并策略的建议

当某张 Buffer 表出现比较明显的读放大问题时，建议手动修改该表的 table_mode，以便系统更频繁地触发该表的合并，加速查询

合并的频率越高，消耗的系统资源越多，建议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

管理 Compaction 任务

OceanBase 数据库通过专门的后台任务（DAG）来执行各类 Compaction 操作，并按照不同 Compaction 操作的优先级进行分组管理。OceanBase 提供了相应的租户级配置项来管理各类 Compaction 操作的并发线程数。

compaction_low_thread_score：低优先级线程（Major Compaction）的权重，当前直接表示线程数，默认值 0（实际 6）

compaction_mid_thread_score：中优先级线程（Minor Compaction）的权重，当前直接表示线程数，默认值 0（实际 6）

compaction_high_thread_score：高优先级线程（Mini Compaction）的权重，当前直接表示线程数，默认值 0（实际 6）

可以调大相应 Compaction 线程数量来加速 Compaction 操作。比如，如果租户的 MemStore 内存使用紧张，可以调小 freeze_trigger_percentage 来增加转储的频率、减小每次转储的数据量，也可以调大 compaction_high_thread_score 来增加转储的线程数，提升转储的速度

关于 Compaction 的系统视图

可以在 SYS 租户和用户租户使用以下系统视图查看各类 Compaction 的执行状态以及历史执行信息。（在 SYS 租户中使用 CDB_OB 为前缀的视图代替 DBA_OB 视图，可以来查看所有租户的信息）

DBA_OB_MAJOR_COMPACTION 展示租户的全局合并状态信息

DBA_OB_ZONE_MAJOR_COMPACTION 展示本租户各个 Zone 的合并状态信息

GV$OB_COMPACTION_PROGRESS 展示本租户的 OBServer 级合并进度信息

GV$OB_TABLET_COMPACTION_PROGRESS 展示 TABLET 级 COMPACTION 的进度信息

GV$OB_TABLET_COMPACTION_HISTORY 展示 TABLET 级 COMPACTION 的历史信息

内存分配机制

内存分配机制

OceanBase 是原生的多租户数据库，对大容量内存的管理和使用提出了很高要求。通常，我们需要将服务器、Docker 的绝大部分内存分配给 OBServer，并预留一部分内存给数据库系统运行，剩下的内存均可分配给租户。

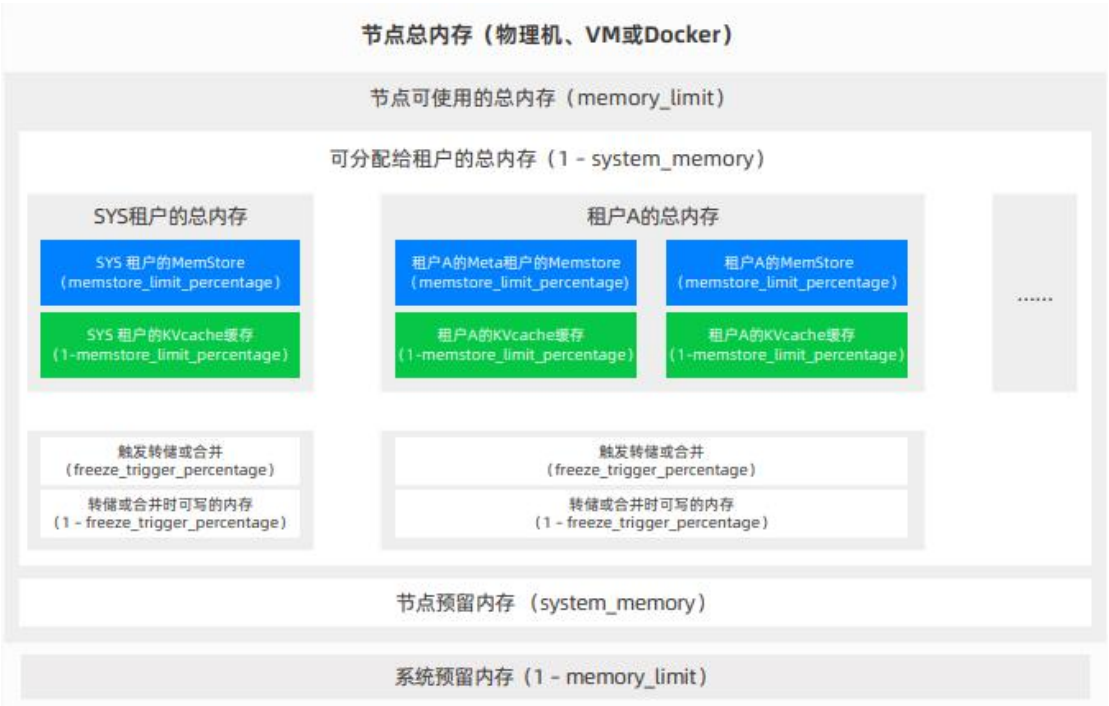
OBServer 内存分配

在为操作系统预留一部分内存外，将服务器节点的大部分内存（memory_limit 或 memory_limit_percentage）分配给 OBServer

OBServer 内存中预留一部分给数据库系统运行（节点预留内存，system_memory），剩余的内存可以给配给租户

每个租户等同于传统数据库的一个实例。租户内存主要分为装载增量数据的 MemStore 以及读缓存 KVCache

用户租户都有对应的 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。系统租户不存在对应的 Meta 租户



节点预留内存

节点预留内存是预留给数据库系统的公用模块使用的内存，即数据库系统内部内存

系统内部内存

虽然 OceanBase 是多租户数据库，大部分的数据库功能都在租户内完成，但是依然存在一些所有租户共用的功能模块。这些功能模块使用的内存并不属于任何一个租户，因此被称为系统内部内存

集群配置项 system_memory 指定了 OBServer 为系统内部内存预留的内存大小（注意：是预留内存，非使用上限）

system_memory 实际上限制的是 OBServer 可以分配给租户使用的内存上限，即 memory_limit - system_memory

system_memory 默认值为 0，表示 OceanBase 数据库根据 OBServer 内存大小自适应地预留系统内部内存

500 租户：为了便于管理，我们将租户共用的功能模块及其资源划归到一系列虚拟租户，其租户 ID 是 500~509，统称为 500 租户

租户内部内存

OceanBase 数据库的租户内部的内存总体上分为两类，可伸缩内存（KVCache 等）与不可伸缩内存（MemStore 等）

不可动态伸缩的内存

主要给写操作保存增量数据的 MemStore 使用，还包括 SQL 执行时使用的 Plan Cache（执行计划缓存）、SQL Area（SQL 执行期内存）等内存，此类内存模块需要在其相应的内存限制下使用内存

memstore：大小由参数 memstore_limit_percentage 决定，表示租户的 MemStore 部分占租户总内存的百分比

可动态伸缩的内存

主要是给读操作缓存数据的 KVCache 使用。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

kvcache：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

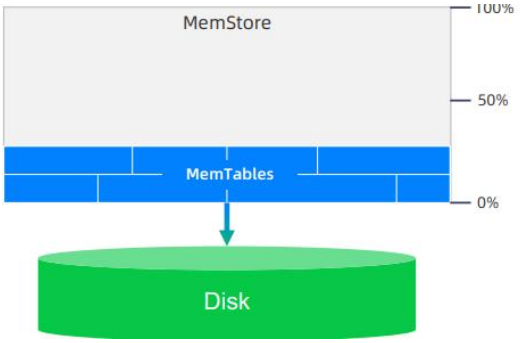
MEMSTORE

OceanBase 数据库的存储引擎基于 LSM Tree 架构，动态增量数据存储在 Memstore 内存中，静态数据存储在磁盘上

MemStore 的大小由集群级配置项 memstore_limit_percentage 控制，表示 MemStore 内存占租户总内存的比例，默认值是 50

MemStore 内部由若干 MemTable 组成，一个 MemTable 存储同一个分区的增量数据

MemStore 中增量数据的内存使用达到租户级配置项 freeze_trigger_percentage 设置的值时，OceanBase 数据库会触发转储操作，将增量数据写入磁盘



使用示例

假设租户的资源规格和相关参数定义如下：

- 资源单元的 Memory_Size 为 30 G
- memstore_limit_percentage 为50%
- freeze_trigger_percentage 为 30%

当 MemStore 中的增量数据达到 30G * 50% * 30% = 4.5G 时，OceanBase 数据库将对该租户进行一次转储操作，将增量数据落盘后释放 MemStore 的内存。

MEMSTORE 写入限速

在大批量更新、插入数据的场景，如果转储的速度跟不上 MemStore 的内存申请速度，可能会出现 MemStore 内存被用满的情况，导致应用执行失败。为了避免这种情况的发生，OceanBase 提供了 MemStore 的写入限速机制

MemStore 限速机制

MemStore 的内存使用率达到 writing_throttling_trigger_percentage 设置的阈值时，

OceanBase 会主动延缓 MemStore 的内存申请，以期待转储完成并释放内存，避免 MemStore 被用满

writing_throttling_trigger_percentage：租户级配置项。默认值是 60，即当 MemStore 内存使用率达到 60%时开启限速。如果设置为 100，则表示关闭限速

避免 MemStore 用满的策略

加快转储：降低 freeze_trigger_percentage 的阀值，增加转储的频次

写入限流

降低应用的并发度或者更新的数据量

OceanBase 侧开启 MemStore 限速机制进行防御

KVCACHE

OceanBase 为了提升读数据的速度，设计了多级缓存机制，由 KVCache 统一管理。KVCache 是动态伸缩的内存，当租户内存充足时，KVCache 内存弹性增长；当租户内存满时，KVCache 会淘汰未被使用的数据来释放内存

多级缓存机制，KVCache 是一个只读 Cache，用于缓存热点数据。为了实现“准内存数据库”的目的，KVCache 包含多级缓存

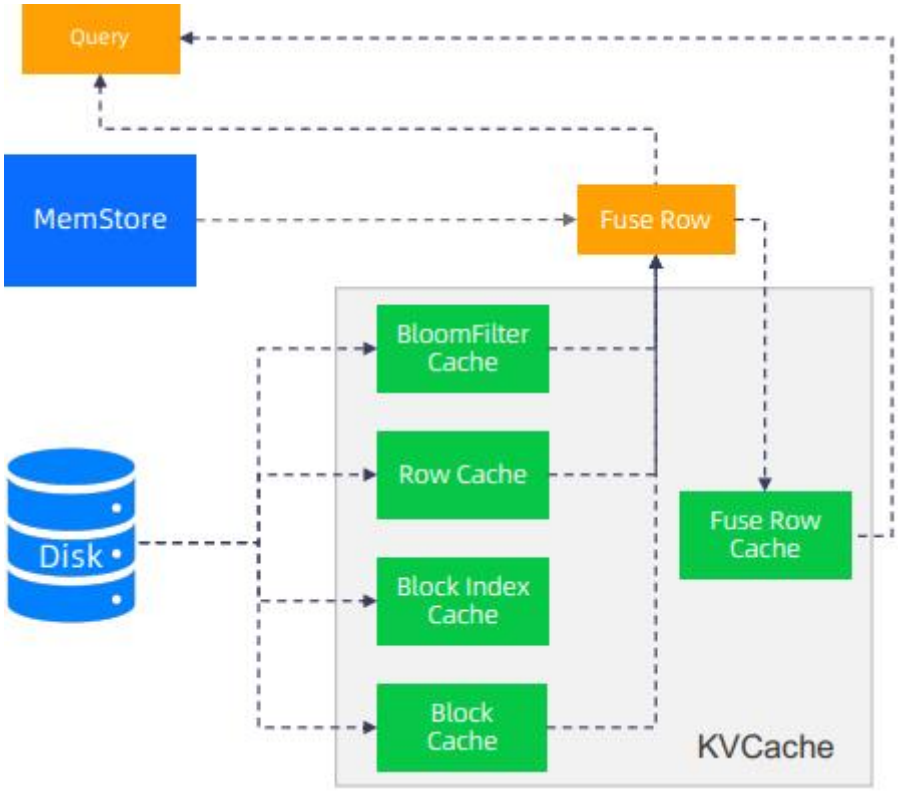
BloomFilter Cache：建立在宏块上的布隆过滤器，提高空查询的效率

Row Cache：缓存热点数据行

Block Cache：缓存解压后的热点数据微块

Block Index Cache：缓存热点微块的索引，加速微块数据的访问

Fuse Row Cache：缓存行的快照点数据，用于提高点查询性能



SQL WORK AREA

SQL Work Area 是 SQL 工作线程的工作区内存，包含了 SQL 解析和执行时的内存使用

SQL 工作区内存由租户变量 ob_sql_work_area_percentage 控制

SQL Work Area 内存 = memory_size * ob_sql_work_area_percentage

ob_sql_work_area_percentage 默认值为租户总内存的 5%

SQL 工作区内存主要用于 SQL 执行算子处理、存放中间结果集，常见有以下场景

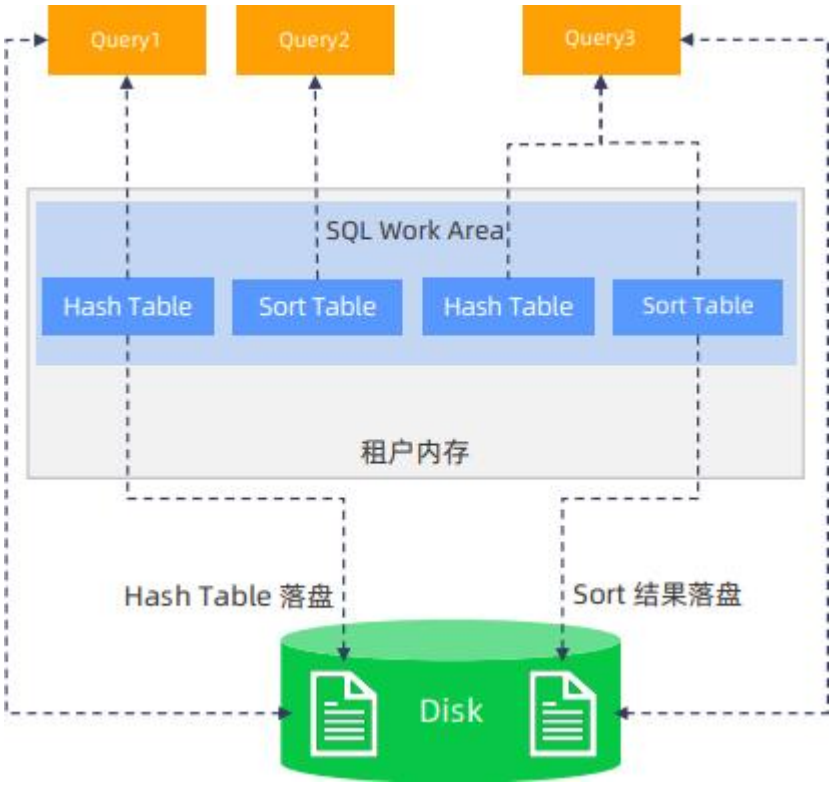
对大量数据的排序操作，比如 Sort，Union 和 Group

子查询的物化操作，比如 Material

Join 时对内外表进行的排序，比如 Merge Join

Join 时对中间结果建立 Hash Table，比如 Hash Join

OceanBase 会限制每个 SQL 执行 Sort、Hash 运算时使用的工作区内存大小，当 SQL 需要处理的数据量过大时会将运算的结果落盘存放。但是，如果并发执行 Sort、Hash 处理的 SQL 数过高，依然可能导致 SQL 工作区内存分配失败



PLAN CACHE

对于执行频次较高且执行时间较短的 SQL，如果每次执行都生成执行计划，SQL 优化的耗时会严重拖累 SQL 性能。OceanBase 将生成的计划放进 Plan Cache，再次执行时直接从 Plan Cache 获取 Plan

OceanBase Plan Cache 的特点

计划缓存是一个典型的 Key-Value 结构，Key 就是参数化后的 SQL 字符串，Value 就是该条 SQL 所对应的执行计划

每个租户在每个节点上都有一个独立的 Plan Cache，用以缓存在此节点上处理过的

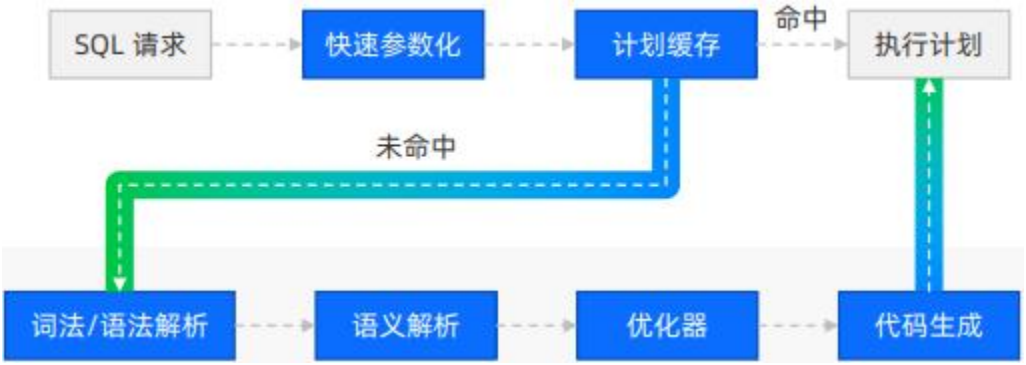
SQL Plan

Plan Cache 内存由租户变量 ob_plan_cache_percentage 控制

Plan Cache 内存 = memory_Size * ob_plan_cache_percentage

ob_plan_cache_percentage 默认值为租户总内存的 5%

当 Plan Cache 内存使用较高时，OceanBase 会自动淘汰旧的 Plan。如果所有的 Plan 都在被使用中，则无法执行淘汰



关于内存分配与使用的系统视图

在 SYS 租户中可以来查看所有租户的信息，在用户租户中仅显示本租户的信息

GV$OB_TENANT_MEMORY 性能视图 展示租户的内存使用统计

GV$OB_MEMORY 性能视图 展示租户内各个内存模块的内存使用统计

GV$OB_MEMSTORE 性能视图 展示租户的 MemStore 的内存使用统计

GV$OB_KVCACHE 性能视图 展示租户的 KVCache 的内存使用统计

GV$OB_SQL_WORKAREA_MEMORY_INFO 性能视图 展示租户的 SQL Work Area 的内存使用统计

GV$OB_PLAN_CACHE_STAT 性能视图 展示租户的 Plan Cache 的内存使用统计

SQL 执行原理

SQL 引擎与优化器

兼容模式

OceanBase 数据库的 SQL 引擎在一个系统中可同时支持 MySQL 模式和 Oracle 模式两种模式的租户。用户在创建租户时，可选择创建 MySQL 兼容模式的租户或 Oracle 兼容模式的租户，租户的兼容模式一经确定就无法更改，所有数据类型、SQL 功能、视图等相应地与 MySQL 数据库或 Oracle 数据库保持一致。

同一个集群,同时支持 MySQL 和 Oracle（OceanBase 数据库社区版仅提供了 MySQL 模式。）

租户创建时需要配置为 MySQL 兼容模式或 Oracle 兼容模式

DBA 由原来维护“多个数据库产品”变为维护一个“统一的数据库产品”，DBA 可以结合

应用需求，创建不同兼容模式的租户

SQL 请求执行流程

SQL 引擎从接受 SQL 请求到执行的典型流程如下图所示：

快速参数化：将语句中的常量变成变量，并基于参数化后的 SQL 文本生成 SQL_ID

语法解析：将 SQL 请求转换为语法树，检查 SQL 语法

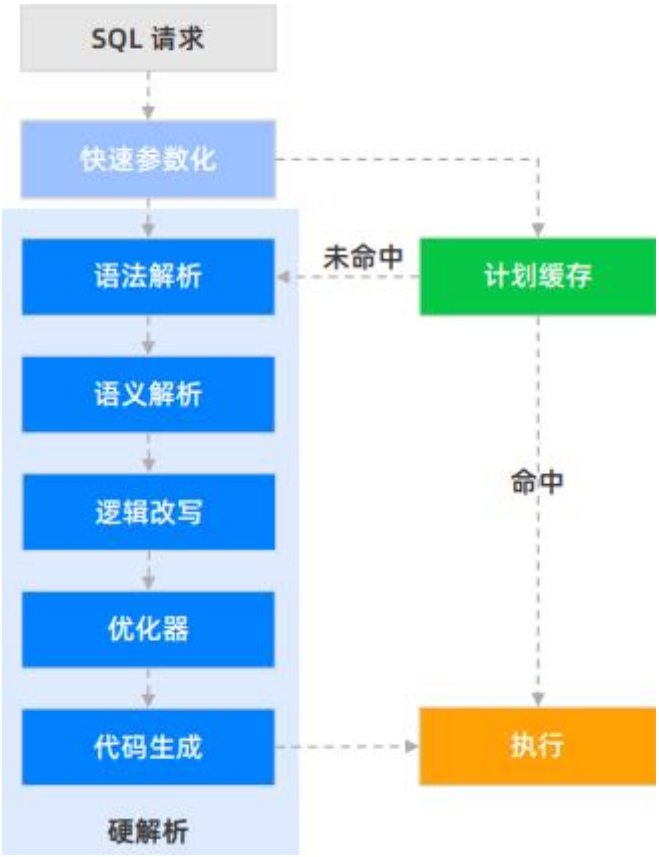
语义解析：将语法树转换为带有数据库语义信息的内部数据结构

逻辑改写：根据规则改写 SQL，以便于优化器生成最优的执行计划

优化器：数据库优化器基于代价与规则模型生成最优的执行计划

代码生成：将执行计划翻译成可执行代码

执行计划缓存：缓存执行计划的内存，相同 SQL_ID 的 SQL 可以复用执行计划，以避免硬解析对 SQL 性能的影响



快速参数化

参数化过程是指把 SQL 语句中的常量变成变量的过程。同一条 SQL 语句在每次执行时可能会使用不同的参数，将这些参数做参数化处理，可以得到与具体参数无关的 SQL 字符串，并使用该字符串作为计划缓存的键值，用于在计划缓存中获取执行计划，从而达到参数不同的 SQL 能够共用相同的计划目的

OceanBase 使用参数化的 SQL 文本生成 SQL_ID 来唯一标识一条 SQL。SQL_ID 是使用 md5 算法对参数化后的文本计算而得，可以参考如下脚本生成对应 SQL 的 SQL_ID

快速参数化的约束条件

SQL 语句的参数化结果决定了多次执行的 SQL 是否可以共享执行计划，但是在以下场景下常量不能做参数化：

ORDER BY 后常量（例如"ORDER BY 1,2;"）

GROUP BY 后常量（例如"GROUP BY 1,2;"）

作为格式串的字符串常量（例如"DATE_FORMAT('2006-06-00', '%d'); "里面的"%d"）

函数输入参数中，影响函数结果或带有隐含信息并最终影响执行计划的常量（例如"CAST(999.88 as NUMBER(2,1))"中的"NUMBER(2,1)"，或者"SUBSTR('abcd', 1, 2)"中的"1, 2"，或者"SELECT UNIX_TIMESTAMP('2015-11-13 10:20:19.012');"里面的"2015-11-13 10:20:19.012"，指定输入时间戳同时，隐含指定了函数处理的精度值为毫秒）

优化器

OceanBase 数据库优化器融合了基于规则的路径选择方法和基于代价的路径选择方法。

基于规则的路径选择（RBO）

用于单表访问中的索引选择，有前置规则和剪枝规则两种

根据索引与查询条件的契合度来优先选择最佳索引或者剪除处于劣势的索引

基于代价的路径选择（CBO）

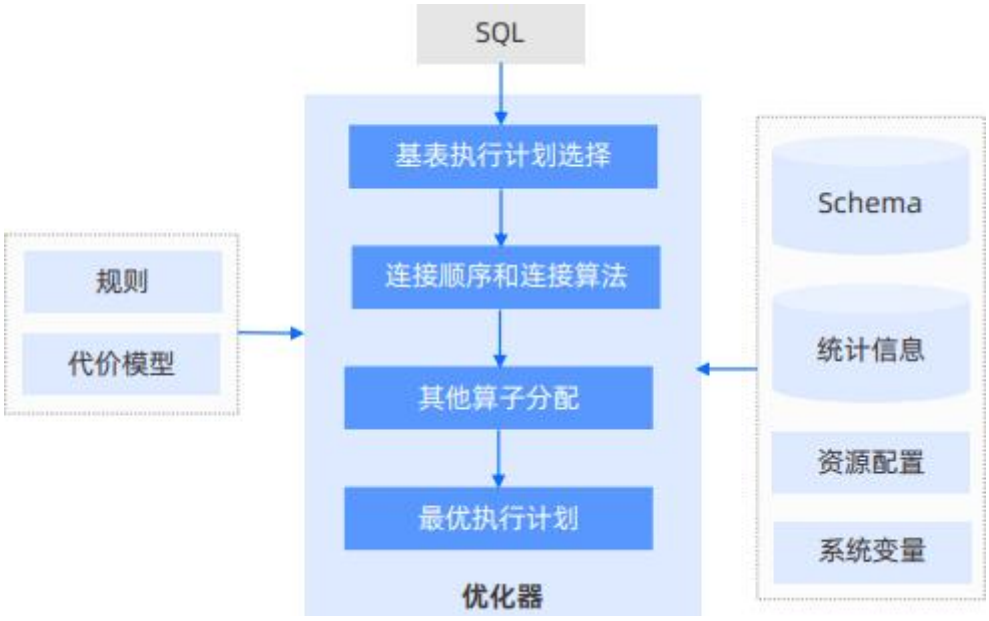
计算多个候选路径的代价，选择最低代价的路径作为执行计划

代价模型考虑了 CPU 代价和 IO 代价，以及分布式执行的代价，执行计划代价是所有算子的代价的总和

执行计划的代价主要与扫描的行数、连接算法、关联的次数有关。代价的估算基于 Schema 和统计信息，并考虑系统资源配置和系统变量的设定

OceanBase 优化器基于 System-R 框架，是自上而下的优化过程

OceanBase 路径选择优先级：RBO > CBO



执行计划

执行计划是对一条 SQL 查询语句在数据库中执行过程的描述。用户可以通过 EXPLAIN 命令查看优化器针对指定 SQL 生成的逻辑执行计划

BASIC：用于最基本的计划展示，例如算子名称、所访问的表和索引名，以及过滤信息（Filter）等。BASIC 是默认的输出模式

EXTENDED：用于最详细的计划展示，可以帮助你理解优化器的执行计划选择，从而解决 SQL 执行计划问题

执行计划的总代价是构成执行计划的每个算子的代价的总和。在 EXPLAIN 信息中，上层父算子的代价包含下层所有子算子的代价，0 号算子的代价通常是整个执行计划的总代价

EXPLAIN 解读：Outputs & filters

rowset：rowset>1 表示算子使用向量化计算

dop：dop>1 表示算子使用并行执行

use_batch：nested loop join 是否使用 batch join（一次 join 多条记录）

partitions：表扫描的分区范围

filter：过滤条件

is_index_back：是否需要回表扫描

range_key：使用的索引可以提供扫描范围的列，是一个数组

range：与 range_key 对应的每一个 key 的扫描范围，MIN~MAX 表示全部范围

range_cond：索引扫描的范围

EXPLAIN 解读：Outputs & filters

执行计划缓存

OceanBase 数据库提供了计划缓存（Plan Cache）机制来共享执行计划，相同的 SQL 执行可以复用执行计划，避免了硬解析的代价，同时也避免了执行计划的频繁变化。在某些场景下，缓存的执行计划会失效，OceanBase 需要重新生成执行计划，存在计划突变、性能回退的风险

缓存执行计划

相同的 SQL 可以共享执行计划

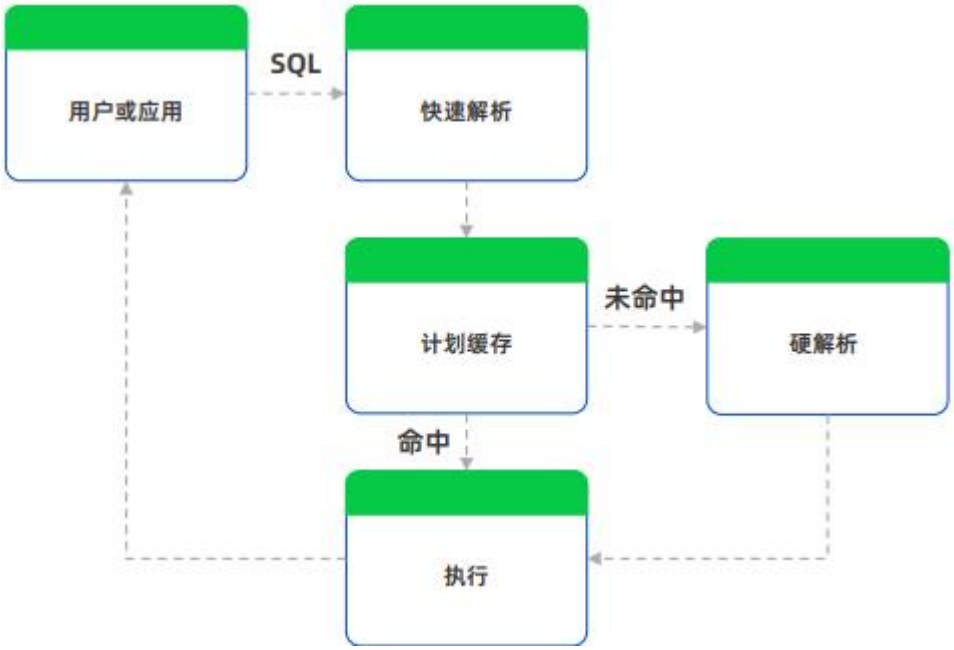
相同的 SQL 指的是参数化处理后 SQL 文本相同

缓存失效的时机

统计信息变化：比如自动、手动收集统计信息

Schema 变化：比如添加、删除索引

优化逻辑变化：比如修改优化器变量、版本升级



分布式与并行计算

分布式并行执行原理

在 OceanBase 分布式数据库中，当一个 SQL 要跨节点访问数据或使用并行计算时，SQL 的执行流程如下

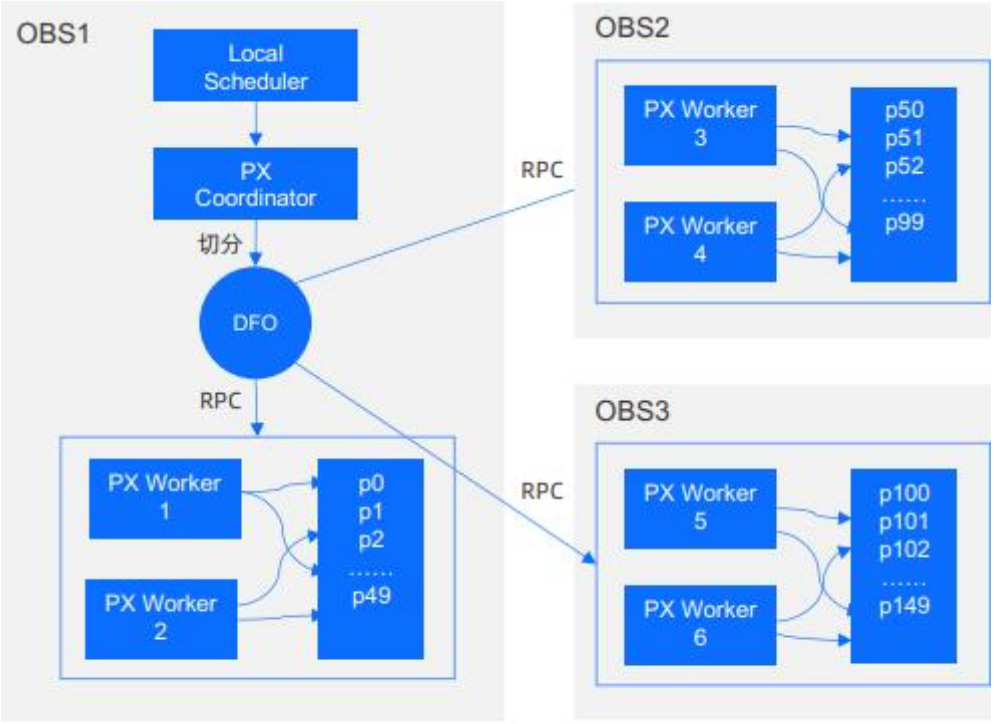
SQL 主线程（接收、解析 SQL 的线程）根据计划形态预约并行执行需要的线程资源。这些线程资源可能来自集群中的多台机器

SQL 主线程打开并行调度算子（PX COORDINATOR）

并行调度算子解析计划，将它们切分成多个操作步骤，每个操作步骤称之为一个 DFO（Data Flow Operation）

当所有操作并行执行完成后，并行调度算子会接收计算结果，并将结果吐给它的上层

算子（如 Aggregate 算子），串行完成剩余不可并行的计算（如最终的 SUM 计算）



示意图：使用6个并行线程扫描150个分区

分布式执行

当一次 SQL 执行需要跨节点访问数据时，我们称这样的 SQL 为分布式 SQL

根据 SQL 执行与所访问数据的分布位置关系，OceanBase 数据库传统上将 SQL 执行计划分为三类

plan_type=1：本地执行计划，SQL 执行与数据都在本机

plan_type=2 : 远程执行计划，SQL 执行要访问的数据存储在另一个节点上，且仅访问 1 个节点上的数据

plan_type=3 : 分布式执行计划，SQL 需要访问多个节点上的数据

在分布式架构下，OceanBase 数据库提供了两种访问数据的方式

第一种：感知数据的分布位置，生成相应类型的执行计划（ plan_type=1，2 或 3）

生成执行计划时，在其中加入数据分发与聚合的算子（EXCHANGE IN/ EXCHANGE OUT），来完成跨节点的数据收发

执行 SQL 时，将相关的子计划（subplan）发送到数据所在的节点上执行

第二种：不感知数据的分布位置，生成的执行计划统一显示为 plan_type=1

生成执行计划时，使用 DAS（Data Access Service）来封装对数据的访问

执行 SQL 时，SQL 执行层直接调用 DAS 接口对数据进行读写，不关心数据的分布位置

在 OceanBase V4 版本，数据库优化器基于代价规则自动选择使用哪种数据访问方式来生成执行计划

DAS

DAS（Data Access Service）是 OceanBase 数据库的分布式数据访问服务，DAS 提供跟存储的远程交互和多分区操作的能力

DAS 只提供跟存储的远程交互能力，并不感知 SQL 的具体语义，只是对相应的接口做一层封装，并提供这些接口多分区操作的能力

在执行计划中，TABLE SCAN 以及 UPDATE、INSERT、DELETE 算子前有 DISTRIBUTED 表示使用了 DAS

并行执行

并行执行是指将一个 SQL 查询任务分成多个子任务，并允许这些子任务在多个处理器上同时运行，以提高整个查询任务的执行效率。在现代计算机系统中，多核处理器、多线程和高速网络连接广泛应用，这使得并行执行成为了一种可行的高效率查询技术

并行执行通过充分利用多个 CPU 和 IO 资源，达到降低 SQL 执行时间的目的

并行执行的适用场景

并行执行能够极大降低计算密集型大查询的响应时间，被广泛应用在离线数据仓库、实时报表、在线大数据分析等业务场景，同时还应用在批量导数，快速构建索引表等领域

以下场景会从并行执行中获益

大表扫描，大表连接，大数据量排序，聚合

大表 DDL 操作，如修改主键、改变列类型，建索引等

从已有大数据建表（Create Table As Select）

批量插入、删除、更新

并行执行的系统需满足以下特征

多处理器系统（SMPs）、集群

IO 带宽足够

内存富余（可用于处理内存密集型操作，如排序、建 hash 表等）

系统负载不高，或有峰谷特征（如系统负载一般在 30%以下）

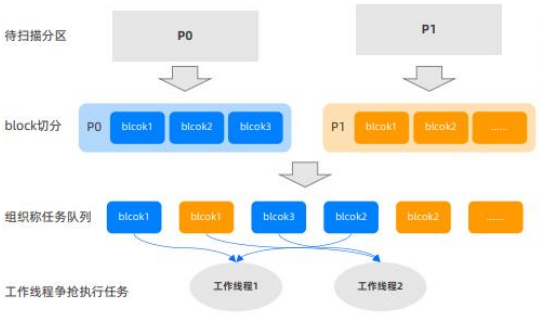
并行的粒度（Granule）

并行数据扫描的基本工作单元被称为 Granule，每个 Granule 对应一个分区内的扫描任务。基于 Granule 的粒度特征，可以分为以下两类

Partition Granule：以分区为并行执行的单位，也就是分区间并行。这里的分区既可以是主表分区，也可以是索引分区

Block Granule：以连续的宏块作为并行执行的单位，也就是分区内并行

在给定并行度的情况下，为了保持扫描任务的均衡，优化器会自动选择将数据划分为 Partition Granule 或 Block Granule，确保扫描任务的均衡




```
Query Plan
==========================================================
|ID|OPERATOR            |NAME    |EST.ROWS|EST.TIME(us)|
----------------------------------------------------------
|0 |PX COORDINATOR      |        |1       |212         |
|1 |└EXCHANGE OUT DISTR |:EX10000|1       |212         |
|2 |  PX BLOCK ITERATOR |        |1       |211         |
|3 |    └TABLE FULL SCAN|L       |1       |211         |
==========================================================
Outputs & filters:
-------------------------------------
  0 - output([INTERNAL_FUNCTION(L.C1, L.C2, L.C3)]), filter(nil), rowset=16
  1 - output([INTERNAL_FUNCTION(L.C1, L.C2, L.C3)]), filter(nil), rowset=16
      dop=2
  2 - output([L.C1], [L.C2], [L.C3]), filter(nil), rowset=16
  3 - output([L.C1], [L.C2], [L.C3]), filter([L.C2 > 100]), rowset=16
      access([L.C1], [L.C2], [L.C3]), partitions(p[0-1])
      is_index_back=false, is_global_index=false, filter_before_indexback[false],
      range_key([L.C1]), range(MIN ; MAX)always true
```

并行度（DOP）

并行度 DOP（Degree Of Parallelism）指定了并行执行的线程数量

OceanBase 多种方式对查询开启并行执行。在查询支持并行执行的前提下，多种并行开启方式优先级由高到低的顺序如下

表级 DOP

查询级 DOP

系统变量级 DOP

Schema 级 DOP

OceanBase 会将 DOP 按照分区个数的比例分配到多个 OBServer 节点上。例如，DOP 为 6，要访问 120 个分区，其中 server1 上有 60 个分区， server2 上有 40 个分区，server3 上有 20 个分区，那么会分 3 个线程给 server1，分 2 个线程给 server2，分 1 个线程给 server3，达到平均每个线程可以处理 20 个分区的效果

并行度的设置方法

在 OceanBase V4 版本， SQL 查询的并行度由两种方式来控制：手动并行和自动并行（Auto DOP）

手动并行

手动并行是指在 SQL 查询中使用 Optimizer Hint 等方式来设置并行度。其控制方式如下：

- 不指定并行度：

```
SELECT ... FROM table ...;
```

  - 跨节点的多分区查询自动并行执行，每个节点都有一个执行线程。

- 指定并行度：

```
SELECT /*+ parallel(n) */ ... FROM table ...;
```

  - n：手动指定并行度为 n。

- 禁止使用并行：

```
SELECT /*+ NO_USE_PX */ ... FROM table ...;
```

自动并行（Auto DOP）

自动并行是指由优化器根据查询的实际情况，自动选择是否开启并行执行，并确定并行度。其控制方式如下：

- 系统变量控制：**parallel_degree_policy**，默认值为 manual。

```
SET [GLOBAL|SESSION] parallel_degree_policy = auto;
SET [GLOBAL|SESSION] parallel_degree_policy = manual;
```

  - auto 开启自动并行；manual 关闭自动并行。

- Optimizer Hint 控制：

```
SELECT /*+ parallel(auto|manual|n) */ ...;
```

  - auto 开启自动并行；manual 关闭自动并行。
  - n：手动指定并行度为 n。

手动 DOP 设置方法

在 OceanBase V4 版本中，我们可以通过以下方式来手动设置 SQL 语句或表扫描的并行度

通过 PARALLEL Hint 指定 SQL 语句的并行度（SELECT /*+ parallel(3) */ * FROM t1, t2 WHERE t1.c2 = t2.c1;）

通过表属性 parallel 设置表扫描的默认并行度（ALTER INDEX tb1_ix1 PARALLEL 3）

通过会话的并行属性，设置会话中查询的默认并行度

mysql 模式：set _force_parallel_query_dop = n

oracle 模式

ALTER SESSION FORCE PARALLEL QUERY PARALLEL n；

ALTER SESSION DISABLE PARALLEL QUERY；

AUTO DOP 策略

Auto DOP 策略下，优化器如何判断是否使用并行执行，如何确定并行度以及每个查询的最大并行读，这些都需要用户进行相应的策略配置。

是否使用并行执行？

在执行计划选择时，OceanBase 数据库优化器会比较顺序执行与并行执行的代价，如果并行执行的耗时更低，才可能选择并行执行

系统变量 parallel_min_scan_time_threshold 控制基表扫描开启并行的最低执行时间

当优化器估算的基表扫描执行时间大于设定值时，会对基表扫描开启并行，并利用这个设定值计算一个合适的并行度

变量默认值设置为 1000 毫秒，即 1 秒

如何确定查询的并行度？

首先确定基表算子的 DOP。通过逐步增大 DOP 来评估并行执行的耗时，并考虑并行执行框架的开销，确定一个最合适的 DOP

其他算子，比如 Join 等，继承与之关联的算子的 DOP

如何限制最大的并行度？

系统变量 parallel_degree_limit 控制 Auto DOP 策略下所有查询语句的最大并行度

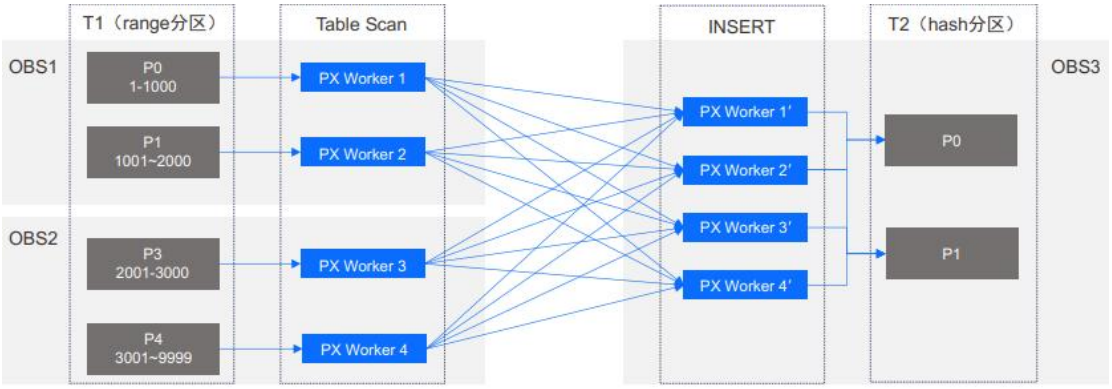
默认值为 0， 表示对自动获取的 DOP 不进行限制

并行 DML

并行 DML 通过使用并行执行机制来提高对大型数据库的表和索引执行插入、更新、删除等操作，以提高执行效率

OceanBase 数据库 V4 版本支持在 INSERT INTO SELECT、UPDATE、DELETE 等语句使用并行执行能力

开启并行 DML 时，查询部分总是自动开启并行，DML 的并行度和查询部分的并行度一致。读出的数据会根据待更新表的分区位置做重分布，然后由多个线程并行 DML，每个线程负责若干个分区



开启并行 DML

OceanBase 数据库支持在 SQL 语句或会话中显式启用并行 DML

通过 Hint 在 SQL 语句中启用或关闭并行 DML（INSERT /*+ ENABLE_PARALLEL_DML PARALLEL(4) */ INTO t2 SELECT * FROM T1）

通过会话的并行属性，设置会话中的 DML 是否使用并行（SET _force_parallel_dml_dop = n;）

注意：Parallel DML 执行只有在 DOP>1 且 ENABLE PARALLEL DML 时才会生效，使用 Auto DOP 时自动开启 Parallel DML

并行执行工作线程

一个并行查询会使用两类线程：1 个主线程，若干个并行工作线程。其中主线程和普通的 TP 查询使用的线程没有任何区别，来自普通工作线程池，并行工作线程来自专用 PX 线程池

租户在每个 OBServer 节点上可申请的 PX 线程总数 N 由系统配置项 parallel_servers_target 控制，其默认值为租户资源单元的 MIN_CPU 的 10 倍。比如，租户的 MIN_CPU 是 5，则租户在任一 OBServer 节点最多可分配 50 个 PX 线程

px_workers_per_cpu_quota，默认值 10，租户级配置项，代表每个 CPU 上可以分配的并行执行线程数

parallel_servers_target，默认值 0，租户级变量，代表租户在每个节点上可申请的并行执行线程数量

查询每次从线程池申请线程时，单次申请的线程数量不会超过 N，如果超过，则最多只分配 N 个线程。比如，查询从 3 台 OBServer 上各申请 30 个 PX 线程，如果租户的 MIN_CPU 是 2，则查询实际可以申请的 PX 线程数是 20*3=60

并行执行开始执行前，需要预约 PX 线程资源，预约成功后才能投入执行。如果预约不成功，查询会被放入队列等待

事务控制原理

事务的概念与操作

事务的概念

数据库事务（Transaction）是指作为单个逻辑工作单元执行的一系列操作，用于确保数据操作的正确性和完整性，这些操作要么全部执行，要么全部不执行。数据库事务有两个作用：

为数据库操作序列提供一个从失败中恢复到正常状态的方法，同时提供了数据库即使在异常状态下仍能保持一致性的方法

为数据库的多个并发访问提供隔离的方法，避免多个并发操作导致数据库进入一个不一致的状态

事务的 ACID 特性

数据库事务具有 4 个特性：原子性、一致性、隔离性、持久性，这四个属性通常称为 ACID 特性。

原子性（ Atomicity）

事务中的所有操作要么全部成功，要么全部失败

事务的原子性通过数据库日志保证

一致性（ Consistency）

数据库在事务执行前后保持一致

一致性通过数据完整性约束和事务回滚机制实现

隔离性（Isolation）

事务在并发执行时相互隔离，互不干扰

隔离性通过锁、MVCC 和事务隔离级别实现

持久性（Durability

事务一旦提交，其对数据库的更改是永久性的

持久性通过 redo log 和 WAL 机制保证

在转账交易中，事务的ACID特性体现在如下方面：

- 原子性：账户A的扣款和账户B的入账必须全部成功或者全部回滚。
- 一致性：转账交易前后，账户A和账户B的余额均处于逻辑一致。
- 隔离性：在转账交易的事务没有提交前：通过锁机制，别人不能同时对账户A、B执行修改操作；通过隔离级别与MVCC机制，别人无法看到账户A、B未提交的余额变更。
- 持久性：转账交易完成时，数据库通过WAL机制保证事务日志落盘，即使机器故障重启，数据库依然可以通过日志恢复出事务提交后的数据状态。

事务的结构

一个数据库事务包含一条或者多条 DML 语句，事务有明确的起始点及结束点。数据库对事务的控制包括开启事务、提交事务和回滚事务。OceanBase 数据库提供了与 MySQL 和

Oracle 兼容的事务控制语句

开启事务：当事务开启时，OceanBase 数据库为事务分配一个事务 ID，用于唯一的标识一个事务

手动开启：执行 START TRANSACTION（仅 MySQL 模式） 或 BEGIN 命令

自动开启：关闭自动提交后（autocommit = 0），执行 DML 语句（除 SELECT 外）、DCL 语句或 DDL 语句时，会默认开启一个事务

语句执行：在语句执行过程中，OceanBase 数据库在语句访问到的每个分区上创建一个事务上下文，用于记录语句执行过程中的数据快照版本号以及语句对该分区所做的修改

在提交事务之前，事务的修改只对当前事务可见，对其他数据库事务是不可见的

在提交事务之前，事务的修改没有持久化，所以不是最终的结果，可以用 ROLLBACK 语句回滚这些修改

结束事务：事务结束时收集事务执行过程中修改过的所有分区，根据不同的场景对这些分区发起提交事务或者回滚事务。以下场景会触发事务的提交或者回滚

用户显式的发起 Commit 或者 Rollback。用户发起 Commit 时，OceanBase 数据库会将事务所做的修改持久化到 clog 文件中

用户执行 DDL 操作。包括 CREATE、DROP、RENAME、或者 ALTER。当用户在事务中发起这些 DDL 操作，OceanBase 数据库会隐式的发起一个 Commit 请求，后续的语句会开启一个新的事务

客户端断开连接。当客户端在事务执行过程中断开连接，OceanBase 数据库会隐式的发起 Rollback 请求，将事务回滚

事务的自动提交（AutoCommit）

提交一个事务会让事务的修改持久化生效，清除保存点并释放事务所持有的所有锁。租户变量 autocommit 控制是否自动提交事务，当 autocommit = 1 时，每条语句执行结束后，OceanBase 数据库将会自动执行 COMMIT，一条语句就是一个事务。

可以通过修改 autocommit 变量来控制事务是否自动提交

通过 SET autocommit 设置变量时，当前会话立即生效，断开链接之后被设置的变量会失效

通过 SET GLOBAL autocommit 设置租户级别的变量时，需要断开链接才生效

注意：执行 START TRANSACTION 或 BEGIN 开启事务后， autocommit = 1 不再起作用，需要执行 COMMIT 或 ROLLBACK 来结束事务

事务的保存点（SavePoint）

SAVEPOINT 语句可以在事务执行的过程中标记一个"保存点"，应用出错时可以选择将事务回滚到这个保存点，从而达到部分回滚事务的效果

标记保存点：执行 "SAVEPOINT point_name" 命令

保存点是可选的，一个事务过程中可以标记多个保存点

回滚到保存点：执行 "ROLLBACK TO SAVEPOINT point_name" 命令

回滚到某个保存点后，事务不会结束

保存点之前的修改被保留；保存点与 ROLLBACKSAVEPOINT 语句之间的修改被丢弃，之间的其他保留点被清除

释放保存点：执行 "RELEASE SAVEPOINT point_name" 命令

当 RELEASE SAVEPOINT 语句执行成功后，不能再回滚到指定的保存点以及 RELEASE SAVEPOINT 语句与指定的保存点之间的任何其他保存点

■ **示例：事务部分回滚**

```
BEGIN;
UPDATE T1 ... ;          # 被提交
SAVEPOINT point1;
INSERT INTO T2 ...;      # 被提交
SAVEPOINT point2;
UPDATE T1 ... ;          # 被回滚
INSERT INTO T2 ...;      # 被回滚
ROLLBACK TO SAVEPOINT point2;
UPDATE T1 ... ;          # 被提交
INSERT INTO T2 ...;      # 被提交
COMMIT;
```

部分回滚

语句级原子性

OceanBase 数据库支持语句级的原子性。语句级的原子性指的是一条语句的操作要么都成功，要么都失败，不会存在部分成功部分失败的情况，即语句执行失败导致的回滚是语句级的，不是事务级的

语句级原子性的特点

当一条语句执行过程中报错，那么该语句执行的操作都会被回滚，不会影响事务中之前执行的语句所做的修改

语句级回滚的效果等价于语句没有被执行过，即语句执行过程中涉及的全局索引、触发器、行锁等均属于语句的操作，语句回滚需要将这些操作都回滚到语句开启之前的状态

■ **示例：语句级回滚**

```
BEGIN;                                  //开启事务
UPDATE T1 SET C1=1 WHERE C2=100;        //执行成功
INSERT INTO T1 SELECT * FROM T2;        //执行失败，回滚
COMMIT;                                 //提交事务
```

事务执行结果：

- UPDATE 执行成功，其更新结果被提交。
- INSERT 执行失败，其所有 INSERT 的记录都被回滚

事务日志（Redo Log）

事务日志（Redo log）是 OceanBase 数据库用于宕机恢复以及维护多副本数据一致性的关键组件。事务日志记录了事务对于数据的全部修改历史

Write-Ahead Logging

与大多数主流数据库相同，OceanBase 数据库遵循 WAL（write-ahead logging）原则，在事务提交前将 Redo 日志持久化，保证事务的原子性和持久性（ ACID 中的 "A" 和 "D"）。如果 observer 进程退出或所在的服务器宕机，重启 OBServer 节点会扫描并回放本地的 Redo 日志用于恢复数据。宕机时未持久化的数据会随着 Redo 日志的回放而重新产生

多副本日志同步

OceanBase 数据库采用 Multi-Paxos 协议在多个副本间同步 Redo 日志。对于事务层来说，一次 Redo 日志的写入只有同步到多数派副本上时才能认为成功。而事务的提交需要所有 Redo 日志都成功写入。最终，所有副本都会收到相同的一段 Redo 日志并回放出数据。这就保证了一个成功提交的事务的修改最终会在所有副本上生效并保持一致。Redo 日志在多个副本上的持久化使得 OceanBase 数据库可以提供更强的容灾能力

租户级日志流

OceanBase 数据库采用了租户级别的日志流，每个分区的所有日志要求在逻辑上连续有序。机器上同一租户的所有 Tablet 产生的日志会写入到同一个日志流，日志流中的日志存在全序关系

只读事务

只读事务是指仅包含只读语句的事务。只读事务不能存在对数据的修改，不需要创建事务上下文，在 COMMIT 或 ROLLBACK 时也无需执行事务提交或回滚的操作

只读不开事务：在没有显式地开启事务的情况下，执行 SELECT 语句（SELECT FOR UPDATE 除外）不会开启事务

```
SELECT * FROM t1;                  //只读不开事务
SELECT * FROM t2;                  //只读不开事务
INSERT INTO t1 VALUES(1);          //自动开启事务
```

```
INSERT INTO t1 VALUES(1);          //自动开启事务
SELECT * FROM t1;                  //事务中
SELECT * FROM t2;                  //事务中
```

只读事务：在开启事务时指定事务为只读模式，不允许在事务中执行数据更新操作，也不允许执行 SELECT FOR UPDATE。只读事务不创建事务上下文。设置只读事务有以下几种方式

使用 START TRANSACTION READ ONLY 开启只读事务

在开启事务（START TRANSACTION 或 BEGIN）前指定事务模式为只读：SET TRANSACTION READ ONLY

在开启事务（START TRANSACTION 或 BEGIN）前设置会话变量 tx_read_only =1

分布式事务

一个事务可以同时更改多个表、分区的数据。在集中式数据库中，所有的表集中在一台服务器上，事务更多是单机事务；在分布式数据库中，一个事务中修改的表、分区可能分布在不同的日志流之中时，事务的执行与提交需要协调多个日志流共同完成，事务更多是分布式事务。

按照事务 Session 位置和事务涉及的日志流 leader 数量，可以将 OceanBase 数据库事务分为以下两种类型：

单日志流事务

事务涉及的日志流数量只有一个，且日志流的 Leader 和事务 Session 在同一个 Server 上

分布式事务：以下两种场景的事务均称为分布式事务

事务涉及的日志流数量大于一个

事务涉及的日志流只有一个，但日志流的 Leader 和事务 Session 位置不在同一个 Server 上

传统的两阶段提交

传统的两阶段提交（Two-Phase Commit，2PC）是一个强一致、中心化的原子提交协议。中心化指的是事务的状态由协调者（Coordinator）来记录和推进，协调者维护参与者列表；强一致性指的是需要所有参与者（Participant）均要执行成功才算成功，否则回滚

第一阶段：PREPARE 阶段

协调者通知所有参与者准备（Prepare）

参与者收到 Prepare 请求后，决定是否可以提交，如果可以则返回 OK，否则返回失败

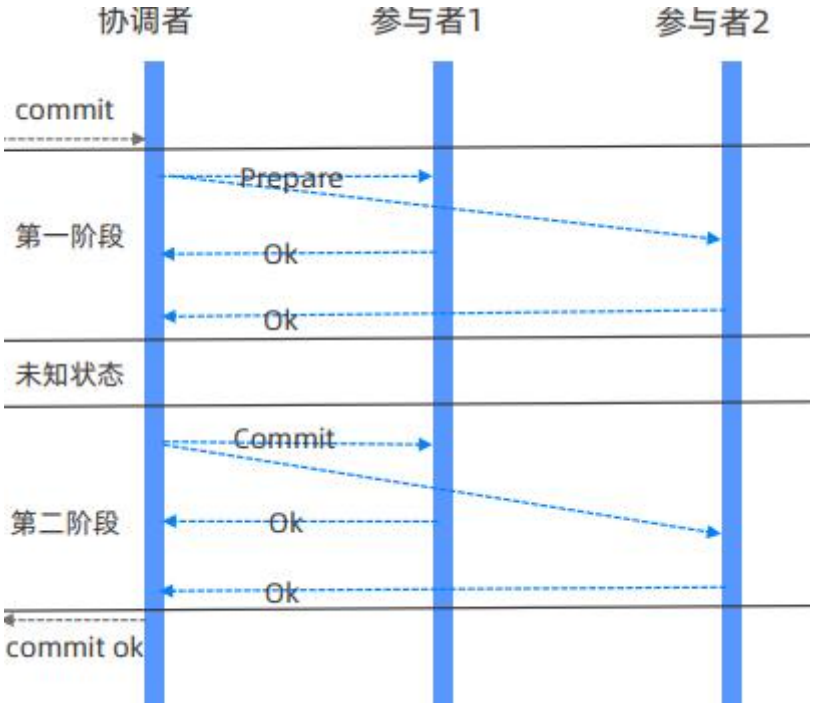
协调者受到所有参与者的 Prepare 应答后，推动事务提交到第二阶段。如果所有参与者都能 Prepare 成功，则进入 COMMIT 阶段，否则进入回滚阶段

第二阶段：COMMIT 阶段

协调者则通知所有参与者执行 Commit 或 Rollback 操作

参与者收到 Commit 或 Rollback 请求后，释放资源解行锁，告知协调者 Commit 或 Rollback 成功

当所有参与者均 Commit 或 Rollback 成功后，参与者返回给应用 Commit 或 Rollback 成功



OceanBase 的两阶段提交

为了提升分布式场景下的事务提交性能，OceanBase 优化了两阶段提交协议，协调者（Scheduler）无状态，所有参与者（Participant）均记录参与者列表与事务状态日志，在第一阶段执行成果后即可对应用返回 Commit 成功

第一阶段：PREPARE 阶段

协调者通知所有参与者准备（Prepare）

参与者收到 Prepare 请求后，决定是否可以提交，如果可以则返回 OK，否则返回失败

如果所有参与者都能 Prepare 成功，协调者立刻返回给应用 Commit 成功，应用不需要等待第二阶段的事务提交

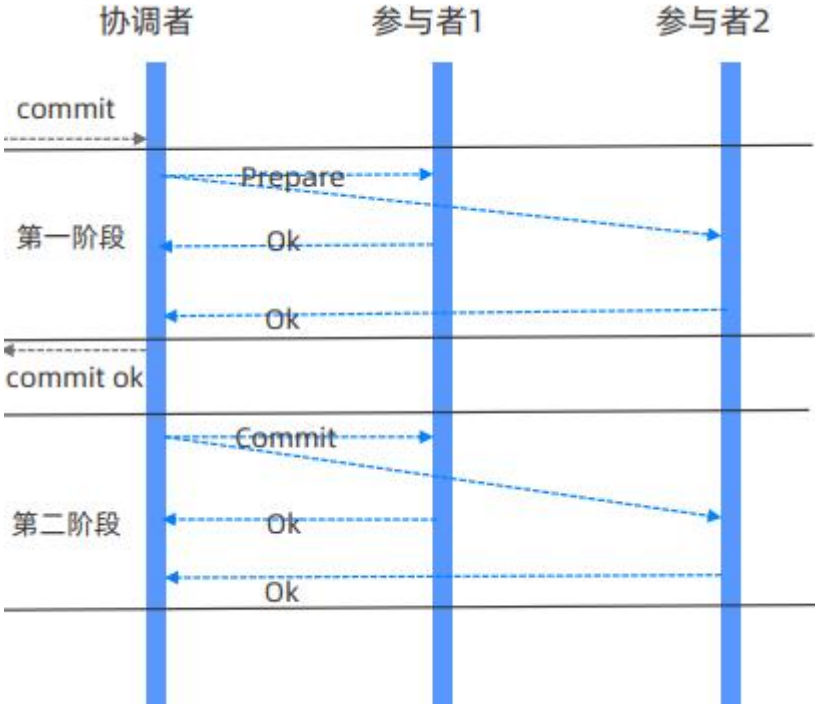
如果有参与者不能 Prepare 成功，协调者会尝试重试或者执行回滚

第二阶段：COMMIT 阶段

协调者则通知所有参与者执行 Commit 或 Rollback 操作

参与者收到 Commit 或 Rollback 请求后，释放资源解行锁，告知协调者 Commit 或 Rollback 成功

在 OceanBase 数据库中，一旦第一阶段执行成功，事务一定会被提交



两种 2PC 协议的对比

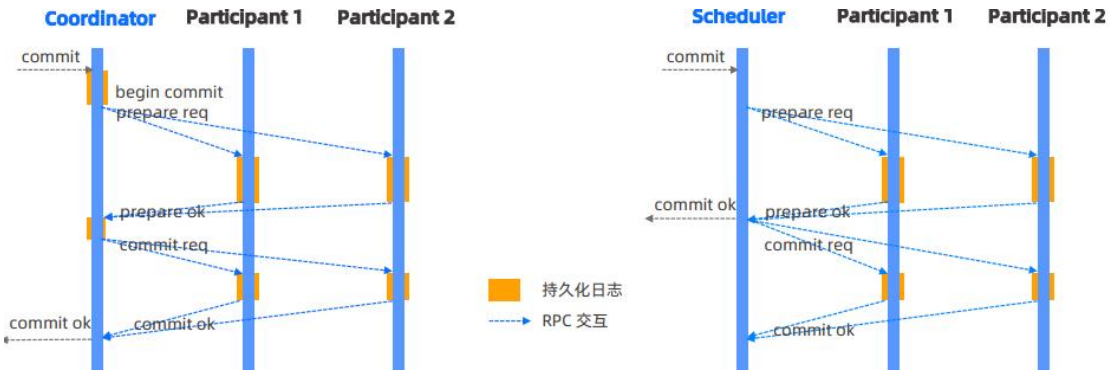
OceanBase 数据库采用协调者无状态设计，协调者不再维护分布式事务的状态，而是在宕机恢复时，通过所有参与者的局部状态动态构造分布式事务的全局状态。这种方式避免了协调者写日志，大幅降低了分布式架构下两阶段提交的延迟

经典二阶段提交

用户感知的 Commit 延迟：4 次写日志 + 2 次 RPC 交互

OceanBase 二阶段提交

用户感知的 Commit 延迟：1 次写日志 + 1 次 RPC 交互



事务超时控制

在 OceanBase 数据库中，事务执行过程中的超时控制主要有语句超时、事务超时和事务空闲超时

语句超时：系统变量 ob_query_timeout 控制着语句执行时间的上限，语句执行时间超过此值会给应用返回语句超时的错误，错误码为 -6212，并回滚语句，通常该值默认为 10s

事务超时：系统变量 ob_trx_timeout 控制着事务超时时间，事务执行时间超过此值会给应用返回事务超时的错误，错误码为 -6210，此时需要应用发起 ROLLBACK 语句回滚该事务

事务空闲超时：系统变量 ob_trx_idle_timeout 表示 Session 上一个事务处于的 IDLE 状态的最长时间，即长时间没有 DML 语句或结束该事务，则超过该时间值后，事务会自动回滚，再执行 DML 语句会给应用返回错误码 -6224，应用需要发起 ROLLBACK 语句清理 Session 状态

活跃事务的状态

一个活跃的事务在执行过程中会有不同的阶段或者状态，了解事务状态对管理事务非常有帮助。在 OceanBase 数据库中，事务的状态如以下表格所述

INIT 表示事务处于活跃状态，所有修改对其他事务不可见。

REDO COMPLETE 表示事务已经将所有事务修改以 Redo 日志形式持久化。

PREPARE 表示事务已经开始提交，目前处于 PREPARE 状态，读取该事务的修改可能会被卡住（取决于版本号）

PRECOMMIT 表示事务即将提交，正在同步事务的提交版本号到所有涉及的参与者

COMMIT 表示事务已经开始提交，且目前处于 COMMIT 状态，其他事务可以看到该事务的修改（取决于版本号）

ABORT 表示事务已经回滚，处于 ABORT 状态，其他事务不能看到该事务的修改

CLEAR 表示事务已经提交或回滚结束，处于 CLEAR 状态

事务监控

系统视图 GV$OB_TRANSACTION_SCHEDULERS 和 GV$OB_TRANSACTION_PARTICIPANTS 分别展示活跃事务的协调者与参与者的信息。可以在 SYS 租户和用户租户使用这两个系统视图

来监控事务的状态

GV$OB_TRANSACTION_SCHEDULERS：事务协调者视图

GV$OB_TRANSACTION_ PARTICIPANTS ：事务参与者视图

会话监控

系统视图 GV$OB_PROCESSLIST 展示 OBServer 节点上所有会话的状态和统计。可以在 SYS 租户和用户租户使用该系统视图来监控会话的状态

查询租户活跃会话数量和最长执行时间

SELECT SVR_IP, TENANT_NAME,TENANT_ID,CASEWHEN CNT IS NULL THEN 0ELSE CNTEND AS ACTIVE_SESSIONS,MAX_TOTAL_TIME,MAX_TIMEFROM (SELECT T1.SVR_IP, T0.TENANT_NAME,T0.TENANT_ID, T1.CNT,T1.MAX_TOTAL_TIME,T1.MAX_TIMEFROM DBA_OB_TENANTS T0LEFT JOIN (SELECT SVR_IP, SVR_PORT,TENANT AS TENANT_NAME,Count(`STATE`= 'ACTIVE'OR NULL) AS CNT,Max(TOTAL_TIME) AS MAX_TOTAL_TIME,Max(TIME) AS MAX_TIMEFROM GV$OB_PROCESSLISTGROUP BY TENANT, SVR_IP,SVR_PORT) T1ON T0.TENANT_NAME = T1.TENANT_NAMEWHERE T0.TENANT_TYPE <> 'META') T2ORDER BY SVR_IP, TENANT_ID;

事务的并发控制原理

多版本读一致性

OceanBase 数据库存储数据的多个版本，通过多版本并发控制（Multi-Version Concurrency Control，MVCC）实现多版本一致性读，避免读写冲突。多版本一致性是通过读版本和数据版本来保证的，查询语句通过读取版本号，返回小于读取版本号的所有提交数据，来定义多版本一致性

多版本读一致性的特点

不能读到非本事务的未提交事务, 否则若对应事务回滚，就会产生脏读（dirty read）

要读取小于读取版本号的所有提交数据，来保证一个用户可理解的一致性点

在满足未提交事务与事务一致性点的前提下，依旧要保证读写不互斥

示例：读版本号为 90 的多版本读一致性

数据行	多版本数据	数据版本号	读一致性
1	x	100	不满足
	y	80	满足
	z	70	满足
2	x	未决定	不满足
	y	100	不满足
	z	80	满足
3	x	未决定	不满足
	z	80	满足

两种级别的一致性读

OceanBase 数据库提供了两种读一致性级别（Consistency Level）：STRONG 和 WEAK

强一致性读（STRONG）

强一致性读是 OceanBase 数据库默认的数据读取方式

强一致性读要求读取满足多版本读一致性的最新的数据版本，即所有数据版本号小于读版本号的多版本数据中的最大的已提交数据版本

强一致性读要求 SQL 必须转发到主副本（Leader）上执行，用以保证获取到实时最新数据

OceanBase 的 DML 语句以及 SELECT FOR UPDATE 始终是强一致性的，必须转发到主副本（Leader）上执行

弱一致性读（WEAK）

弱一致性读要求读取满足多版本读一致性的数据，但不要求是数据的最新版本

弱一致性读的 SELECT 请求可以路由到从副本（Follower）或备库副本上执行

复制表的强一致性读

复制表的强一致性读可以在从副本上执行，因为复制表的主从副本是强同步的

复制表的 DML 语句以及 SELECT FOR UPDATE 依然只能在主副本（Leader）上执行

一致性级别的指定方式

OceanBase 数据库支持两种一致性基本的指定方式，一种是通过系统变量来指定会话的默认一致性读级别，一种是通过 Hint 来指定 SELECT 语句的一致性读级别

通过系统变量 ob_read_consistency 指定会话的一致性读级别

设置 Session 变量，影响当前 Session(SET ob_read_consistency = WEAK;)

设置 Global 变量，影响之后新建的所有 Session(SET GLOBAL ob_read_consistency = STRONG;)

通过 Hint 指定 SQL 的一致性读级别

指定 WEAK Consistency，优先级高于 ob_read_consistency(SELECT /*+READ_CONSISTENCY(WEAK) */ * FROM t1;)

指定 STRONG Consistency，优先级高于 ob_read_consistency(SELECT /*+READ_CONSISTENCY(STRONG) */ * FROM t1;)

注意：V4 版本中，OceanBase 支持在事务中的查询指定弱一致性读 hint，但是会转发到 leader 节点执行，保持强读语义。而如果备租户的查询中使用弱一致性 hint，查询会被转发到备租户的 follower 节点执行

弱一致性读时间戳

在从副本或备库上执行弱一致性读时，OceanBase 通过弱一致性读时间戳与分区的最大安全可读点位的对比，确保弱一致性读返回的数据满足事务一致性点, 不会出现返回未提交事务和一半事务的情况

弱一致性读时间戳：弱读执行之前，需要确定一个读快照，该快照确定了弱一致性读可以读取的数据版本的范围

租户配置项 max_stale_time_for_weak_consistency 设置弱一致性读允许读到多旧的数据，默认值为 5s

系统变量 ob_max_read_stale_time 设置当前 Session 中弱一致性读允许读到多旧的数据，默认值为 -1，表示使用租户配置项的值

最大安全可读位点：每个日志流副本实时维护一个最大安全可读的位点，凡是读快照大于该位点的读请求，均不能读该副本

示例：弱一致性读与最大安全可读点位的比较

当前时间为 10:30:10，分区所在的日志流由三个副本存，其最大安全可读位点（同步位点）分别是：

副本 1（Leader）：10:30:10

副本 2（Follower）：10:30:06

副本 3（Follower）：10:30:07

所有副本中最小的可读时间戳（10:30:06）就是当前弱读请求的最大安全可读位点，也就是弱读能读到“10:30:06”以及之前的数据。

如果 max_stale_time_for_weak_consistency 设置为 5s，那么弱一致性读时间戳为 10:30:05，所有副本均满足弱读的一致性

如果 max_stale_time_for_weak_consistency 设置为 3s，那么弱一致性读时间戳为 10:30:07，副本 2 不满足弱读的一致性

事务的隔离级别

并发的事务可能会读、写相同的记录，如果不对事务进行并发控制，可能会出现事务之间相互干扰的现象，导致数据不一致或者其他异常。数据库使用事务的隔离级别（Isolation Level）来控制多个事务的并发操作，以保证数据库事务的 ACID 属性中的 I

ANSI/ISO SQL 标准基于事务执行过程中必须避免的异常情况，定义了四种隔离级别，按照其严格程度从小到大排列如下

读未提交（Read Uncommitted）

含义：事务可以读取其他事务尚未提交的更改。这意味着可能会读取到不一致的数据（脏读）

允许的异常：脏读（Dirty Read）、不可重复读（Non-repeatable Read）、幻读（Phantom Read）

读已提交（Read Committed）

含义：事务仅能读取其他事务已提交的更改，这避免了脏读问题

允许的异常：不可重复读（Non-repeatable Read）、幻读（Phantom Read）

可重复读（Repeatable Read）

含义：事务内多次重复读取同一批数据时，数据的版本是一致的，即可重复读

允许的异常：幻读（Phantom Read）

可串行化（Serializable）

含义：这是最严格的隔离级别，相当于所有事务按顺序依次执行

允许的异常：无（所有并发异常均被避免）

注意：幻读是指一个事务在多次读取一组数据时，因为其他事务插入或删除了数据，导致该事务多次读取时的结果集发生变化

OceanBase 的隔离级别

OceanBase 数据库支持读已提交、可串行化以及可重复读（仅 MySQL 模式）三种隔离级别，不支持读未提交。OceanBase 的可重复读实际使用的是可串行化（基于兼容的考虑，Oracle 模式的租户不支持可重复读隔离级别）

读已提交

读已提交是 OceanBase 数据库默认的隔离级别

每条 SELECT 语句在执行之前获取当前数据库的最新快照，执行时仅能够读到在此之前已经提交事务的数据

UPDATE、DELETE、SELECT FOR UPDATE 等更新操作在查找目标行时，其行为与 SELECT 相同，即只能找到在语句开始前已经提交的行版本

可重复读或可串行化

事务的第一条语句会获取一个事务快照版本，后续 SELECT 语句都会基于事务快照读取数据，只能读到在事务快照之前所有已经提交事务的数据，而不会读到在事务执行过程中新提交或被并发事务修改的数据

UPDATE、DELETE、SELECT FOR UPDATE 等更新操作，它们在搜索目标行时的行为与 SELECT 相同，即只能找到在获取事务快照前已经提交的行版本。如果目标行的最新已提交数据版本大于事务的快照版本，意味着事务执行过程中有别的事务更新了相同的行，不满足可串行化的目的，则本事务会执行回滚

OceanBase 的可重复读或可串行化与 Oracle 数据库的可串行化类似，都不是严格意义上的 Serializable

锁机制：表级锁

OceanBase V4 实现了表锁和行锁两级锁。通过表锁与行锁，OceanBase 从而实现对 DML、DDL 的并发控制，确保数据的一致性和完整性。

表级锁：表锁的对象是表或分区，表锁有不同的模式，不同模式间的表锁请求根据相容矩阵相容或互斥

ROW SHARE（RS）：行共享锁，禁止其他用户对表加 X 锁

ROW EXCLUSIVE（RX）：行排他锁，禁止其他用户对表加 S、SRX、X 锁

SHARE（S）：表共享锁，其他用户可以对表加 S 锁，禁止其他用户对表进行写操作

SHARE ROW EXCLUSIVE(SRX)：S+RX，禁止其他人使用 S 锁，禁止其他人更新数据

Exclusive(X)：表排他锁，只允许其他用户对表进行查询（SELECT），禁止进行其他任何操作

Request	Lock Mode				
	RS	RX	S	SRX	X
RS	✓	✓	✓	✓	✗
RX	✓	✓	✗	✗	✗
S	✓	✗	✓	✗	✗
SRX	✓	✗	✗	✗	✗
X	✗	✗	✗	✗	✗

表锁的加锁机制

隐式上锁

执行 DML 语句或 SELECT…FOR UPDATE 时，除了在对应的行加行锁外，还会在表上隐式地加 RX(ROW EXCLUSIVE) 锁

执行 DDL 操作时会根据 DDL 的类型隐式地加相应的表锁

显式上锁

使用 Lock Table 语句加锁

目前仅 Oracle 模式支持，MySQL 租户无效果

LOCK TABLE table_name [partition|subpartition] IN lock_mode MODE [NOWAIT | WAIT integer] ;

锁机制：行锁

OceanBase 数据库的行锁只支持互斥行锁。读操作不加行锁，写操作才加行锁，读写不冲突，只有并发更新同一行时才会出现写写冲突。

写写冲突：事务的修改需要对行加排他锁，对同一个行的并发更新操作不被允许

读写不冲突：读取数据时通过一致性快照获取已提交事务的数据版本，不需要加行锁，不与正在执行更新的事务发生冲突

行锁的释放：在 OceanBase 数据库中，行锁在事务结束（提交或回滚）的时候释放

锁等待的唤醒：当行锁的持有者（holder）做事务提交时， 会按照等待列表中的顺序唤醒等待行锁的事务

锁等待超时：系统变量 ob_trx_lock_timeout 控制等锁的超时时间，当发生锁等待的时间超过设置时，OceanBase 则会回滚对应的语句，并返回锁超时对应的错误码。用户可以

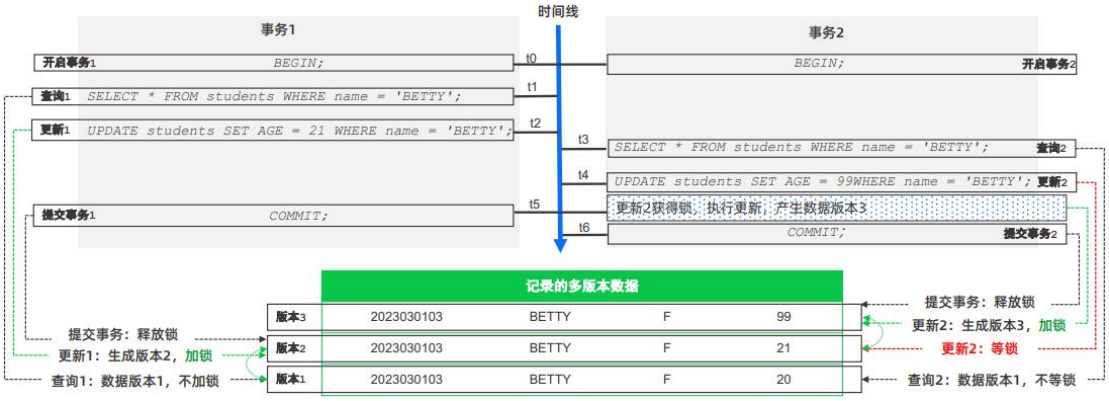
根据应用的特征自主设置

变量的默认值为 -1，即不做锁等待超时处理

如果设置 ob_trx_lock_timeout = 0，则表示锁不等待，一旦发生锁冲突立刻返回锁超时的错误码

行锁与 MVCC 机制下的读与写

OceanBase 通过行锁与多版本并发控制（MVCC）机制一起来控制并发的读、写操作



死锁等待

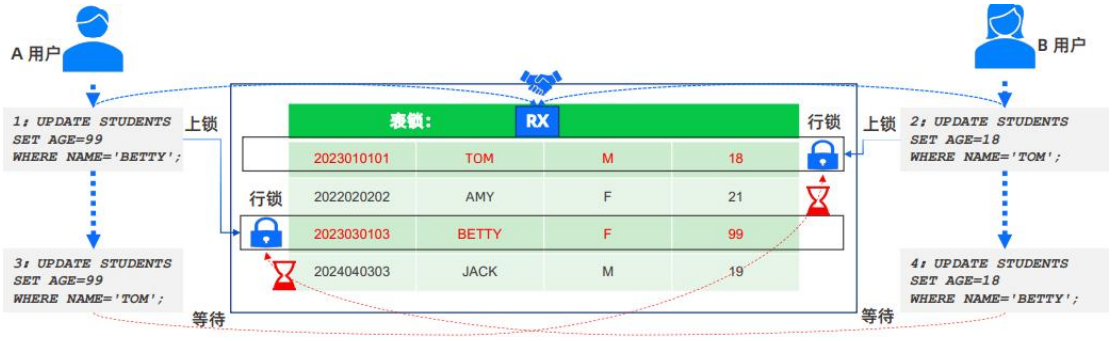
死锁是指两个或多个事务无限期地等待对方持有的资源，导致所有涉及的事务都无法继续执行的状态。在数据库系统中，死锁通常发生在多个事务同时请求锁资源，并且这些锁资源以循环依赖的方式被持有和请求时

A、B 用户互相等待，陷入死锁

A 用户持有 Betty 行的排他锁，申请 Tom 行的排他锁被用户 B 阻塞;

B 用户持有 Tom 行的排他锁，申请 Betty 行的排他锁被用户 A 阻塞;

A、B 用户分别持有对方想要的锁，互不相让



主动死锁检测

OceanBase V4 数据库默认开启主动死锁检测的功能，数据库周期性地检测所有节点上的锁等待，主动发现死锁并解死锁

主动死锁检测：OceanBase 采用 LCL（Lock Chain Length）死锁检测方案，是一种基于优先级的分布式死锁检测方案

目前的优先级指标主要为事务的开启时间，越晚开启的事务具有越低的优先级，越容

易被死锁检测杀掉

主动解死锁是杀掉低优先级事务中等待锁的 SQL 语句（KILL QUERY），而非杀掉整个会话（KILL CONNECTION）

死锁检测结果：系统视图 DBA_OB_DEADLOCK_EVENT_HISTORY 记录了 OceanBase 主动检测并解决的死锁事件

总结

本课程全面阐述了 OceanBase V4 作为单机、分布式一体化的数据库的核心特点和架构，以及分布式数据库的运作机制和底层原理。

集群架构与原理

单机、分布式一体化：更好的性能，更低的资源消耗。可以从单机数据库动态扩展为分布式

原生多租户：每一个租户都是单独的数据库实例，独占租户的资源和数据，拥有租户级的系统视图，变量与参数

资源隔离：租户之间的 CPU、内存等资源强隔离

数据分片：分区是数据分片的单位，不同的分区分布式存储

日志流副本：租户级、单机日志流，以日志流为单位进行副本管理与同步

分布式一致性协议：使用 Multi-Paxos 协议进行日志流副本的同步和选举

负载均衡机制：租户间的资源单元均衡，以及租户内的日志流、分区、数据量的均衡

LSM-Tree 存储架构：读写分离的准内存数据库，多级转储架构的 Compaction 机制

分布式与并行：OceanBase 分布式数据库下的并行处理机制

事务控制：MVCC、多版本一致性读，隔离级别与锁机制

两阶段提交协议：优化的两阶段提交协议，无状态的协调者，更好的分布式事务性能

通过学习该课程，我们能够对集群、租户进行资源分配和管理，对内存进行管理和监控，能够合理分布数据负载与业务流量，制定适当的转储、合并策略，并能够指导应用开发根据业务需求进行合理的事务并发控制。

集群管理

集群资源分配：OBServer 资源初始化，租户资源分配，资源使用监控

均衡方案设计：制定合适的负载分布方案，手动迁移资源单元、分区，通过 Primary Zone 对业务流量进行均衡

存储管理：根据系统负载、业务情况，制定合适的 Compaction 策略，避免系统抖动

内存管理：主要内存模块的管理与监控，避免出现内存用满的问题

并发线程管理：根据系统资源与负载情况，设置合理的 SQL 并发度策略

事务管理：对数据库事务进行控制管理，指导应用开发合理设置事务的隔离级别、一致性读级别，对事务、锁进行监控与管理

数据库运维与监控

集群管理

配置项管理

配置项

在 OceanBase 数据库中通过配置项的设置可以控制集群的功能行为，例如负载均衡、合并时间、合并方式、资源分配和模块开关等功能

配置项的级别

集群级配置项：作用范围为整个集群所有 OBServer 节点

租户级配置项：作用范围为当前租户在集群内的所有资源单元

配置项的生效方式（若无特别提示，本教材中的所涉及的配置项均为动态生效）

动态生效（dynamic_effective）：一经修改，动态生效，无需重启集群或 OBServer

重启生效（static_effective）：配置项的修改需要重启集群或 OBServer 后生效

配置项的持久化

内部表：配置项的修改自动同步到内部表，可以通过系统视图 GV$OB_PARAMETERS 查看

配置文件：配置项的修改自动同步到配置文件中，配置文件的默认位置为 home/admin/oceanbase/etc/observer.config.bin

OBServer 启动时会对配置文件进行 checksum 校验，手动修改配置文件中的内容不生效

查看配置项：通过命令与系统视图

可以使用 SHOW PARAMETERS 命令或者查询系统配置项视图 GV$OB_PARAMETERS 来查看配置项的设置（名称以 "_" 开头的配置项称为隐藏配置项，只能使用 GV$OB_PARAMETERS 视图查看，无法使用 SHOW 命令查看）

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

LIKE 与 WHERE 二选一

SYS 租户查看租户级配置项时，需要指定 TENANT；在用户租户查看配置项时，不可指定 TENANT

GV$OB_PARAMETERS 视图

在 SYS 租户可以查看所有租户的配置项；用户租户只能查看集群级配置项和本租户的配置项

示例：在 SYS 租户中查看用户租户 oracle 的租户级配置项

SELECT * FROM GV$OB_PARAMETERS WHERE NAME LIKE 'freeze%' AND TENANT_ID=1004;SHOW PARAMETERS LIKE 'freeze%' TENANT=oracle；

查看配置项：通过配置文件

可以使用配置文件查看配置项的设置与变更历史

配置文件

observer.config.bin：实时同步最新的配置项设置

observer.config.bin.history：记录最近一次配置项变更之前的配置项设置

设置配置项：通过 ALTER SYSTEM 命令

可以使用 ALTER SYSTEM 命令修改集群和租户的配置项

ALTER SYSTEM SET PARAMETER 命令（MySQL 模式）

SCOPE：本次配置项修改的生效范围

SPFILE：表示仅修改配置文件中的配置项值，本次修改不立刻生效，OBServer 节点重启以后才生效

BOTH：表示既修改配置表，又修改内存值，修改后即生效（需要重启生效的配置项除外），且 OBServer 节点重启后仍然生效

SERVER：指定集群中待修改配置项值的 OBServer 节点，仅支持指定一个 OBServer 节点。 默认为所有 OBServer

ZONE：指定集群中待修改配置项值的 Zone，仅支持指定一个 Zone。默认为所有 Zone

TENANT：在系统租户中修改租户级配置项时需指定用户租户的名字。如果不指定，默认为本租户

sys 代表系统租户，all_user 代表所有用户租户，all_meta 代表所有 Meta 租户

Oracle 模式的租户无法在使用 ALTER 命令设置配置项时指定 SCOPE、ZONE、SERVER 等选项

设置配置项：通过 –o 启动选项

也可以在启动 observer 进程时通过 –o 选项设置集群配置项

observer 进程启动命令

cd /home/admin/oceanbase

./bin/observer -I 10.10.10.1 -P 2882 ... -o "cpu_count=32,memory_size=128G"

-o：指定集群启动配置项列表

可以使用 –o 选项同时为多个配置项指定值，多个配置项的值之间使用英文逗号进行分隔

为 observer 启动命令的可选项

首次启动 observer 时可根据需要设置相应的配置项，比如：

cpu_count：用于设置 observer 进程的可用 CPU 总数

memory_limit：用于设置 observer 进程的可用内存大小

system_memory：用于设置预留给数据库系统的保留内存

使用 OCP 管理系统参数

路径：集群--参数管理、租户--参数管理
集群参数管理既可以管理集群级配置项，也可以管理租户级配置项
租户参数管理既可以管理租户级配置项，也可以管理租户的全局变量

集群运维与监控

查看节点状态

OceanBase 数据库是单进程软件，进程名为 observer。通常一台服务器运行一个 observer 进程，由 IP 和端口作为唯一标识，称之为节点。

SELECT ZONE,SVR_IP,WITH_ROOTSERVER,STATUS,START_SERVICE_TIME,STOP_TIME,LAST_OFFLINE_TIME FROM DBA_OB_SERVERS;

主要字段	描述
ZONE	OBServer 节点所属的 Zone
SVR_IP	服务器 IP 地址
WITH_ROOTSERVER	是否存在 Root Service 服务
STATUS	• ACTIVE：节点心跳状态正常 • INACTIVE：节点心跳状态异常 • DELETING：节点正在删除中

主要字段	描述
START_SERVICE_TIME	节点启动后，开始对外服务的时间点。 • NULL：节点未启动或状态不可用 • 有效值：节点开始对外服务的时间
STOP_TIME	节点停止服务的时间点。 • NULL：节点没有被 Stop • 有效值：节点被 Stop 的时间
LAST_OFFLINE_TIME	OBServer 节点上次下线的时间 • NULL：节点没有下线过 • 有效值：节点的上次下线时间

重启节点

重启是常见运维动作之一，适用于对机器进行短暂维修，以及修改系统配置项后需要重启生效的场景。

重启节点从 OceanBase 层面包括 Stop Server 与 Start Server 的过程，在操作系统层面还包括 observer 进程的重启

STOP SERVER：停止 OBServer 节点的服务，STOP SERVER 命令执行以下逻辑：

将待重启节点上的 Leader 全部切走，并保证除了重启节点以外的其他节点上的副本满足多数派

当 OBServer 停止完成后，Root Service 将节点标记为服务停止状态，此时 DBA_OB_SERVERS 视图的状态标识为：STATUS=ACTIVE，STOP_TIME>0

OBProxy 识别 OBServer 的状态为停止后，不会将业务请求路由到该节点

START SERVER：启动 OBServer 节点的服务，START SERVER 命令执行以下逻辑

将 Clog 或者基线数据与其他副本进行同步（补齐），并将上一次合并之后的内存数据恢复出来（clog 回放）

Root Service 调度负载均衡来完成 Leader 切回等任务

OBServer 启动完成后，Root Service 将节点标记为可服务状态，此时 DBA_OB_SERVERS 视图的状态标识为：STATUS=ACTIVE，STOP_TIME IS NULL，

START_SERVICE_TIME>0

OBProxy 识别 OBServer 的状态为可服务状态后，会将相应的业务请求路由到该节点上

重启节点的操作步骤

重启节点的主要流程为：停止服务 -> 转储 -> 关闭进程 -> 启动进程 -> 启动服务

使用 root 用户登录到集群的 SYS 租户

停止服务（若剩余可服务的节点不满足多数派的要求，则整个集群停止服务）:ALTER SYSTEM STOP SERVER '10.xx.xx.10:2882';

对待重启的节点进行转储操作，以便缩短重启后回放 Redo Log 的时间，加速重启:ALTER SYSTEM MINOR FREEZE SERVER = ('10.xx.xx.10:2882')

在操作系统中使用 admin 用户停止 observer 进程:kill -9 $(pidof observer)

（可选）执行相对应应的运维操作

在操作系统中使用 admin 用户启动 observer 进程： cd /home/admin/oceanbase && ./bin/observer

启动服务 ALTER SYSTEM START SERVER '10.xx.xx.10:2882';

重启节点的注意事项

重启过程中节点的下线时间需要在配置项 server_permanent_offline_time 设置的时间以内，否则会被永久下线。

server_permanent_offline_time：集群级配置项，用于设置节点心跳中断的时间阈值，即节点心跳中断多久后认为其被永久下线

永久下线意味着节点上的副本从 Paxos Group 中剔除，也即数据失效。节点再次上线时，Oceanbase 数据库需要进行节点上的数据副本的重建，通常是一个比较耗时的操作

默认值为 3600s，即 1 小时。该配置项的适用场景及建议值如下

OceanBase 数据库版本升级场景：建议将该配置项的值设置为 72h

OBServer 节点硬件更换场景：建议将该配置项的值设置为 4h

OBServer 节点清空上线场景：建议将该配置项的值设置为 10m，使集群快速上线

ALTER SYSTEM SET server_permanent_offline_time = '72h' ;

添加、删除节点

通过添加、删除 OBServer 节点来实实现对 Oceanbase 对集群的弹性扩缩容。

ADD SERVER：为 Zone 内添加新的 OBServer，从而可以执行后续的迁移 Unit、调整租户的 UNIT_NUM、新建租户等操作

待添加的 OBServer 服务器按照 OceanBase 的部署要求进行相应的配置，并完成安装 OceanBase 数据库软件

待添加的 OBServer 服务器上已启动 observer 进程

执行命令：ALTER SYSTEM ADD SERVER 'svr_ip:svr_port' ['svr_ip:svr_port'...] [ZONE [=] 'zone_name'];

DELETE SERVER：删除 OBServer 节点，通常用于集群缩容或者故障节点替换

Delete Server 操作涉及到负载均衡，被删除的节点上的资源单元（称为 Unit ）会在同一个 Zone 中进行迁移。待 Unit 迁移成功，Delete Server 操作即可执行成功

Unit 的迁移动作是 Unit 自动均衡的过程，主要由 Root Service 控制

如果您想要选择待删除节点上 Unit 迁移的目标机器，可以手动执行 Unit 迁移

执行命令：ALTER SYSTEM DELETE SERVER 'svr_ip:svr_port' [,'svr_ip:svr_port'...] [ZONE [=] 'zone_name'];

替换节点的操作步骤

替换节点适用于硬件故障场景、容量场景（不同规格机器替换），替换后集群的节点数量不变。替换节点是一套运维操作的组合：即先添加新节点并迁移副本，然后再删除旧节点，整体时间较长。

使用 root 用户登录到集群的 SYS 租户

在待替换节点所在的 Zone 上添加新节点（10.xx.xx.11）：ALTER SYSTEM ADD SERVER '10.xx.xx.11:2882' ZONE 'zone1';

通过 DBA_OB_UNITS 视图查询旧节点（10.xx.xx.10）上的 Unit：SELECT UNIT_ID FROM oceanbase.DBA_OB_UNITS WHERE SVR_IP = '10.xx.xx.10';

执行 MIGRATE UNIT 命令迁移 Unit：ALTER SYSTEM MIGRATE UNIT 1016 DESTINATION '10.xx.xx.11:2882';

每次仅可迁移一个 Unit。如果旧节点上有多个 Unit，则需多次执行该命令

等待旧节点上的所有 Unit 迁移完成后，将旧节点删除：ALTER SYSTEM DELETE SERVER "10.xx.xx.10:2882" zone='zone1'

查询 DBA_OB_SERVERS 视图，确认旧节点是否删除成功

查看可用区状态

OceanBase 集群由若干个 Zone 组成。从物理层面来讲，一个 Zone 通常是一个独立的物理部署单元，可以是一个数据中心（IDC）或者云上的一个 Zone（可用区），也可以是一个单独的机架（Rack）

SELECT * FROM DBA_OB_ZONES；

可用区的运维

可用区的运维通常包括 Zone 的停止与启动，Zone 的属性修改，以及添加、删除 Zone。

STOP ZONE：停止 Zone 内所有 OBServer 节点的服务，其执行逻辑与 Stop Server 类似（ALTER SYSTEM STOP ZONE zone_name;）

START ZONE：启动 Zone 内所有 OBServer 节点的服务，其执行逻辑与 Start Server 类似（ALTER SYSTEM START ZONE zone_name;）

ALTER ZONE：修改 Zone 所属的 Region 及 IDC 信息（ALTER SYSTEM ALTER ZONE zone_name SET [IDC [=]'idc_name', REGION [=]'region_name'];）

ADD ZONE：添加 Zone 通常用于集群的扩容场景中，添加 Zone 之和还要添加 OBServer 节点（ALTER SYSTEM ADD ZONE zone_name [IDC [=] 'idc_name', REGION [=]

'region_name'];）

DELETE ZONE：删除 Zone 通常用于集群缩容的场景中，删除 Zone 的前提是 Zone 内的 OBServer 节点已经全部删除（ALTER SYSTEM DELETE ZONE zone_name [IDC [=] 'idc_name', REGION [=] 'region_name'];）

OBServer 资源分配视图

字典视图 DBA_OB_SERVERS 与性能视图 GV$OB_SERVERS 均可以展示 OBServer 的资源分配情况，所不同的是，后者还动态显示磁盘的使用情况。

GV$OB_SERVERS 主要字段：

资源类型	字段名称	描述
CPU	CPU_CAPACITY	节点 CPU 总容量
	CPU_CAPACITY_MAX	节点 CPU 总容量的超卖值。**CPU_CAPACITY_MAX = CPU_CAPACITY * resource_hard_limit**
	CPU_ASSIGNED	节点已经分配的 CPU 数量，它是节点上所有 Unit 的 MIN_CPU 规格总和
	CPU_ASSIGNED_MAX	节点已经分配的 CPU 上限值，它是节点上所有 Unit 的 MAX_CPU 规格总和
内存	MEMORY_LIMIT	节点总的内存大小
	MEM_CAPACITY	节点总的可分配内存大小。**MEM_CAPACITY = MEMORY_LIMIT - SYSTEM_MEMORY**
	MEM_ASSIGNED	节点已分配的内存大小，它是节点上所有 Unit 的 MEMORY_SIZE 规格总和
日志盘	LOG_DISK_CAPACITY	节点的日志盘空间大小
	LOG_DISK_ASSIGNED	节点已分配的日志盘大小，它是节点上所有 Unit 的 LOG_DISK_SIZE 规格总和
	LOG_DISK_IN_USE	节点实际使用的日志盘大小
数据盘	DATA_DISK_CAPACITY	节点的数据盘空间大小
	DATA_DISK_IN_USE	节点实际使用的数据盘大小

查看系统任务与历史事件

OceanBase 数据库 V4 版本提供了一系列的系统视图来监控系统后台任务的执行，并记录后台任务执行的历史（默认保留最近一周的历史任务）

可以在 SYS 租户使用以下 SQL 查询系统定义的任务视图

SHOW TABLES [FROM oceanBase] LIKE 'DBA_OB%JOBS' ; // 查找系统作业视图

SHOW TABLES [FROM oceanBase] LIKE 'DBA_OB%TASKS' ; // 查找系统任务视图

JOB 代表一个特定的系统工作。一个 JOB 可以被拆分成可以并行执行的一个或多个 TASK。比如租户内的负载均衡，一个租户在任一时刻只有一个负载均衡工作（BALANCE JOB），每个工作会生成多个负载均衡任务（BALANCE TASK）并行执行

OBS 与 TASKS 视图通常只展示正在执行的作业和任务

可以在 SYS 租户通过 DBA_OB_ROOTSERVICE_EVENT_HISTORY 查询 RootService 调度的历史任务，比如负载均衡、容灾切换、合并等

SELECT * FROM DBA_OB_ROOTSERVICE_EVENT_HISTORY where module like "%disaster%" and value1 = $TENANT_ID and value2 =$LS_ID;

可以在 SYS 租户通过 DBA_OB_SERVER_EVENT_HISTORY 查询 OBServer 节点调度的历史任务，比如选举、转储等

SELECT * FROM DBA_OB_ROOTSERVICE_EVENT_HISTORY where module like "%election%" and value1 = $TENANT_ID and value2 =$LS_ID;

磁盘管理

数据磁盘 IO 性能校准

OceanBase 根据不同的数据磁盘 IO 性能自适应地调整数据库系统的行为，以达到最佳的使用状态。磁盘 IO 性能校准通常在 OceanBase 数据库安装过程中的自动执行。在需要重新校准磁盘 IO 性能的场景，比如更换磁盘，OceanBase 为用户提供了主动进行磁盘 IO 校准的能力。

在 SYS 租户主动执行 IO_CALIBRATION 作业校准磁盘 IO 性能：

ALTER SYSTEM RUN JOB "IO_CALIBRATION" SERVER = 'ip:port' ;

SYS 租户通过系统视图 GV$OB_IO_CALIBRATION_STATUS 查看磁盘 IO 校准任务的执行状态

SELECT * FROM oceanbase.GV$OB_IO_CALIBRATION_STATUS;

SYS 租户通过系统视图 GV$OB_IO_BENCHMARK 查看磁盘 IO 的校准结果（磁盘性能数据）

SELECT * FROM oceanbase.GV$OB_IO_BENCHMARK;

手动刷新磁盘 IO 性能数据

调用 IO_CALIBRATION 执行磁盘 IO 校准任务会对数据磁盘进行较长时间的大量读写操作，如果当前 OceanBase 集群中已有负载，磁盘 IO 校准会对业务执行产生性能影响。因此，OceanBase 数据库也提供了直接修改磁盘 IO 性能数据的手段。

在 SYS 租户主动执行 REFRESH IO CALIBRATION 命令设置磁盘 IO 性能数据

ALTER SYSTEM REFRESH IO CALIBRATION [STORAGE = 'DATA'][CALIBRATION_INFO [=] ("mode : size : latency : iops" [, "mode : size : latency : iops"])][ZONE [=] zone_name] [SERVER [=] 'svr_ip:svr_port'];

CALIBRATION_INFO：待刷新的磁盘校准信息。如果指定为""，则表示将磁盘性能数据清空

ZONE：刷新指定 Zone 内所有 OBServer 的磁盘校准信息

SERVER：刷新指定 OBServer 的磁盘校准信息

示例：清空集群内所有 OBServer 的磁盘性能信息：ALTER SYSTEM REFRESH IO CALIBRATION CALIBRATION_INFO = ("");

示例：刷新 zone1 内所有 OBServer 节点的数据磁盘的校准信息，4K 读的性能为 IPOS = 200000，读 IO 的 latency 是 100us

ALTER SYSTEM REFRESH IO CALIBRATION STORAGE = 'DATA' CALIBRATION_INFO = ("read:4K:100us:200000) ZONE = zone1；

数据磁盘文件分配模式

在 OceanBase V4.2 版本中，磁盘数据文件的分配同时支持预分配模式和动态扩容模式

不同版本的行为差异

V4.2.0 之前：OceanBase 采用预分配的方式将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

V4.2.0 及之后：OceanBase 数据库还支持根据磁盘数据文件的实际使用情况来进行自动扩容

在 OceanBase V4.2 版本中，数据文件的分配与扩容由以下配置项来控制

配置项	级别	默认值	描述
datafile_size	集群级	0	设置 OBServer 初始分配的数据文件的大小，默认单位为 MB。
datafile_disk_percentage	集群级	0	设置 OBServer 初始分配的数据文件占其所在磁盘空间的百分比。
datafile_next	集群级	0	设置 OBServer 磁盘数据文件自动扩容的步长，默认单位为 MB。
datafile_maxsize	集群级	0	设置 OBServer 磁盘数据文件自动扩容的最大空间，默认单位为 MB。

预分配模式：当 datafile_maxsize 与 datafile_next 为 0 时，datafile_size 和 datafile_disk_percentage 确定了预分配的磁盘文件的大小

动态扩容模式：当 datafile_maxsize 与 datafile_next 大于 0 时，datafile_size 和 datafile_disk_percentage 确定了初始分配的磁盘文件大小，datafile_next 确定了每次扩容的大小，datafile_maxsize 确定了最大的磁盘空间大小

开启数据文件的动态扩容

可以通过对相关配置项的修改，将磁盘数据文件的分配方式从预分配模式修改为动态扩容模式

操作步骤

使用 root 用户登录到集群的 SYS 租户

使用 SHOW PARAMETERS 命令查看当前集群的相关配置项设置

通过系统视图 GV$OB_SERVERS 查看当前 OBServer 已经分配和使用的数据磁盘空间大小

SELECT SVR_IP, CONCAT(ROUND((DATA_DISK_CAPACITY)/1024/1024/1024,1),' GB') AS DATA_DISK_CAPACITY,CONCAT(ROUND((DATA_DISK_ALLOCATED)/1024/1024/1024,1),' GB') AS DATA_DISK_ALLOCATED,CONCAT(ROUND((DATA_DISK_IN_USE)/1024/1024/1024,1),' GB') AS DATA_DISK_IN_USE FROM GV$OB_SERVERS;

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

使用限制：当前版本暂不支持磁盘数据文件的动态缩容，如果 DATA_DISK_ALLOCATED 已经是数据磁盘的最大空间，则无法通过将 datafile_size 改小的方式实现动态扩容

配置建议：建议将 datafile_next 设置为 datafile_maxsize 的 20% 左右，避免频繁扩容

日志磁盘空间管理

日志磁盘用来保存数据库的事务日志和存储日志，日志以一个个独立的文件存放在磁盘上。日志磁盘的空间使用随着新的日志文件的创建而增长，随着旧的日志文件的回收而释放。

log_disk_utilization_threshold，租户级，默认值 80, 用于设置租户日志盘利用率阈值，当租户日志盘使用量超过租户日志盘空间总量乘以该值时，进行旧日志文件收回。

log_disk_utilization_limit_threshold，租户级,默认值 95 ,用于设置租户日志盘利用率限

制阈值。当租户日志盘使用量超过租户日志盘空间总量乘以该值时，不再允许日志写入。

log_storage_compress_all ，租户级，默认值 False，用于控制是否开启 Clog 存储压缩功能（该配置项从 V4.2.1 BP5 版本开始引入）。

日志的回收

可回收日志：当日志中所有的事务的数据更新已经转储落盘，则日志处于可回收状态

回收的时机：当租户日志磁盘的空间使用率超过 log_disk_utilization_threshold 时，OceanBase 会对可回收的日志进行回收

日志的停写：当租户日志磁盘的空间使用率超过 log_disk_utilization_limit_threshold 时，OceanBase 不允许日志写入，即禁止事务执行

日志的压缩：基于日志落盘的性能考虑，默认设置日志存储不使用压缩。如果存在日志磁盘使用紧张的问题，则建议开启日志存储压缩

日志写入限速

日志回收的前提是转储，当日志写入的速度大于转储的速度时，则可能导致日志盘满和日志停写，对业务造成严重影响。为了避免这个问题，V4.2 版本引入了日志写入限速的功能。

日志写入限速：在日志盘使用量达到一定水位时，通过限制日志提交写盘的速度，来避免写入压力特别大的集群频繁出现日志盘满的现象

log_disk_throttling_percentage， 租户级， 默认值 60，用于触发日志写入限速的不可回收日志盘空间占比。

log_disk_throttling_percentage 的值越小，则日志写入限速开始的时机越早

log_disk_throttling_maximum_duration 的值越大，则预期日志限速支撑的时间越长，限速的力度也越大

log_disk_throttling_maximum_duration，租户级，默认值 2h，用于调整触发日志限速后，日志盘的最大可用时间。

配置建议：对于写入压力和转储速度差距较大的场景，可以尝试将 log_disk_throttling_percentage 调小，或将 log_disk_throttling_maximum_duration 调大，建议优先调大 log_disk_throttling_maximum_duration。

限速的影响：日志写入限速会影响 DML、事务等的执行速度，可能导致 SQL 执行变慢甚至超时，事务超时等现象。

租户管理

系统变量管理

租户系统变量

在 OceanBase 数据库中，租户系统变量用来控制数据库系统、会话和 SQL 的各种行为，如最大连接数、SQL 执行的内存大小、超时设置等。OceanBase 数据库的租户系统变量分为全局变量和 Session 变量。

全局变量：表示 Global 级别的设置，数据库同一租户内的不同用户共享全局变量
全局变量修改后，仅对新建立的 Session 生效，对当前已打开的 Session 不生效
全局变量的修改会进行持久化，不会随会话的退出而失效
会话变量：表示 Session 级别的设置，仅控制本 Session 的行为
新建 Session 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
Session 级别的变量不会进行持久化，随会话的退出而失效

使用 SHOW 命令查看系统变量

可以使用 SHOW VARIABLES 命令查询全局变量和会话变量的设置
SHOW VARIABLES 命令
SHOW [GLBOAL|SESSION] VARIABLES [LIKE 'pattern' | WHERE expr]
SESSION | GLOBAL：SESSION 表示会话变量，GLOBAL 表示全局变量。不填写默认查看 Session 变量
LIKE 与 WHERE 二选一

使用系统视图查看系统变量

OceanBase 数据库也提供了系统视图来查询全局变量与会话变量的设置。（名称以 "_" 开头的变量称为隐藏变量，只能使用系统视图查看，无法使用 SHOW 命令查看）

租户	视图名称	功能描述
SYS 租户	oceanbase.CDB_OB_SYS_VARIABLES	所有租户的全局变量信息
MySQL 租户（包含SYS租户）	information_schema.GLOBAL_VARIABLES	本租户的全局变量信息
	information_schema.SESSION_VARIABLES	本 SESSION 的变量信息
Oracle 租户	SYS.TENANT_VIRTUAL_GLOBAL_VARIABLE	本租户的全局变量信息
	SYS.TENANT_VIRTUAL_SESSION_VARIABLE	本 SESSION 的变量信息

修改系统变量

通过系统变量的设置可以使 OceanBase 数据库的行为符合业务的要求，租户的系统变量可以通过 SET VARIABLE 语句修改
SET VARIABLE 命令（SET [GLOBAL|SESSION] variable_name = expression[,variable_name = expression...]）
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，需要重新登录后，建立新的 Session 才会生效
可以同时修改多个系统变量

在 SYS 租户修改用户租户的全局变量

OceanBase 提供了在 SYS 租户内修改用户租户的全局变量的能力，可以在因为用户租户全局变量设置错误而无法连接用户租户时提供帮助 。

CHANGE TENANT 命令（ALTER SYSTEM CHANGE TENANT tenant_name|TENANT_ID=tenant_id;）

切换命令中即可以使用租户名，也可以使用租户 ID

该命令仅可在 SYS 租户内执行

不能通过 OBProxy 连接执行该命令，必须直连 OBServer，且 OBServer 上要有目标用户租户的资源单元

租户运维与监控

查看集群中的租户

OceanBase 是原生多租户架构，一个用户租户就是一个独立的数据库实例。

DBA_OB_TENANTS：租户的定义视图。在 SYS 租户中展示集群中的所有租户，在用户租户中仅展示本租户的信息

```
SELECT TENANT_ID,TENANT_NAME,TENANT_TYPE,PRIMARY_ZONE,LOCALITY FROM DBA_OB_TENANTS;

+-----------+-------------------------+-------------+--------------------+-------------------------------------------+
| TENANT_ID | TENANT_NAME             | TENANT_TYPE | PRIMARY_ZONE       | LOCALITY                                  |
+-----------+-------------------------+-------------+--------------------+-------------------------------------------+
|         1 | sys                     | SYS         | RANDOM             | FULL{1}@zone1, FULL{1}@zone2, FULL{1}@zone3 |
|      1001 | META$1002               | META        | RANDOM             | FULL{1}@zone1, FULL{1}@zone2, FULL{1}@zone3 |
|      1002 | oracle                  | USER        | RANDOM             | FULL{1}@zone1, FULL{1}@zone2, FULL{1}@zone3 |
+-----------+-------------------------+-------------+--------------------+-------------------------------------------+
```

TENANT_ID：租户在集群内的唯一识别号，1 为 SYS 租户。META 和用户租户的 ID 从 1001 开始，META 租户的 ID 比其对应的用户租户 ID 小 1

META 租户总是与用户租户一一对应，META$1002 是 oracle 租户的 Meta 租户

TENANT_TYPE：租户类型，目前有 SYS、META 和 USER 三种

PRIMARY_ZONE：Zone 的优先级设置。RANDOM 表示各个 Zone 具有相同的优先级，Leader 副本会平均分布在各个 Zone 内

SYS 租户的 Leader 副本总是集中在一个 Zone 内

LOCALITY：副本的分布。FULL{1}@zone1 表示在 zone1 内有 1 个全功能型副本

PREVIOUS_LOCALITY：租户之前的副本分布

ALTER TENANT

ALTER TENANT 语句是最常用的租户运维命令，用来修改用户租户的属性 。

ALTER TENANT 命令

可以使用 ALTER TENANT ... RENAME 命令对用户租户进行重命名（ALTER TENANT old_tenant_name RENAME GLOBAL_NAME TO new_tenant_name;）

可以使用 ALTER TENANT ... LOCK / UNLOCK 命令对用户租户执行锁定与解锁（ALTER TENANT tenant_name LOCK | UNLOCK）

锁定租户后，系统将不允许用户在被锁定的租户上创建新的连接，已存在的连接不受影响

可以使用 ALTER TENANT SET 命令修改租户的属性，包括 Locality、主 Zone、资源池列表以及全局变量值

ALTER TENANT tenant_name[SET] [tenant_option [, tenant_option ...]] [set_sys_var];

tenant_option:LOCALITY [=] 'locality_description'| PRIMARY_ZONE [=] zone| RESOURCE_POOL_LIST [=](pool_name [, pool_name...])

set_sys_var:VARIABLES var_name {TO | =} var_value [,var_name {TO | =} var_value...]

修改 Locality

Locality 描述了数据的多个副本的类型及分布策略。通过修改 Locality 属性可以调整租户的部署架构，适用于机房搬迁、调整容灾级别等场景。

Locality 的语法（replicas{量词}@location）

replicas：副本类型，即 FULL 或 READONLY，也可用 F 或 R 代替

量词：副本在指定可用区内的数量，固定为 1

location：可用区的名字

修改 Locality 的示例（ALTER TENANT mysql LOCALITY="F@zone1,F@zone2,R@zone3";）

每个 Zone 内进可创建 1 份副本，或者是全功能型副本，或者是只读型副本

资源单元时副本的容器，在 Zone 内分配副本前，租户需要在 Zone 内有对应的资源单元，也即资源池

修改 Locality 的命令会立即返回用户，异步生效。可以通过 DBA_OB_TENANTS 视图查看租户 Locality 的修改结果：

LOCALITY 字段表示租户最新的 Locality 设置

PREVIOUS_LOCALITY 仅在 Locality 修改过程中有值，当 PREVIOUS_LOCALITY 为空时，表示 Locality 的修改已完成

修改 Primary Zone

Primary Zone 决定了 OceanBase 数据库的流量分布。通过修改 Primary Zone 属性可以切换业务流量，适用于容灾场景、扩缩容等场景。

Primary Zone 的语法：PRIMARY_ZONE [=] zone

RANDOM：指定租户的 Primary Zone 为租户所在的所有 Zone，表示租户所在的所有 Zone 具有相同的优先级

zone_list：指定租户的 Primary Zone 为具体的 Zone 的集合。用分号（;）分割表示不同的优先级，用逗号（,）分隔表示相同的优先级，逗号两侧优先级相同，分号左侧优先级高于右侧

修改 Primary Zone 的示例：ALTER TENANT mysql PRIMARY_ZONE='zone1,zone2,zone3';

修改 Primary Zone 的命令会立即返回用户，异步生效。Primary Zone 的修改最终会影响日志流的 Leader 副本的重分布，可以通过 CDB_OB_LS_LOCATIONS（或 DBA_OB_LS_LOCATIONS）视图来确认修改结果

租户扩缩容

随着用户业务的发展或衰退，同时业务在不同时段的峰值往往也不同，需要数据库提供弹性的服务能力，OceanBase 数据库通过租户扩缩容功能来支持业务的弹性服务能力。

方法 1：垂直扩缩容

修改资源配置：直接修改资源配置的 CPU 或 Memory 等值，进而直接影响租户在该资源单元的资源规格和服务能力。（适用于租户使用独立的资源配置的场景）

切换资源配置：创建一个新的资源配置，将租户资源池的资源配置切换为新的配置，从而达到租户资源变更的目的。（适用于多个租户使用相同资源配置的场景）

方法 2：水平扩缩容

修改资源池定义中资源单元的数量

每个 Zone 内的资源单元个数要相同

资源池变更要以租户为单位，需同步修改租户所有 Zone 内的资源池

查看租户的资源配置

资源配置（Unit Config）确定了租户资源单元的规格，可以通过系统视图查看租户的资源配置。

DBA_OB_UNIT_CONFIGS：SYS 租户的字典视图，展示集群中所有的资源配置信息

DBA_OB_RESOURCE_POOLS：SYS 租户的字典视图，展示所有租户的资源池的信息，可以关联租户与资源配置

租户资源垂直扩缩容

OceanBase 支持动态调整资源配置来实现业务无感的租户资源的垂直扩缩容。（仅 sys 租户的 root 用户（root@sys）可以创建、修改资源规格）

修改租户的资源配置（ALTER RESOURCE UNIT unit_name [MAX_CPU [=] cpunum,] [MIN_CPU [=] cpunum,] [MEMORY_SIZE [=] memsize,] [MAX_IOPS[=] iopsnum,] [MIN_IOPS [=] iopsnum,] [IOPS_WEIGHT [=]iopsweight,] [LOG_DISK_SIZE [=] logdisksize];）

修改资源规格时，可修改 CPU、内存、IOPS 及日志盘规格中的若干项，没有被修改的值将保持不变

在调大资源规格前，如果该资源规格对应的 Unit 正在被租户使用，必须保证各节点有足够的剩余资源可用于分配，即所有资源单元的资源之和不能超过节点的资源分配上限（可以从系统视图 GV$OB_SERVERS 中获知各个节点的资源分配上限）

查看租户的资源单元

资源单元是租户的容器，提供完整的计算与存储服务，资源单元的大小和数量决定了租户的服务能力

DBA_OB_UNITS： SYS 租户的字典视图，展示所有租户的资源单元的配置信息

GV$OB_UNITS： SYS 租户的性能视图，除了展示 DBA_OB_UNITS 所包含的静态配置信

息，还展示磁盘空间的实际使用信息

UNIT_ID：资源单元在集群内的唯一标识

UNIT_GROUP_ID：同一个租户的承载相同日志流副本的 Unit 属于同一个 Unit Group

修改租户的资源池

资源池的修改主要包括资源配置的更换和资源单元个数的改变，OceanBase 支持动态调整资源池来实现业务无感的租户资源的弹性扩缩容。（仅 sys 租户的 root 用户（root@sys）可以执行以上命令）

ALTER RESOURCE POOL 命令：修改资源池的属性，包括修改 UNIT、UNIT_NUM 和 ZONE_LIST（ALTER RESOURCE POOL pool_name UNIT [=] unit_name | UNIT_NUM [=] unit_num [DELETE UNIT = (unit_id_list)] |ZONE_LIST [=] ('zone_name' [, 'zone_name' ...]);）

ALTER RESOURCE POOL 语句每次仅可修改一种属性，即不能同时修改 UNIT、UNIT_NUM 和 ZONE_LIST

修改 UNIT_NUM 时，如果资源池已分配给租户使用，则需要使用 ALTER RESOURCE TENANT 语句进行修改

减小 unit_num 时，使用 DELETE UNIT 可以删除指定的 Unit

ALTER RESOURCE TENANT 命令：修改租户所有资源池的 Unit 个数（ALTER RESOURCE TENANT tenant_name UNIT_NUM [=] unit_num [DELETE UNIT_GROUP = (unit_group_id_list)];）

减小 unit_num 时，使用 DELETE UNIT_GROUP 可以删除指定的 Unit GROUP

租户资源水平扩缩容

从 V4.0.0 版本开始，OceanBase 数据库要求租户内每个 Zone 的 Unit 个数必须保持一致，租户资源的水平扩缩容要求在所有 Zone 内同时进行。

前提条件：修改 Unit 个数会导致触发负载均衡操作，包括日志流的合并、分裂等操作，分区数与存储量的均衡，需要打开负载均衡开关。

租户配置项 enable_rebalance：控制是否进行租户内均衡，默认值为 true，需要设置为 true

租户配置项 enable_transfer：控制是否开启租户下的 Transfer 功能，默认值为 true，需要设置为 true

调大 Unit 个数：调大 Unit 个数时，须确保每个 Zone 内有多余的 OBServer 来分配新的 Unit，且这些 OBServer 上有充足的资源

调小 Unit 个数：调小 Unit 个数时，可以指定资源单元的 Unit_ID 或 Unit_Group_ID 来删除特定的资源单元

用户与安全

用户管理

用户名称在租户内是唯一的，不同租户的用户可以同名，通过 用户名@租户名 的形式在系统全局唯一确定一个用户。用户分为两类：系统租户下的用户，一般租户下的用户。

数据库用户管理操作包括创建用户、删除用户、修改密码、修改用户名、锁定用户、用户授权和撤销授权等

用户操作	MySQL 模式	Oracle 模式
创建用户	CREATE USER user_name [IDENTIFIED BY 'password'];	CREATE USER user_name IDENTIFIED BY password;
删除用户	DROP USER user_name;	DROP USER user_name CASCADE;
修改用户密码	ALTER USER user_name IDENTIFIED BY 'password';	ALTER USER user_name IDENTIFIED BY password;
锁定和解锁用户	ALTER USER user_name ACCOUNT {LOCK | UNLOCK};	ALTER USER user_name ACCOUNT {LOCK | UNLOCK};

也可以使用 OCP 云平台对租户的用户和权限进行管理

MySQL 模式身份鉴别

OceanBase 数据库 MySQL 模式中，用户默认具有连接数据库的权限。一个 user 由 user_name 和 host 共同体组成，数据库的身份鉴别包含密码验证和连接 IP 验证两部分。

创建 user 时默认的 host 为 %，即允许该用户从任意 IP 连接数据库

创建 user 时指定了 host 时，user 必须从 host 发起数据库连接

- **示例：** 创建 3 个具有相同用户名的用户。

```
create user 'u1'@'%' identified by '*******';
create user 'u1'@'10.xxx.xxx.1' identified by '*******';
create user 'u1'@'10.xxx.xxx.2' identified by '*******';
```

- %：表示允许任何客户端 IP 连接该租户。
- 10.xxx.xxx.1：表示只允许 10.xxx.xxx.1 这个 IP 连接该租户。
- 10.xxx.xxx.2：表示只允许 10.xxx.xxx.2 这个 IP 连接该租户。

MySQL 模式权限管理

OceanBase 数据库 MySQL 模式的租户提供了与 MySQL 数据库兼容的权限管理。

MySQL 模式的权限根据权限的类型和授予权限的范围分了 3 个级别

管理权限：可以影响整个租户的权限，例如：修改系统设置、访问所有的表等权限

数据库权限：可以影响某个特定数据库下所有对象的权限，例如：在对应数据库下创建删除表，访问表等权限

对象权限：可以影响某个特定对象的权限，例如：访问一个特定的表、视图或索引的权限

授权命令：GRANT {priv_type [, priv_type...]} ON obj_level TO {user [, user...]} [WITH GRANT OPTION]

priv_type：指定授予的权限类型，可以同时授予多个权限

obj_level：指定授予权限的对象层级，*.* 代表所有对象

WITH GRANT OPTION：指定权限是否允许转授，取消授权时自动级联

Oracle 模式权限管理

OceanBase 数据库 Oracle 模式的租户提供了与 Oracle 数据库兼容的权限策略与角色策略。

Oracle 模式的权限主要有以下两类：

对象权限：对特定对象的操作权限，例如：某个表对象的 Alter、Select、Update 等权限

系统权限：允许用户执行在一个 Schema 或者任何 Schema 上进行特定的数据库操作的权限

授权命令：

GRANT {priv_type [, priv_type...]} ON obj_level TO {user [, user...]} [WITH GRANT OPTION]

GRANT {priv_type [, priv_type...]} TO {user [, user...]} [WITH ADMIN OPTION]

priv_type：指定授予的权限类型或角色，可以同时授予多个权限

obj_level：指定授予权限的对象层级，*.* 代表所有对象

WITH GRANT OPTION：指定授予的对象权限是否允许转授，取消授权时自动级联

WITH ADMIN OPTION：指定授予的系统权限是否允许转授，取消授权时不自动级联

Oracle 模式角色管理

为了方便权限的管理，OceanBase 数据库 Oracle 模式设定了角色。角色是一组系统权限、对象权限的组合，角色中也可以包含其他角色。可以把角色授予用户，用户就拥有了角色里面的所有权限。

角色（ROLE）：Oracle 模式可以使用 ROLE 来把一组权限授予用户。系统在创建新租户时内置了以下默认角色

CONNECT：该角色提供了 CREATE SESSION 系统权限。用户必须要有 CREATE SESSION 的权限才可以连接数据库。

RESOURCE：该角色拥有创建对象所需的系统权限，比如 CREATE TABLE、CREATE PROCEDURE 等。

DBA：该角色拥有数据库管理员所需的系统权限，比如 DELETE ANY TABLE、GRANT ANY PRIVILEGE 等

PUBLIC：所有用户的默认角色，默认未包含任何授权。如果 PUBLIC 角色被授予了权限，则所有用户都立刻拥有该权限

创建角色：CREATE ROLE role_name [NOT IDENTIFIED | IDENTIFIED BY password] ;

NOT IDENTIFIED：启用新建的角色时不使用任何验证方法

IDENTIFIED BY password：用于设置启用角色的密码。使用该子句后，被授予该角色的用户必须指定密码才能使用 SETROLE 语句启用该角色

如果 NOT IDENTIFIED 和 IDENTIFIED BY password 均未指定，则新建的角色默认为 NOT IDENTIFIED

网络访问控制

OceanBase 数据库提供白名单策略，实现网络安全访问控制。租户白名单功能用来设置允许哪些客户端访问当前租户（修改全局变量不影响已创建的 Session）

ob_tcp_invited_nodes：全局变量，是租户全局的白名单限制参数。可以在创建租户的时候指定，也可以在创建租户后通过设置全局变量的方式修改

默认值为 127.0.0.1,::1，表示仅允许本机的 IP 连接该租户。 如果值为 %，表示所有 IP 均可访问

该变量支持列表形式取值，列表值之间使用英文逗号（,）分隔，列表值支持以下赋值

IP 地址，例如：192.168.1.1。匹配时采用等值匹配，即用户 Client IP 等于该 IP 值时，才算匹配

包含百分号（%）或下划线（_）的 IP 地址，例如：192.168.1.% 或 192.168.1._。匹配时采用模糊匹配，即类似 LIKE 语法

IP/NETMASK 地址，例如：192.168.1.0/24 或者 192.168.1.0/255.255.255.0。匹配时采用掩码匹配，只有满足 Client_IP &NetMask == IP 才算匹配成功，类似于 MySQL 的掩码匹配

用户登录时，OBServer 会将用户的 IP 地址与 ob_tcp_invited_nodes 进行匹配，如果匹配失败则拒绝用户登录

```
SET GLOBAL ob_tcp_invited_nodes='10.10.10.%';
SHOW GLOBAL VARIABLES LIKE 'ob_tcp_invited_nodes';
```

容灾管理

回收站与闪回查询

回收站

回收站中存储被删除的数据库和表，用于误删除场景下的数据恢复。进入回收站的对象仍然占据着物理空间，但不能被用户直接访问。

可以进入回收站的对象

租户	Truncate 表	删除的索引	删除的表	删除的数据库	删除的租户
SYS 租户	x	√	√	√	√
MySQL 租户	x	√	√	√	x
Oracle 租户	x	√	√	x	x

删除表时，表上的索引会跟随主表一起进入回收站；单纯删除索引不会进入回收站

删除租户时，被删除的租户进入 SYS 租户的回收站，可以在 SYS 租户中将租户恢复出

来

查看回收站中的内容： SHOW RECYCLEBIN

管理回收站

回收站的开启与关闭由系统变量控制，可以为不同的 Session 设置不同的回收站开关。进入回收站中的对象可以恢复，也可以彻底清理。

开启与关闭回收站 SET {GLOBAL|SESSION} recyclebin = {on|off};

GLOBAL：为租户全局打开回收站，新建的 Session 中执行 DROP 操作时，被删除的对象进入租户回收站

Session：为当前 Session 打开回收站，当前 Session 中执行 DROP 操作时，被删除的对象进入租户回收站

SYS 租户中打开回收站时，被删除的租户会进入回收站

手动清理回收站中的内容 PURGE { RECYCLEBIN | [{TENANT | DATABASE | TABLE | INDEX} object_name] };

PURGE RECYCLEBIN 会清理租户回收站中的所有对象

清理回收站对象时，会同时清理被清理对象的子对象，对象的从属关系为 Database > Table > Index

自动清理回收站中的内容：集群级配置项 recyclebin_object_expire_time 设置回收站中对象的自动清理周期

默认值为 0s，表示关闭自动 Purge 回收站功能

值为非 0s 时，表示回收站的对象可以存放的时间，超出时间将被自动清理

恢复回收站中的对象

被删除的对象进入回收站后，可以使用 FLASHBACK 语句恢复回收站对象。（注意：PURGE 或者 FLASHBACK TABLE、INDEX 时，要在对象所属的 Database 下执行）

FLASHBACK 命令 FLASHBACK {TENANT | DATABASE | TABLE} object_name TO BEFORE DROP [RENAME TO new_object_name];

object_name：指定要恢复的数据库对象在回收站中的名称。恢复表会连通索引一起恢复。

new_object_name：恢复时将数据库对象重命名。如果不指定 RENAME 选项，则使用删除前对象的原始名字。

闪回查询

OceanBase 数据库提供了记录级别的闪回查询（Flashback Query）功能，该功能允许用户获取某个历史版本的数据

undo_retention：用于设置系统应保留的多版本数据范围，在转储时控制多版本数据的回收

租户级配置项，默认值为 1800，单位为秒

闪回查询可查询当前时间 T 到 T - undo_retention 时间范围内的任意多版本数据

闪回查询：通过在查询中指定版本号来获取历史版本的数据

Oracle 模式：使用 AS OF SCN 和 AS OF TIMESTAMP 两种语法来闪回查询

MySQL 模式：使用 AS OF SNAPSHOT 语法来闪回查询

OceanBase 数据库中，SCN 和 SNAPSHOT 都是指事务版本号，本质上还是时间戳，可以使用转换函数来相互转换

SCN_TO_TIMESTAMP：将 SCN 转换为时间戳

TIMESTAMP_TO_SCN：将时间戳转换为 SCN

示例

MySQL 模式: SELECT * FROM tb1 AS OF SNAPSHOT TIMESTAMP_TO_SCN('2024-08-13 16:20:00');

Oracle 模式： SELECT * FROM tb1 AS OF TIMESTAMP TO_TIMESTAMP('2024-08-13 16:20:00','YYYY-MM-DD HH24:MI:SS');

Oracle 模式：SELECT * FROM tb1 AS OF SCN TIMESTAMP_TO_SCN('2024-08-13 16:20:00');

物理备份与恢复

物理备份与逻辑备份

物理备份恢复是 OceanBase 数据库高可用特性的核心组件，主要用于保障数据的安全，包括预防存储介质损坏和用户的错误操作等。如果存储介质损坏或者用户误操作而导致了数据丢失，可以通过恢复的方式恢复用户的数据。

物理备份： 直接复制数据库的物理文件，包括数据文件、日志文件等

在 OceanBase 中，物理备份包括数据备份和日志归档：数据备份包含存储层的基线和转储数据，日志归档包含事务层生成的 Clog

OceanBase 数据库支持租户级别的物理备份

物理备份适用于需要快速恢复和高可靠性的场景，特别是在数据量较大时

逻辑备份：导出数据库的逻辑结构和数据，不涉及物理文件

逻辑备份通常以 SQL 语句或特定格式（如 CSV）导出数据

逻辑备份可以通过工具、SQL 语句来导出特定的数据库、表、分区的数据

逻辑备份适用于需要灵活性和可移植性的场景，特别是在数据量较小或需要跨平台迁移时

备份介质

在 V4.2.1 BP7 版本之前，主要支持 NFS、阿里云 OSS、腾讯云 COS 三种备份介质

V4.2.1 BP7 版本开始，还支持 AWS S3 以及兼容 S3 协议的对象存储（例如：华为 OBS、Google GCS）

日志归档架构

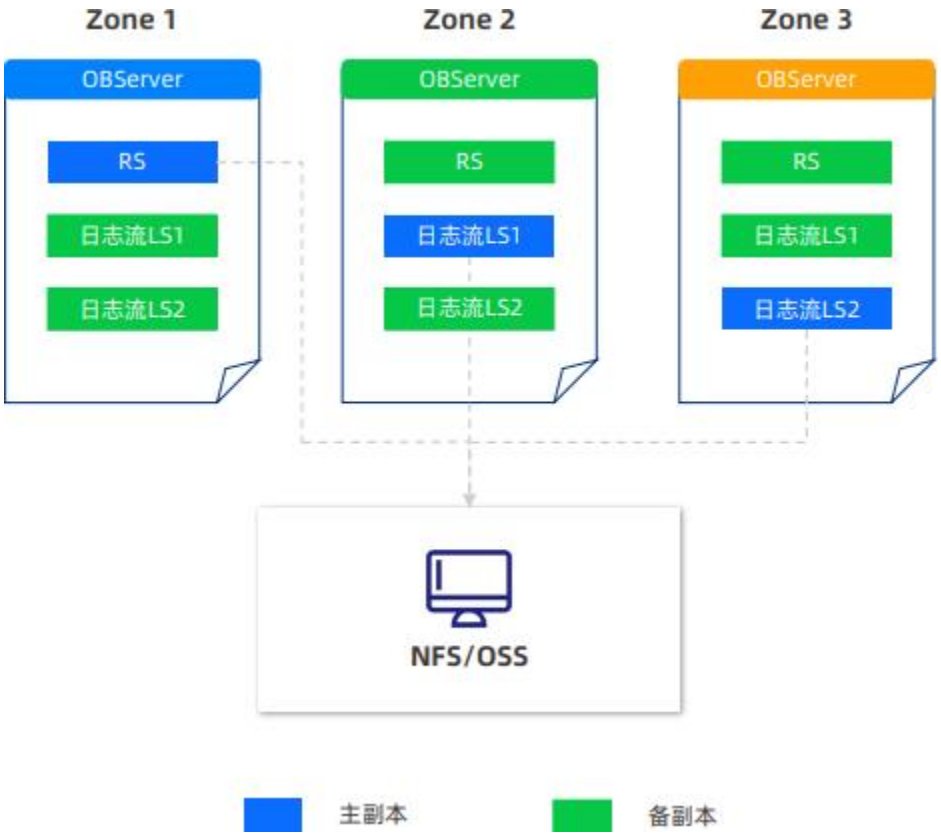
日志归档是指日志数据的自动归档功能，OBServer 节点会持续地将日志数据归档到指定的备份路径。

日志归档就是将数据库产生的事务日志存储到其他介质上，从而可以更长时间地保存数据库的所有 Redo Log 记录

OceanBase 是一个多租户数据库，每个租户可以独立决定是否开启日志归档功能，各租户之间的日志归档相互不影响

归档日志由日志流 Leader 负责，每个日志仅归档一份

归档过程中，由 RS Leader 负责根据每个日志流的归档进度，计算出当前租户的归档进度



数据备份架构

数据备份是指 OceanBase 数据库定期地将存储在磁盘上的数据文件复制到指定的备份路径。

数据备份将数据库转储、合并生成的 SSTable 中的数据存储到其他介质上

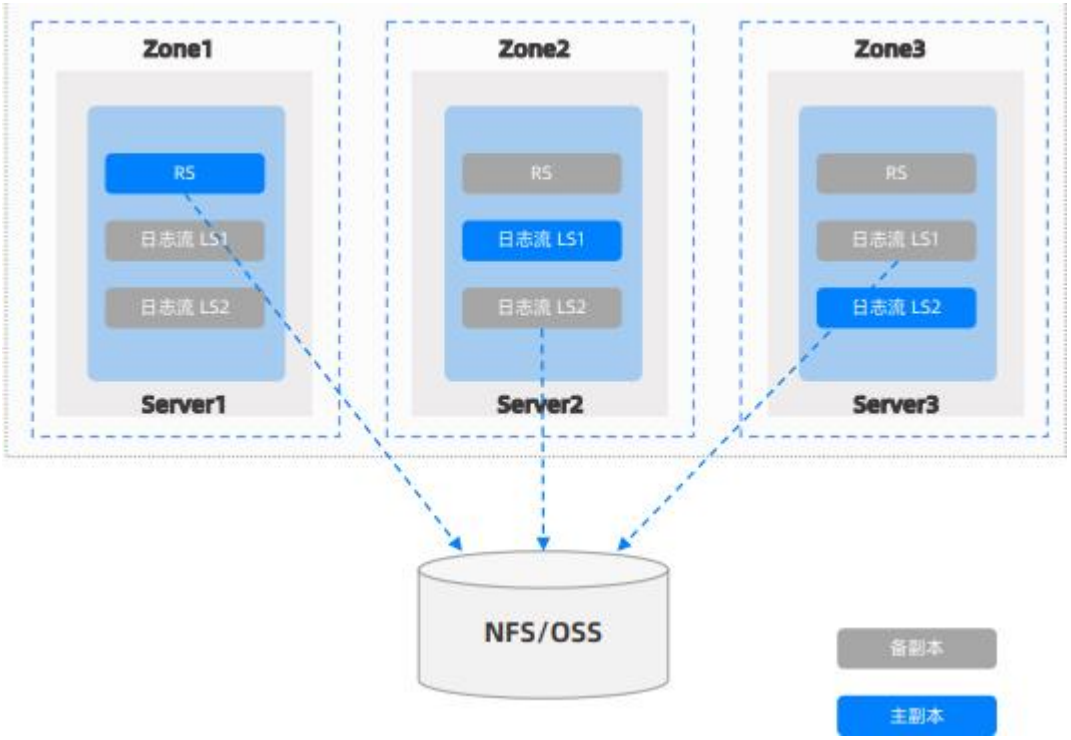
备份方式分为全量备份与增量备份：

全量备份是指备份所有的宏块

增量备份是指备份上一次备份以后新增和修改过的宏块

每个租户可以独立决定是否开启数据备份功能，各租户之间的数据备份相互不影响。不同租户不能配置相同的备份目录

数据备份优先选择 Follower 副本进行备份



物理备份步骤 1：设置日志归档目的地

执行物理备份前需要先开启日志归档，开启归档前需要先设置归档的目的地和策略

设置日志归档目的地 ALTER SYSTEM SET LOG_ARCHIVE_DEST='LOCATION=archive_path [BINDING=archive_mode][PIECE_SWITCH_INTERVAL=piece_switch_interval]' [TENANT = tenant_name];

LOCATION：归档的目的地（路径和文件目录），支持 NFS、OSS、COS、AWS S3 等存储介质

BINDING（可选）：设置归档和业务的优先模式，默认为 Optional 模式

Optional：表示以用户业务优先。在该模式下，日志回收不会等待归档完成，可能会发生归档断流的问题

Mandatory：表示以归档优先。在该模式下，日志回收需要等待归档完成，可能会导致日志盘满的问题

PIECE_SWITCH_INTERVAL （可选）：配置 piece 的切换周期，取值范围为 [1d, 7d]。如果不设置，默认为 1d

OceanBase 数据库按照 Piece 来组织和管理归档日志数据，一个 Piece 是某个租户连续一段时间内完备日志的集合

TENANT：在 SYS 租户中执行命令时，需要指定开启归档的用户租户名；在用户租户中执行命令时，不需要指定租户名

查看归档目的地

可以通过系统视图 CDB_OB_ARCHIVE_DEST（系统租户）或 DBA_OB_ARCHIVE_DEST（用户租户）来查看归档目的地及相关策略的设置

物理备份步骤 2：开启日志归档

开启/关闭日志归档 ALTER SYSTEM {ARCHIVELOG | NOARCHIVELOG} [TENANT = tenant_name]

ARCHIVELOG 为开启归档，NOARCHIVELOG 为关闭归档

TENANT：在 SYS 租户中执行命令时，需要指定开启归档的用户租户名；在用户租户中执行命令时，不需要指定租户名

查看归档进度 SELECT ROUND_ID,DEST_ID,PATH,STATUS,START_SCN_DISPLAY,CHECKPOINT_SCN_DISPLAY FROMDBA_OB_ARCHIVELOG

在用户租户中使用 DBA_OB_ARCHIVELOG 查看本租户的归档进度，在 SYS 租户中使用 CDB_OB 视图查看所有租户的信息

以上示例中，租户是首次开启归档（ROUND_ID=1）归档，且正在执行归档（STATUS=DOING），对应的存储介质是 NFS（PATH=file），归档的起点如 START_SCN_DISPLAY 所示，归档的最新点位如 CHECKPOINT_SCN_DISPLAY 所示

物理备份步骤 3：设置数据备份目的地

可以单独开启日志归档，并在需要的时候执行数据备份。OceanBase V4 可以为日志归档和数据备份设置不同的目的地（或目录）。

设置数据备份目的地 ALTER SYSTEM SET DATA_BACKUP_DEST='data_backup_path' [TENANT = tenant_name];

TENANT：在 SYS 租户中执行命令时，需要指定开启归档的用户租户名；在用户租户中执行命令时，不需要指定租户名

查看数据备份目的地 SELECT * FROM DBA_OB_BACKUP_PARAMETER;

使用 DBA_OB_BACKUP_PARAMETER 查看数据备份目的地，也可以使用 DBA_OB_BACKUP_STORAGE_INFO 查看日志归档、数据备份与秘钥文件的目的地

系统租户使用 CDB_OB 视图查看所有租户的信息，用户租户使用 DBA_OB 视图查看本租户的信息

物理备份步骤 4：开启数据备份

可以单独开启日志归档，并在需要的时候执行数据备份。数据备份可以是全量备份，也可以是增量备份，执行增量备份前至少要有一份全量备份。

发起全量/增量备份：前提条件是日志归档状态正常

全量备份：ALTER SYSTEM BACKUP {DATABASE | TENANT = tenant_name} [PLUS ARCHIVELOG];

增量备份：ALTER SYSTEM BACKUP INCREMENTAL {DATABASE | TENANT = tenant_name};

DATABASE：在 SYS 租户中执行命令时，表示对所有租户发起全量备份；在用户租户中执行命令时，表示对该租户发起全量备份

TENANT：在 SYS 租户中执行命令时，表示对指定的用户租户发起全量备份；在用户租户中执行命令时，不需要指定租户名

PLUS ARCHIVELOG（可选）：额外将日志一起备份到数据备份目录，形成具备恢复能力的数据集，可以不依赖归档日志进行恢复

查看备份任务：SELECT JOB_ID,BACKUP_TYPE,PLUS_ARCHIVELOG,START_TIMESTAMP,END_TIMESTAMP,STATUS,RESULT FROM DBA_OB_BACKUP_JOB_HISTORY;

使用 DBA_OB_BACKUP_JOBS 查看正在执行的备份任务，使用 DBA_OB_BACKUP_JOB_HISTORY 查看已经完成的备份任务

系统租户使用 CDB_OB 视图查看所有租户的信息，用户租户使用 DBA_OB 视图查看本租户的信息

物理恢复架构

物理恢复本质上是基于物理备份恢复租户的过程。OceanBase 数据库可以将物理备份恢复到原有的集群，也可以恢复会到不同的集群。

物理恢复的粒度

租户级恢复：基于已有数据的备份重建新租户的过程，恢复的内容包括租户系统表和用户表

表级恢复：从备份数据中将用户指定的表恢复到一个已存在的租户

表级恢复需要借助“辅助租户”来完成，实际上先将表恢复到 “辅助租户”，再从“辅助租户”中将表复制到目标租户

物理恢复的位点

完全恢复：是指使用数据备份恢复数据以后，重做所有的 clog 日志，让租户的数据能够恢复到最新的一致位点

不完全恢复：是指使用数据备份恢复以后，只重做部分 clog 日志，让租户的数据恢复到指定的一致位点

物理恢复租户

物理恢复租户的过程首先要新建目标租户，然后将物理备份的数据恢复到目标租户中，最后通过重做 clog 日志来恢复数据到指定的一致点

恢复租户：ALTER SYSTEM RESTORE dest_tenant_name FROM uri[UNTIL {TIME='timestamp' | SCN=scn}] WITH 'restore_option';

FROM：指定备份时设置的数据备份路径 backup_data_dest 与日志归档路径 log_archive_dest，路径之间使用英文逗号（,）分隔

UNTIL：指定恢复的终点，恢复到该位点位置，且包含该位点。如果不指定 UNTIL 子句，则默认恢复到最新的位点

WITH：指定目标租户的 pool_list、locality、primary_zone 等属性，以及恢复任务的 concurrency

查看恢复进度

使用 DBA_OB_RESTORE_PROGRESS 查看正在执行的恢复任务，使用 DBA_OB_RESTORE_HISTORY 查看已经完成的恢复任务

系统租户使用 CDB_OB 视图查看所有租户的信息，用户租户使用 DBA_OB 视图查看本租户的信息

物理恢复租户的后续操作

物理恢复出来的租户角色默认为备租户，需要按照使用场景进行后续操作

基于备份创建备租户：持续回放源租户的日志。ALTER SYSTEM RECOVER STANDBY[TENANT [=] tenant_name] UNTIL { TIME='timestamp' | SCN=scn_no | UNLIMITED };

TENANT：需要恢复的备租户的名字

UNTIL：用于指定的恢复终点（TIME 或者 SCN）。UNLIMITED 表示开启主备库的持续同步

物理恢复：将恢复后的租户转化为可读写 ALTER SYSTEM ACTIVATE STANDBY [TENANT = tenant_name];

在 SYS 租户执行该命令时，需要指定租户名

升级租户：将恢复出来的备租户转为主租户后，如果是从低版本的备份数据恢复到高版本的集群中，还需要对该租户进行升级 ALTER SYSTEM RUN UPGRADE JOB "UPGRADE_ALL" TENANT = tenant_name;

只有当恢复出来的租户转化为主租户时，才可能需要执行升级租户的操作

租户主备库

物理备库是主库的准实时热备份，可以在当主库不可用时接管服务并提供无损切换（RPO = 0）和有损切换（RPO > 0）两种容灾能力，最大限度降低服务停机时间，减少可能带来的数据损失。

V4.1.0 版本开始，物理备库的产品形态从集群级变更为租户级，即主或备的角色信息属于租户，分为主租户和备租户

主租户是用户创建的业务租户，支持完整的数据库服务能力

备租户则仅提供容灾和只读服务的能力

一个主租户和若干备租户共同组成了租户级的物理备库高可用解决方案

物理备库主要通过日志传输服务和日志存储服务来完成日志的传输及存储，并通过日志回放服务来保证主备租户数据的一致，其中

日志传输服务在主租户和备租户之间实时同步 Redo 日志。当前，OceanBase 数据库物理备库仅提供异步同步模式

日志存储服务为物理备库提供高可用、高可靠的日志存储和读写能力

备租户写入日志存储服务的日志会通过日志回放服务实时或延后地应用到内存的 MemStore 之中，以保证主租户和备租户的数据完全一致

主备库部署方案

一个主租户和对应的备租户可以部署在不同的集群中，也可以部署在同一个 OceanBase 集群中。相应的，同一个 OceanBase 集群中既可以包含主租户，也可以包含备租户，也可以同时包含主备租户。

纯主+纯备：该方案中，每个 OceanBase 集群包含的业务租户或者都是主租户，或者都是备租户

主备混合（多集群）：该方案中，每个集群中既有主租户又有备租户，同时也允许集

群中的租户只有主租户没有备租户

主备混合（单集群）：该方案中，用户只有一个集群，集群中的租户互为主备

物理备库使用限制

主备库功能使用要求与限制

功能与限制	具体描述
备租户个数	无限制
资源配置	不要求同构。建议主租户和备租户使用相同的资源规格。
租户名	不要求相同。
配置项	主租户与备租户的配置项相互独立，不会物理同步。如果修改了主租户的配置项，需要评估是否需要修改备租户的相同配置项。
系统变量	主租户和备租户的系统变量物理同步，如果在主租户上修改了系统变量，系统会同步修改备租户的相同系统变量。
用户及用户密码	仅支持在主租户上创建用户和修改用户密码，更新后信息会同步给备租户。
读写操作	备租户支持读操作，不支持写操作。
转储与合并	主租户和备租户的转储相对独立。 备租户从主租户同步合并信息，不支持独立合并。

主备库日志传输服务

物理备库通过日志传输服务在主租户和备租户之间实时同步 Redo 日志。备租户既可以从主租户同步日志，也可以从另外一个备租户同步日志。

基于日志归档的物理备库

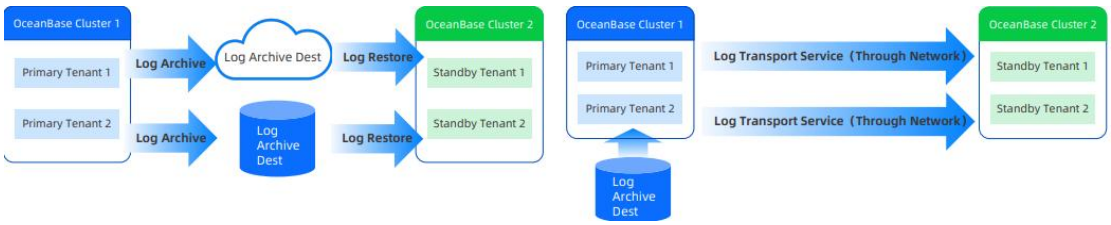
物理备库的 Redo 日志来源于主租户或其他备租户的日志归档，类似于 Oracle 数据库的 Far Sync，备租户仅与日志归档交互，而不会和上游的主租户或备租户有任何其他形式的交互

在该部署模式下，备租户与上游租户不需要网络联通，但其同步性能和可用性会受到日志归档介质的影响

基于网络的物理备库（V4.2+）

备租户直接通过网络连接主租户或其他备租户读取日志，类似于 MySQL 数据库的 Replication

在该部署模式下，备租户和主租户的网络需要联通。备租户从主租户读取的日志，既可以是主租户的在线日志，也可以是主租户的归档日志，两种日志来源支持自动切换



创建备租户

创建空的备租户意味着仅靠主租户的日志来创建备租户，不依赖除日志外的其他基线或转储数据。这个方式仅适用于主租户刚刚新建或者主租户所在集群或日志归档介质上保存有该主租户自创建后完整日志的场景。

步骤 1：创建同步专用用户

备租户在连接主租户时，需要访问主租户的部分系统视图，因此需要有一个访问视图专用用户具备这部分系统视图的查询权限

使用 OCP 云平台创建备租户时，OCP 会自动在主租户中创建 STANDYRO 用户

手动创建备租户时，需要手动创建相应的用户并授予权限，创建用户及授权的具体步骤参见官网文档

步骤 2：创建备租户

```
CREATE STANDBY TENANT standby_tenant_name
LOG_RESTORE_SOURCE = 'SERVICE=ip_list USER=username@tenant_name
PASSWORD=password'
RESOURCE_POOL_LIST=('pool_for_standby')
LOCALITY [=] 'locality_description'
PRIMARY_ZONE [=] primary_zone_name;
```

LOG_RESTORE_SOURCE：备租户的日志恢复源

SERVICE ： 主 租 户 所 在 的 OBServer 节 点 的 IP 列 表 ， 比 如 SERVICE=10.xxx.xxx.1;10.xxx.xxx.2:2881;10.xxx.xxx.3:2881

USER/ PASSWORD：用于访问主租户的专用用户和密码

从物理备份 Restore 一个备租户

使用备份恢复（带日志）功能创建备租户的方式为一种通用方式，该方式主要是从数据备份和日志归档中恢复出备租户，并且既可以从主租户的数据备份和日志归档中恢复，也可以从已存在的备租户的数据备份和日志归档中恢复

RESTORE 租户：从源租户的物理备份恢复出一个备租户

```
ALTER SYSTEM RESTORE dest_tenant_name FROM uri [ UNTIL {TIME='timestamp' |
SCN=scn} ]WITH
'pool_list=pool_name[&locality=locality][&primary_zone=zone_name][&concurrency=int_num]'
```

FROM：需要分别指定主租户或源端的备租户的数据备份路径和归档路径

UNTIL：用于指定的恢复终点（TIME 或者 SCN）。如果不指定 UNTIL 子句，则默认恢复到最新的位点

WITH： 备租户的 pool_list、locality、primary_zone 属性，以及恢复租户的并发线程数（concurrency），pool_list 是必选项

RECOVER 租户：RESTORE 命令创建的备租户，默认不会与源租户持续同步，需要执行 RECOVER 命令开启持续同步模式

```
ALTER SYSTEM RECOVER STANDBY[TENANT [=] tenant_name] UNTIL { TIME='timestamp' |
SCN=scn_no | UNLIMITED };
```

TENANT：需要恢复的备租户的名字

UNTIL：用于指定的恢复终点（TIME 或者 SCN）。UNLIMITED 表示开启主备库的持续

同步

修改日志恢复源

无论是基于日志归档的物理备库，还是基于网络的物理备库，都支持动态修改日志恢复源，同时也支持从一种日志恢复源切换为另外一种日志恢复源（修改日志恢复源必须保证日志恢复源租户与创建备租户时所指定的源租户一致，否则将导致备租户数据错误且无法恢复）。

设置日志恢复源 ALTER SYSTEM SET LOG_RESTORE_SOURCE ={'SERVICE=$ip_list USER=$user_name@$tenant_name PASSWORD=$password' |'LOCATION=$archive_path'} [TENANT = $tenant_name];

SERVICE：设置日志恢复源为网络直连模式，ip_list 为主租户所在的 OBServer 节点的 IP 列表

LOCATION：设置日志恢复源为日志归档模式，archive_path 为源租户的归档路径

TENANT：在 SYS 租户中执行命令时，需要指定备租户名；在用户租户中执行命令时，不需要指定租户名

可以通过视图 CDB_OB_LOG_RESTORE_SOURCE（SYS 租户） 或 DBA_OB_LOG_RESTORE_SOURCE 查看日志恢复源的设置

ALTER SYSTEM SET LOG_RESTORE_SOURCE='LOCATION=file:///data/1/sh_archive' TENANT =mysql_back;

RECOVERY_UNTIL_SCN：表示租户可恢复到的终点。4611686018427387903 是 MAX SCN，则表示持续同步

主备库角色切换

OceanBase 支持通过 SWITCHOVER 和 FAILOVER 操作动态改变主备租户角色，分别应对日常切换与故障切换的场景

SWITCHOVER（日常切换：是用户计划内的角色交换。SWITCHOVER 操作后）

主租户会与对应的备租户会交换租户角色；

整个过程数据不丢失，RPO = 0；

执行时间一般为秒级

FAILOVER（故障切换：是主租户出现故障时所做切换。FAILOVER 操作后）

原备租户升级为主租户；

会产生百毫秒级别的数据损失；

执行时间一般为秒级

SWITCHOVER：基于网络的物理备库

在基于网络的物理备库场景中，SWITCHOVER 主要包括三个阶段：主租户切换为备租户 --> 备租户切换为主租户 --> 为新备租户设置日志恢复源(执行 SWITCHOVER 操作时，必须先将主租户切换为备租户，再将备租户切换为主租户，以避免出现双主租户的问题)

步骤 1：将主租户切换为备租户

验证 SWITCHOVER 命令是否可以执行成功 ALTER SYSTEM SWITCHOVER TO STANDBY [TENANT = tenant_name] VERIFY;

TENANT：在 SYS 租户中执行用户租户的 SWITCHOVER 操作时，需要指定租户名

VERIFY：执行 SWITCHOVER 前的验证操作。SWITCHOVER 要求主库与备库状态正常，主备库之间同步状态正常

验证通过后，将主租户 SWITCHOVER 为备租户 ALTER SYSTEM SWITCHOVER TO STANDBY [TENANT = tenant_name];

查询 DBA_OB_TENANTS 视图，确认主租户是否已切换为备租户 SELECT TENANT_NAME, TENANT_TYPE, TENANT_ROLE, SWITCHOVER_STATUS FROMDBA_OB_TENANTS;

SWITCHOVER_STATUS：NORMAL 表示 SWITCHOVER 已完成

步骤 2：将备租户切换为主租户，其执行过程与步骤 1 类似

验证 SWITCHOVER 命令是否可以执行成功 ALTER SYSTEM SWITCHOVER TO PRIMARY [TENANT = tenant_name] VERIFY;

验证通过后，将主租户 SWITCHOVER 为备租户 ALTER SYSTEM SWITCHOVER TO PRIMARY [TENANT = tenant_name];

查询 DBA_OB_TENANTS 视图，确认主租户是否已切换为备租户 SELECT TENANT_NAME, TENANT_TYPE, TENANT_ROLE, SWITCHOVER_STATUS FROM DBA_OB_TENANTS;

步骤 3：设置当前的备租户（mysql）的日志恢复源为当前的主租户（mysql_back）

ALTER SYSTEM SET LOG_RESTORE_SOURCE ='SERVICE=$ip_list USER=$user_name@$tenant_namePASSWORD=$password' TENANT = tenant_name;

SWITCHOVER：基于归档的物理备库

在基于日志归档的物理备库场景中，SWITCHOVER 操作前需要先在备租户中开启日志归档。（执行 SWITCHOVER 操作时，必须先将主租户切换为备租户，再将备租户切换为主租户，以避免出现双主租户的问题）

步骤 1：备租户开启日志归档模式

在基于日志归档的物理备库场景中，SWITHOVER 操作执行成功后，由于原主租户后续需要通过持续读取新主租户（原备租户）的归档日志来同步新主租户（原备租户）上的所有修改操作，因此，要求新主租户（原备租户）开启日志归档模式

具体操作参考《物理备份与恢复》章节的内容

步骤 2：将主租户切换为备租户

具体操作与基于网络的物理备库的 SWITCHOVER 操作相同

步骤 3：将备租户切换为主租户

具体操作与基于网络的物理备库的 SWITCHOVER 操作相同

步骤 4：设置当前备租户的日志恢复源为当前主租户的日志归档

ALTER SYSTEM SET LOG_RESTORE_SOURCE ='LOCATION=archive_path' [TENANT =

tenant_name];

FAILOVER

当主租户不可用时，可以将备租户 FAILOVER 为主租户继续提供服务。FAILOVER 完成后需要整个租户达到数据一致的状态，因此 FAILOVER 操作会将租户统一恢复到所有日志流的最小同步位点，也意味着 FAILOVER 不能保证数据零丢失。（注意：FAILOVER 将主备租户解耦，因此 Failover 操作也可以用于基于主租户某一个时间点的快照数据的业务验证场景。）

将备租户 FAILOVER 为主租户的步骤

验证 FAILOVER 命令是否可以执行成功：ALTER SYSTEM ACTIVATE STANDBY [TENANT = tenant_name] VERIFY;

TENANT：在 SYS 租户中执行用户租户的 FAILOVER 操作时，需要指定租户名

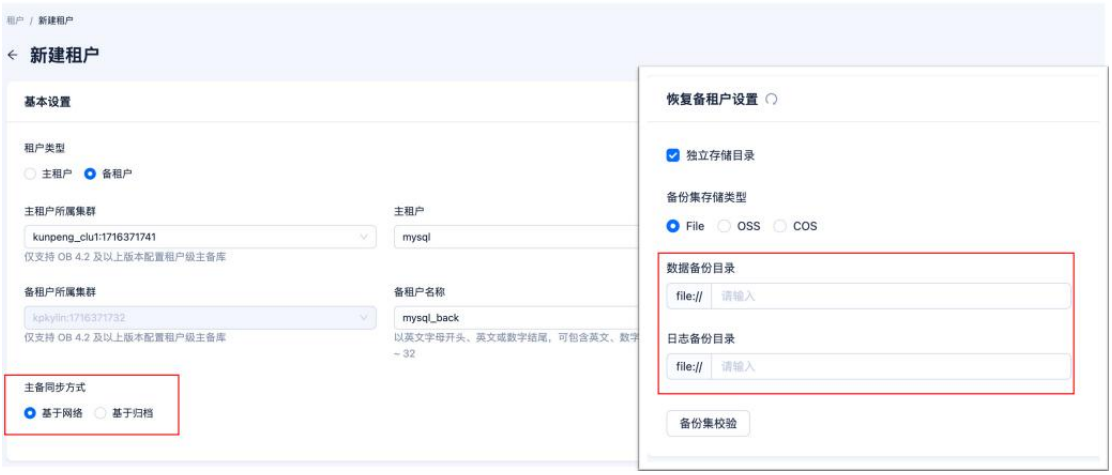
VERIFY：执行 FAILOVER 前的验证操作。 FAILOVER 操作不检查主租户的状态，要求备库状态正常

验证通过后，将备租户 FAILOVER 为主租户：ALTER SYSTEM ACTIVATE STANDBY [TENANT = tenant_name];

查询 DBA_OB_TENANTS 视图，确认主租户是否已切换为备租户，SELECT TENANT_NAME, TENANT_TYPE, TENANT_ROLE, SWITCHOVER_STATUS FROM DBA_OB_TENANTS;

使用 OCP 创建备租户

- **路径：**集群--新建租户、租户--新建备租户



使用 OCP 执行主备切换

■ **路径：** 租户--日常切换/容灾切换

系统监控

监控体系概览

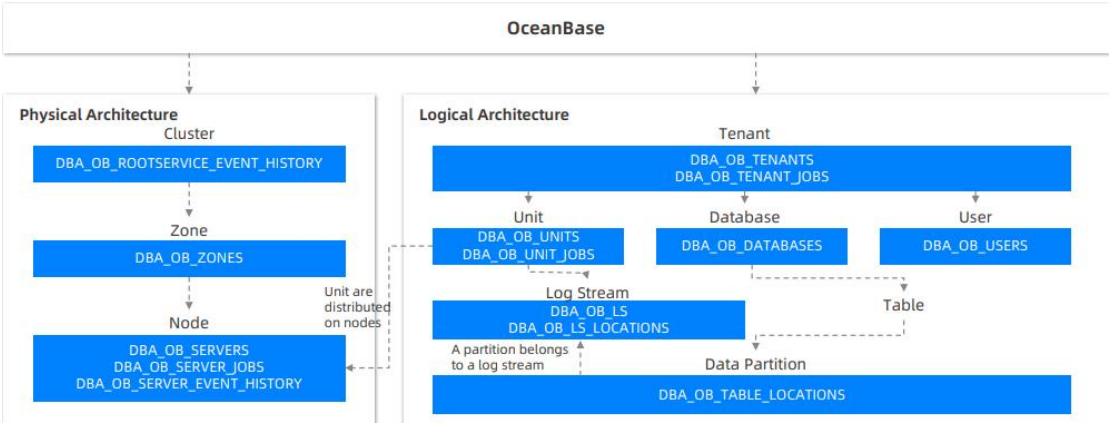
OceanBase 监控体系

OceanBase V4 数据库在内核层面进一步完善了系统监控与诊断体系，在基于系统日志的监控诊断之外，提供了标准化的系统视图、WR 与 ASH，以及全链路追踪等功能，让用户可以便捷地监控、诊断数据库。

系统视图

OceanBase 数据库 V4 有着非常丰富的视图，通过这些视图可以获取 OceanBase 集群各种数据库对象的基本信息和实时状态信息。

丰富的视图展示了 OceanBase 数据库的内部架构，及系统运行的详细状态



日志

OceanBase 的日志分为系统日志、事务日志和存储日志三种（通常，我们说“查询数据库日志”时指的是系统日志 Syslog。系统日志是 OceanBase 最基本的监控诊断途径。）

日志类别	隔离级别	日志路径	内容与用途
系统日志 Syslog	OBServer 级	$work_dir/log	OBServer 进程运行过程中打印的程序日志，用于监控报警和诊断。
事务日志 Clog	日志流级	$data_dir/clog	OceanBase 事务日志，记录事务变更的 Redo Log，用于保证事务性。
存储日志 Slog	租户级	$data_dir/slog	OceanBase 存储层用于记录元数据修改的 Redo Log。 • 创建租户、日志流、Tablet 需要写 Slog，转储等操作时也需要写 Slog。 • OBServer 重启后会回放 Slog，使存储层恢复到重启前的状态。

WR 与 ASH

从 V4.2 版本开始 OceanBase 提供了与 Oracle 数据库相似的 AWR、ASH 功能，加强了数据库系统性能诊断的能力。WR 定期采集系统视图的数据，以便于回溯系统的历史状态。ASH 采样、记录活跃会话的统计信息，可以生成基于等待事件的 TOP 报告，帮助用户排查瞬时异常的问题。

常见性能场景：

数据库整体运行慢，但无明显资源瓶颈和直接方向

确定具体 SQL 执行响应时间突然增加，性能变差

系统资源异常飙高，业务繁忙，SQL 众多混杂，变化频繁

系统表现间歇型卡顿，分钟级或者秒级

夜间跑批性能异常，无人值守，需要问题回溯复盘
租户内多用户运行，整体资源瓶颈，希望识别有侵略性的用户
业务系统遭遇连接风暴，造成资源紧张性能下降，如何确定并发量变化趋势
需要了解整体导致 TOP 等待事件的 会话或者 SQL 信息，明确优化目标

全链路 Trace

分布式系统拥有复杂的调用链路，当链路出现超时问题时，往往需要排查各个链路组件的日志，整个过程耗时费力。OceanBase V4 提供了全链路追踪的能力，能够以可视化的方式展示 SQL 请求在数据库全链路过程中，在不同组件、不同阶段执行的相关信息，助力用户快速精准地排查问题。

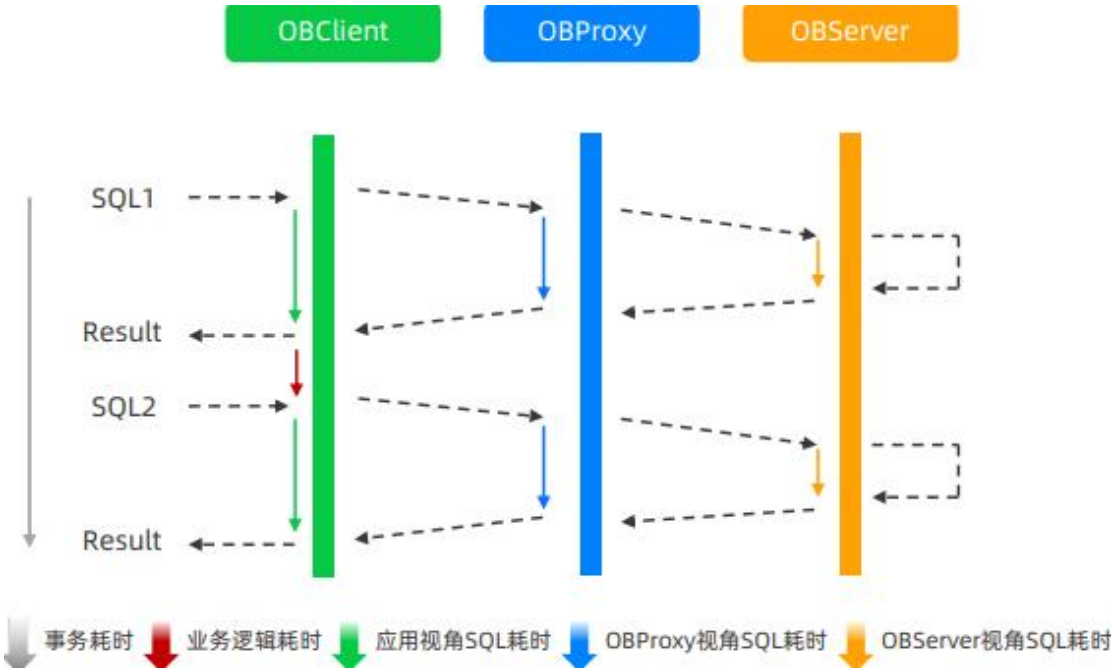
全链路（Full-Link）Trace

全链路顾名思义就是整个链路，从业务端生成 SQL 到 SQL 从数据库执行完返回客户端一整套下来的链路

全链路 Trace 即客户端到 OBProxy 再到 OBServer 全链路各阶段的耗时及各组件诊断相关的 Trace 信息

全链路追踪

OceanBase 基于各个链路的 trace log，一键生成 SQL 的全链路耗时信息，以便用户可以快速定位链路问题



标准化系统视图

自有视图与兼容视图

OceanBase 数据库 V4 对视图进行了规范化定义，MySQL 模式和 Oracle 模式分别有兼容 Oracle 数据库与 MySQL 数据库的系统视图，同时也有 OceanBase 特有的系统视图。

兼容视图

Oracle 租户兼容大部分 Oracle 系统视图：DBA_*，USER_*， ALL_*

MySQL 租户兼容大部分 MySQL 视图：information_schema.*，mysql.*

将部分 Oracle 视图迁移到 MySQL 租户（包括系统租户），以弥补 MySQL 系统视图的不足

自有视图

字典视图：基于实体表的封装，用于显示数据库的元数据信息和系统状态信息

CDB_OB_xxx：系统租户独有，包含所有租户的信息

DBA_OB_xxx：只包含当前租户的信息

性能视图：基于虚拟表的封装，用于显示数据库的性能统计信息和监控信息

GV$OB_xxx：包含所有节点信息，系统租户下可以查到所有租户的信息，用户租户下只能查到当前租户下的信息

V$OB_xxx：仅包含当前节点信息，系统租户下可以查到所有租户的信息，用户租户下只能查到当前租户下的信息

查找系统视图

可以在系统租户中查询系统视图 CDB_OBJECTS 来查看各个租户都分别有哪些系统视图。

查找 MySQL 模式的租户中与 MySQL 数据库兼容的系统视图：SELECT

OWNER,OBJECT_NAME FROM CDB_OBJECTS WHERE CON_ID=1004 AND OBJECT_TYPE='VIEW' AND `GENERATED`='Y' AND OWNER IN('mysql','information_schema') ORDER BY OWNER, OBJECT_NAME;

查找 Oracle 模式的租户中与 Oracle 数据库兼容的系统视图：SELECT OWNER,OBJECT_NAME FROM CDB_OBJECTS WHERE CON_ID=1002 AND OBJECT_TYPE='VIEW' AND `GENERATED`='Y' ANDOWNER='SYS' AND NOT(OBJECT_NAME LIKE 'DBA_OB_%' OR OBJECT_NAME LIKE '%V$OB_%') ORDER BY OWNER, OBJECT_NAME;

查找 MySQL 模式的租户中的 OceanBase 自有系统视图：SELECT OWNER,OBJECT_NAME FROM CDB_OBJECTS WHERE CON_ID=1004 AND OBJECT_TYPE='VIEW' AND `GENERATED`='Y' ANDOWNER='oceanbase' AND (OBJECT_NAME LIKE 'DBA_OB%' OR OBJECT_NAME LIKE '%V$OB_%') ORDER BY OWNER, OBJECT_NAME;

查找 Oracle 模式的租户中的 OceanBase 自有系统视图：SELECT OWNER,OBJECT_NAME FROM CDB_OBJECTS WHERE CON_ID=1002 AND OBJECT_TYPE='VIEW' AND `GENERATED`='Y' ANDOWNER='SYS' AND (OBJECT_NAME LIKE 'DBA_OB%' OR OBJECT_NAME LIKE '%V$OB_%') ORDER BY OWNER, OBJECT_NAME;

数据库对象视图

数据库对象主要依赖 MySQL 或 Oracle 兼容视图，OceanBase 自有视图主要涵盖 OceanBase 特有的数据对象

数据库对象	MySQL 租户	Oracle 租户
所有对象	DBA_OBJECTS	DBA_OBJECTS
DATABASE	DBA_OB_DATABASES / information_schema.SCHEMATA	DBA_OB_DATABASES
TABLE	information_schema.TABLES	DBA_TABLES
VIEW	information_schema.VIEWS	DBA_VIEWS
INDEX	information_schema.STATISTICS	DBA_INDEXES
COLUMN	information_schema.COLUMNS	DBA_TAB_COLUMNS
TABLEGROUP	DBA_OB_TABLEGROUP_TABLES	DBA_OB_TABLEGROUP_TABLES
AUTO_INCREMENT	DBA_OB_AUTO_INCREMENT	不支持自增列
SEQUENCE	DBA_SEQUECE / DBA_OB_SEQUENCE_OBJECTS	DBA_SEQUECE
ROUTINE	information_schema.ROUTINES/PARAMETERS, mysql.proc	DBA_PROCEDURES/DBA_SOURCE/DBA_ARGUMENTS
DBLINK	DBA_DB_LINK	DBA_DB_LINK
USER	mysql.user	DBA_USERS

系统监控视图

OceanBase 数据库的各项监控指标可以通过监控工具（例如 OCP）或直接使用 SQL 查询 OceanBase 数据库的性能视图来获取

系统监控：系统监控包括集群、租户、会话和 SQL 等多个维度，对多样化的监控信息进行汇总展现，以满足不同场景的诊断需求，包括系统监控、故障诊断、SQL 分析等

GV$SYSSTAT：系统监控统计，涵盖网络、请求队列、事务、SQL、缓存、存储、资源配额、日志等所有关键的执行信息

GV$SYSTEM_EVENT：租户级别的等待事件统计，包括各类等待事件的总等待次数、总等待时间、总超时次数和平均等待时间等

GV$SESSION_EVENT：会话级别的等待明细记录，包括一次特定等待事件的等待开始时间、当前状态、剩余时间等

SQL 监控：根据特定 SQL 的多次执行或特定执行中收集的监控信息对系统的性能问题进行诊断

GV$OB_SQL_AUDIT：SQL 审计表，包含每次 SQL 执行的详细统计信息，比如执行状态、等待时间以及执行各阶段耗时等

GV$OB_PLAN_CACHE_PLAN_STAT：SQL Plan 执行统计，以 Plan 的维度统计相同 SQL 的多次执行情况

GV$OB_PLAN_CACHE_PLAN_EXPLAIN：执行计划的详细信息，记录 Plan Cache 中所有缓存的执行计划

GV$SQL_PLAN_MONITOR：SQL 算子级别的执行统计，实时统计 SQL 执行过程中每一个算子的执行耗时

除此之外，OceanBase 还提供了大量的监控视图用来监控系统各个层面的运行状态，比如内存监控、队列监控、事务监控等

日志与错误码

系统日志

OBServer 的程序日志存放在 observer 的安装目录的 log 目录下面，并按照功能和模块进行了分类存放（注意：OceanBase V4.2.3 版本开始系统日志新增 Alert 日志，用于提供格式化的重要日志信息，并可在数据库中以外表的形式查询）

日志类别	文件名	日志路径
启动和运行日志	observer.log observer.log.wf	$work_dir/log
选举日志	election.log	$work_dir/log
Root Service 日志	rootservice.log	$work_dir/log
全链路追踪日志	trace.log	$work_dir/log

一个日志文件的大小为 256 MB，文件写满后会切换到下一个日志文件

当前正在写入的日志文件，其文件名为 $syslog_type.log，比如 observer.log

旧的日志文件，其文件名为 $syslog_type.$timestamp，比如 observer.log.20240820052822459

observer.log.wf 是 observer.log 的子集，其中仅包含 observer 日志中的告警日志和报错日志，跟随 observer.log 日志文件切换

日志级别

OceanBase 数据库将系统日志划分了七个日志级别，用户可以根据需要设置不同的日志级别，来获得相应的监控、诊断信息

日志级别	含义
ERROR	严重错误。用于记录系统的故障信息，且必须进行故障排除，否则系统不可用。
WARN	警告。系统能继续提供服务，但行为可能不符合预期，或可能将有严重错误发生，需进行故障排除。
INFO	提示。用于记录系统运行的当前状态，该信息为正常信息。
EDIAG	Error Diagnosis, 协助故障排查的诊断信息。
WDIAG	**Warning Diagnosis, 协助故障排查的诊断信息。WDIAG 是默认的日志级别。**
TRACE	SQL 语句级调试信息，日志数与 SQL 复杂度相关，与访问数据量无关。
DEBUG	调试信息。用于调试时更详细地了解系统运行状态，包括当前调用的函数名、参数、变量、函数调用返回值等。

日志级别表示了日志信息的详细程度，日志级别越低，打印的日志内容越详细

ERROR 的级别最高，且日志比较特殊，会将打日志时所在的堆栈打印出来

DEBUG 的级别最低，开启 DEBUG 日志将耗费大量资源，不建议使用

observer 总是打印设置的日志级别及其上级别的日志信息。比如设置日志级别为 INFO 时，会打印 INFO、WARN 和 ERROR 日志

设置日志级别

OceanBase 数据库支持设置日志级别，从而获取更精确的目标日志。可以从三个级别设置：系统级别、Session 级别、语句级别。

系统级别：ALTER SYSTEM SET syslog_level='WDIAG';

syslog_level 为集群级配置项，用于设置整个集群的的日志级别，仅支持在系统租户下配置

会话级别：SET GLOBAL ob_log_level='ERROR';

ob_log_level 为系统变量，可以设置租户级别或者 Session 级别的日志级别

语句级别：SELECT /*+log_level('DEBUG')*/ * FROM t;

log_level 为 SQL Hint 参数，用于设置 SQL 级别的日志级别

日志文件管理

除了设置系统日志的级别，还可以对系统日志的写入与回收进行统一的管理

配置项	默认值	说明
enable_syslog_recycle	False	用于设置是否开启回收系统日志。
enable_syslog_wf	True	用于设置是否把 WARN 以上级别的系统日志打印到一个单独的日志文件（日志文件的后缀为 wf）中。
max_syslog_file_count	0	用于设置在回收日志文件之前可以容纳的最大日志文件数量。0 表示不回收。
syslog_io_bandwidth_limit	30MB	用于设置系统日志所能占用的磁盘 IO 带宽上限，超过带宽上限容量的系统日志将被丢弃。取值为 0 表示关闭系统日志。

OceanBase 默认不自动清理日志。一个繁忙的生产系统可能在短时间内产生比较多的日志文件，需要开启日志回收。我们建议根据日志磁盘的空间大小，设置合理的日志回收参数：

enable_syslog_recycle = True

max_syslog_file_count：计算日志磁盘最多可以保存的日志文件数量（一个日志文件 256MB），设置合理的水位线

在问题诊断时，为了防止因为日志限流导致的日志记录丢失，可以适当调大

syslog_io_bandwidth_limit 的阈值

日志格式

系统日志记录在打印时遵循固定的格式，其中会标记打印日志的模块、程序，以及日志内容的详细信息。

日志格式： [time] log_level [module_name] function_name (file_name:linue_number) [thread_id] [coroutine_id] [Ytrace_id][lt=log_used_time] [dc=dropped_msg_count] log_data

参数	说明
log_level	该条日志的级别
module_name	打印该条日志的语句所在模块
function_name	打印该条日志的语句所在的函数（以及代码源文件名和行号）
thread_id	打印该条日志的线程的线程号
coroutine_id	当前打印日志的协程 ID
trace_id	用于跟踪一个任务的 trace_id，任务级唯一，该 trace_id 通过 RPC 在各个 OBServer 间传递，从而可以根据 trace_id 来获取一个任务的所有日志数据。对应 GV$OB_SQL_AUDIT 视图中的 TRACE_ID 字段。
log_used_time	写上一条日志所用的数据，单位为 us
log_data	具体的日志数据，通常的形式为 info(parameters)

```
[2022-09-14 20:51:52.308317] WARN  [SQL.DTL] process_base (ob_dtl_channel_loop.cpp:232)
[116982][2526][YB42C0A86681-00061E5AE0AD029B-0-0] [lt=6] [dc=0] message loop is interrupted(code={code:-4377,
info:{msg:"tid:73549,from:"192.168.1.3:2882",TASK ABORT QC"}}, ret=-4377)
```

错误码

OceanBase 数据库高度兼容 MySQL 和 Oracle，这不仅体现在常用功能与 SQL 语法上，也体现在返回给应用程序的错误信息上。

错误码格式

模式	错误格式	说明
MySQL模式	ERROR <err_num> (<sql_stat>) : err_msg	➢ err_num 表示错误码。 ➢ sql_stat 表示 SQL STATE。 ➢ err_msg 表示错误信息。
Oracle模式	ORA-<err_num>: <err_msg>	➢ err_num 表示兼容Oracle数据库的错误码。 ➢ err_msg表示错误信息

出于兼容性设计原则，对于同一个错误，在 MySQL 模式（sys 租户属于 MySQL 模式）和 Oracle 模式下可能返回不同的错误码

租户模式	错误码	SQLSTATE	错误消息
MySQL	1054	42S22	Unknown column
Oracle	904，即 ORA-00904	42S22	invalid identifier

WR 与 ASH

WR 概述

Workload Repository（工作负载仓库，简称 WR）会周期性的记录 OceanBase 数据库的性能数据，因此可以通过 WR 回溯 OceanBase 数据库过往的性能记录，也可以研究某一具体时间段内的性能相关问题。

调度机制：通过 ObTimer 定时器线程来触发采集任务。默认情况下，每小时采集一次快照

内容采集：在 OceanBase V4.2 版本中，WR 主要采集以下 3 个性能视图的数据

视图名称	视图含义
[G]V$SYSSTAT	系统监控统计视图，涵盖网络、请求队列、事务、SQL、资源使用、日志等所有关键的执行信息。
V$STATNAME	静态数据，提供系统统计值的信息，能够准确反映数据库运行状况的各项指标和指标名称。
[G]V$ACTIVE_SESSION_HISTORY	对活跃 Session 每秒采样一次，记录系统的运行状况。

数据保存：WR 将采集的数据保存在系统内部表中，可以通过以下系统视图查看

视图名称	视图含义
CDB(DBA)_WR_SNAPSHOT	展示租户的 SNAPSHOT 信息。
CDB(DBA)_WR_ACTIVE_SESSION_HISTORY	展示租户持久化后的 ASH 数据（WR 以 10:1 的采样率进行持久化）。
CDB(DBA)_WR_CONTROL	展示 WR 相关的配置信息。
CDB(DBA)_WR_SYSSTAT	展示统计项指标信息。

WR 管理：设置快照策略

OceanBase 数据库提供了 DBMS 包来管理 WR。在 OceanBase 4.2 版本中，WR 是一个集群级的功能，不能对单个租户进行独立的设置。（注意：WR 的管理功能仅支持在 SYS 租户下使用）

设置快照间隔和保留时间

DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(retention INT DEFAULT NULL,interval INT DEFAULT NULL);

retention：系统快照的保留时间，超期的历史快照会被自动删除。单位为分钟，rentention=0 表示永久保存

interval：创建快照的周期间隔，WR 按照周期设置自动创建快照。单位为分钟，interval=0 表示关闭 WR 的自动采集

设置 WR 的采集频率为 10 分钟，保留时间 3600 分钟

```
CALL DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS ('3600','10');
SELECT TENANT_ID, SNAP_INTERVAL,RETENTION FROM oceanbase.CDB_WR_CONTROL;
```

WR 管理：创建、删除快照

OceanBase 数据库提供了 DBMS 包来管理 WR。在 OceanBase 4.2 版本中，WR 是一个集群级

的功能，不能对单个租户进行独立的设置。（注意：在手动创建快照后，定时器将会以手动创建快照的时间开始重新计时。）

手动创建系统快照：DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT(flush_level VARCHAR(64) DEFAULT 'TYPICAL');

flush_level：目前只支持 TYPICAL，即定期快照

手动删除系统快照：DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE(low_snap_id INT,high_snap_id INT);

low_snap_id：要删除的快照的起始快照 ID

high_snap_id：要删除的快照的结束快照 ID

创建、查看、删除一个快照

```
CALL DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT(flush_level => 'TYPICAL');
SELECT * FROM oceanbase.DBA_WR_SNAPSHOT ORDER BY SNAP_ID DESC LIMIT 1;
CALL DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE ('2140','2140');
```

ASH 概述

ASH（Active Session History）周期性地采样系统中所有 Active Session 的状态，并将采集的数据写入内部虚拟表中供实时查询和 WR 采样使用。

调度机制：总是开启，以 1s 为周期采样系统中所有 ActiveSession 状态

内容采集：数据库自动采集活跃线程的执行信息，特别是线程当前的执行状态和等待耗时

数据保存：

GV$ACTIVE_SESSION_HISTORY：该视图展示实时采集的 ASH 数据

从 V4.2.2 版本开始更名为 GV$OB_ACTIVE_SESSION_HISTORY

CDB(DBA)_WR_ACTIVE_SESSION_HISTORY：该视图展示 WR 持久化后的 ASH 数据，持久化时以 10:1 的比例进行采样

ASH 管理：生成 ASH 报告

OceanBase 数据库提供了 ASH PL 包 DBMS_WORKLOAD_REPOSITORY. ASH_REPORT 生成对应的 ASH 报告。

生成 ASH 报告：ASH_REPORT(BTIME IN DATE,ETIME IN DATE,SQL_ID IN VARCHAR2 DEFAULT NULL,TRACE_ID IN VARCHAR2 DEFAULT NULL,WAIT_CLASS IN VARCHAR2 DEFAULT NULL,REPORT_TYPE IN VARCHAR2 DEFAULT 'text');

BTIME/ETIME：采样数据的开始与结束时间

SQL_ID/TRACE_ID：采样 SQL 的 SQL_ID 与 Trace_ID

WAIT_CLASS：采样的等待事件类型

REPORT_TYPE：报告类型，目前仅支持 TEXT 类型

示例：在 Oracle 租户内生成某个时间段特定 SQL 的 ASH 报告

ASH 报告内容

ASH_REPORT 通过对采样数量的分析汇总，从不同的维度生成 9 个模块的信息，以便于用户从多个角度或组合来进行性能分析。

ASH 报告的 9 个模块

Top User Events：占比最高的用户进程中排名靠前的等待事件

Top Events P1/P2/P3 Value：占比靠前的等待事件的百分比以及参数值

Top Phase of Execution：百分比最高的执行阶段的排序，例如 SQL、PL/SQL 和 Java 编译和执行的过程

Top SQL with Top Events：活动百分比最高的 SQL 语句以及这些 SQL 语句的等待事件

Top SQL with Top Blocking Events：显示被阻塞的会话信息

Complete List of SQL Text：Top SQL 中 SQL 语句的完整文本内容

Top Sessions：显示等待事件排序的靠前活跃会话信息

Top Blocking Sessions：显示占采样会话活动百分比最高的被阻塞会话

Top latches：显示了在采样会话活动中百分比最高的闩锁

案例：使用 ASH 报告分析性能问题

案例：业务测试数据导入耗时过高，需要定位主要的耗时原因。

确定 TOP 等待事件

CPU + Wait for CPU ： 非等待的状态占比 59.00%

das wait remote response: 通过 DAS 访问数据等待远程响应 39.22%，占比过高

确定执行阶段 TOP 指标

IN_SQL_EXECUTION: SQL 正常执行耗时占比 47.096%

IN_STORAGE_WRITE: 存储 IO 写入耗时占比 45.883%

IN_REMOTE_DAS_EXECUTION: DAS 远程执行耗时占比 40.594%，占比过高

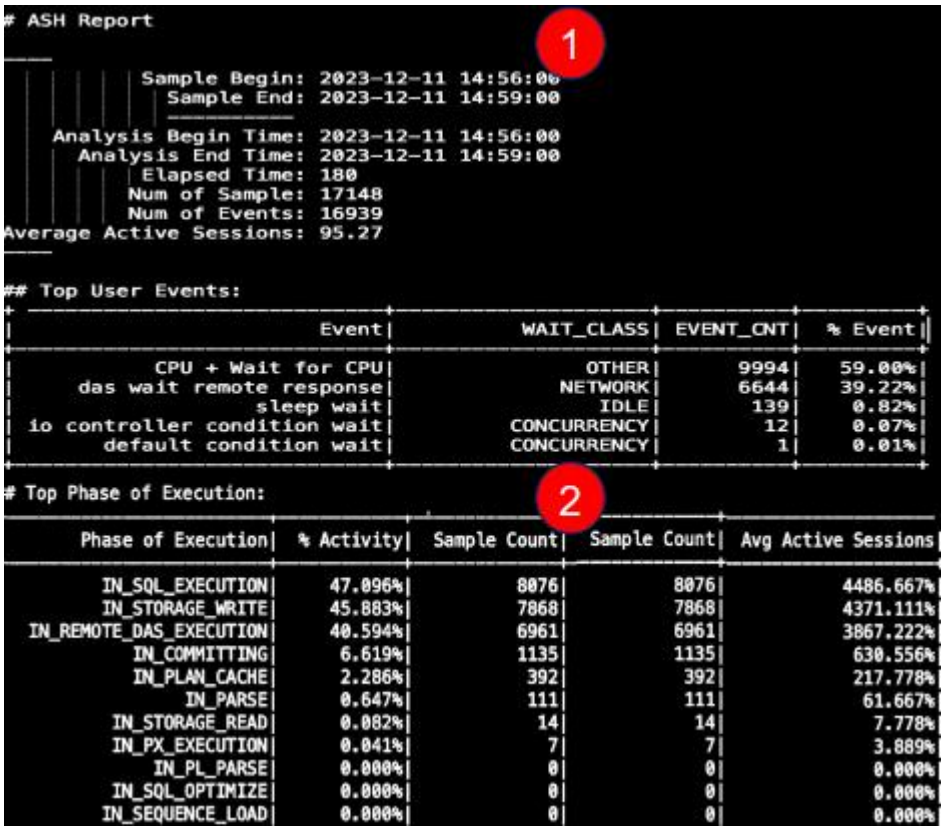
识别造成等待事件的 SQL 以及执行计划

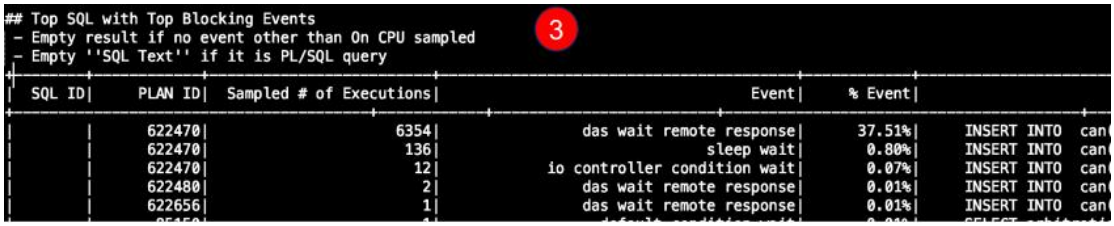
Plan ID 为 622470 的 SQL 执行引起主要的 das wait remote response 等待（37.51%）

也可以通过查询 ASH 视图获取相关的 SQL：SELECT * FROM (SELECT SQL_ID, TRACE_ID, COUNT(1) AS NUM FROM GV$ACTIVE_SESSION_HISTORY WHERE EVENT LIKE"%DAS WAIT REMOTE%" GROUP BY SQL_ID,TRACE_ID) ORDER BY NUM DESC LIMIT 20;

用 QL_ID 或者 trace_id 去 SQL_AUDIT 等系统视图查询该 SQL 的执行统计，或者在 SQL EXPLAIN 视图查询执行计划信息

在这个案例中，最终确定是 Insert SQL 被 OBProxy 错误路由导致了远程的数据访问





使用 OCP 生成 ASH 报告

路径：租户--会话--活跃会话历史报告

全链路追踪

全链路 Trace（FLT）

全链路追踪依赖全链路 Trace，应用端、数据库代理端和数据库端分别生成具有统一 Trace ID 的 SQL 执行日志，再使用数据库命令或工具将分布在各个链路端点的 Trace 日志收集整合在一起，形成图形化的全链路追踪报告。

全链路 Trace 的生成

应用客户端：比如 OBClient 或 JDBC 驱动，生成当前 SQL 执行在客户端的 trace 日志，并随着下一条 SQL 请求一起传输给 OBProxy

请求路由层：OBProxy 生成代理端的 trace 日志，并结合收到的应用端的 trace 日志一

起记录在 obproxy 的 trace.log 中

数据库层：参与 SQL 执行的每个生成 OBServer 节点分别生成数据库端的 trace 日志，并记录在 observer 的 trace.log 中

全链路 Trace 的图形化展示

通过 SHOW TRACE 命令

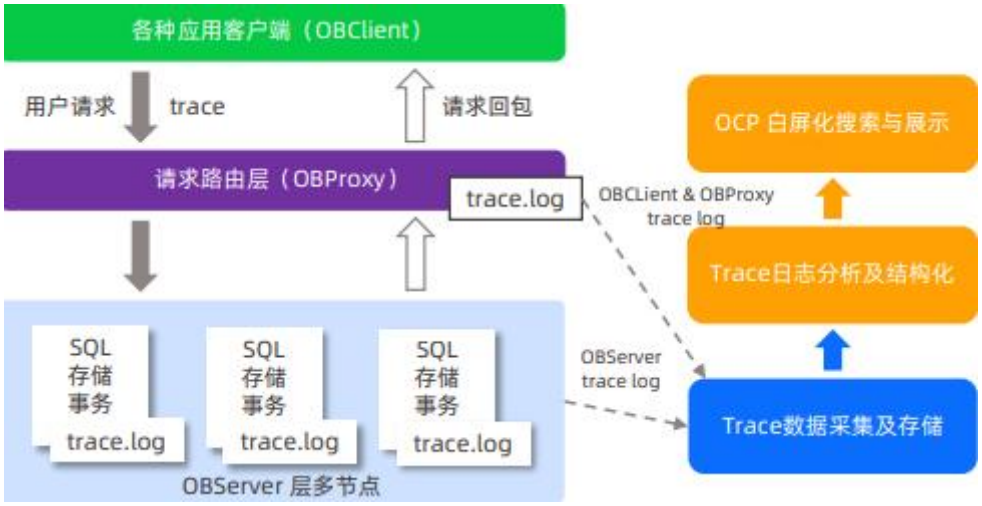
首先设置变量 ob_enable_show_trace=true

执行 SQL 后，执行 SHOW TRACE

通过工具搜集并展示全链路 Trace

OCP 云平台

obdiag 工具



设置全链路 Trace 收集策略：租户

OceanBase 默认开启全链路 Trace 功能，用户可以使用 DBMS_MONITOR 系统包来配置租户的全链路 Trace 收集策略。

设置当前租户的收集策略：DBMS_MONITOR.OB_TENANT_TRACE_ENABLE(level IN INT,sample_pct IN NUMBER,record_policy IN VARCHAR2);

tenant_name：只能指定当前租户的名字，Null 表示当前租户

level：trace 的粒度，1~3 级，等级越高粒度越小

sample_pct：事务内的 SQL 采样频率

record_policy：全链路 Trace 的收集策略

ALL：所有的 query 执行都记录全链路 Trace

ONLY_SLOW_QUERY：只记录 slow query 的全链路 Trace

SAMPLE_AND_SLOW_QUERY：既记录 slow query 的全链路 Trace，也记录采用 query 的全链路 Trace 日志

关闭指定租户的全链路 Trace：DBMS_MONITOR.OB_TENANT_TRACE_DISABLE(tenant_name IN VARCHAR2 DEFAULT NULL);

设置当前租户的全链路跟踪：CALL DBMS_MONITOR.OB_TENANT_TRACE_ENABLE(1,0.1,'SAMPLE_AND_SLOW_QUERY');

关闭当前租户的全链路跟踪：CALL DBMS_MONITOR.OB_TENANT_TRACE_DISABLE(null);

设置全链路 Trace 收集策略：会话

OceanBase 默认开启全链路 Trace 功能，用户可以使用 DBMS_MONITOR 系统包来配置租户的全链路 Trace 收集策略。

设定指定 Session 的收集策略：DBMS_MONITOR.OB_SESSION_TRACE_ENABLE(session_id IN BINARY_INTEGER DEFAULT NULL,level IN INT,sample_pct IN NUMBER,record_policy IN VARCHAR2);

停止采集指定 Session 的全链路 Trace：DBMS_MONITOR.OB_SESSION_TRACE_DISABLE(session_id IN BINARY_INTEGER);

session_id：会话 ID。null 表示当前 session

设置指定 Session 的全链路跟踪：CALL DBMS_MONITOR.OB_SESSION_TRACE_ENABLE(3221918689,1,0.5,'ONLY_SLOW_QUERY') ;

关闭当前 Session 的全链路跟踪：CALL DBMS_MONITOR.OB_SESSION_TRACE_DISABLE(null) ;

设置全链路 Trace 收集策略：应用连接

除了按照租户、会话配置全链路 Trace 的收集策略外，OceanBase 也支持按照应用的标识来单独设置全链路 Trace 的收集策略。

应用侧：设置应用的 client_id DBMS_SESSION.SET_IDENTIFIER(client_id VARCHAR2);

应用侧：清除应用的 client_id DBMS_SESSION.CLEAR_IDENTIFIER();

数据库侧：设置指定 client_id 的收集策略 DBMS_MONITOR.OB_CLIENT_ID_TRACE_ENABLE(client_id IN VARCHAR2,level IN INT,sample_pct IN NUMBER,record_policy IN VARCHAR2);

数据库侧：关闭指定 client_id 的全链路 trace DBMS_MONITOR.OB_CLIENT_ID_TRACE_DISABLE(client_id IN VARCHAR2);

示例：

设置应用连接的标识：CALL DBMS_SESSION.SET_IDENTIFIER('obclient') ;

设置对应的收集策略：CALL DBMS_MONITOR.OB_CLIENT_ID_TRACE_ENABLE('obclient',1,0.1,'ONLY_SLOW_QUERY') ;

设置全链路 Trace 收集策略：程序模块

更进一步，OceanBase 还支持对应用程序的 Module 和 Action 单独设置全链路 Trace 的收集策略。

应用层：设置当前程序的 Module、Action 标识。DBMS_APPLICATION_INFO.SET_MODULE (module_name IN VARCHAR2,action_name IN VARCHAR2);

应用程序执行过程中，通过 SET_MODULE 注册自己的标识

应用层：设置当前程序的 Client Info 标识 DBMS_APPLICATION_INFO.SET_CLIENT_INFO (client_info IN VARCHAR2)；

Client Info 是标识客户端应用程序的额外信息

数据库侧：设置指定 Module 和 Action 的收集策略 DBMS_MONITOR.OB_MOD_ACT_TRACE_ENABLE(module_name IN VARCHAR2 DEFAULT ANY_MODULE,action_name IN VARCHAR2 DEFAULT ANY_ACTION,level IN INT,sample_pct IN NUMBER,record_policy IN VARCHAR2);

数据库侧：清除指定 Module 和 Action 的收集策略 DBMS_MONITOR.OB_MOD_ACT_TRACE_DISABLE(module_name IN VARCHAR2 DEFAULT ANY_MODULE,action_name IN VARCHAR2 DEFAULT ANY_ACTION)；

设置应用程序的标识：CALL DBMS_APPLICATION_INFO.SET_MODULE('backup','insert') ;

设置对应的收集策略：CALL DBMS_MONITOR.OB_MOD_ACT_TRACE_ENABLE('backup','insert',1,0.1,'ONLY_SLOW_QUERY') ;

查询全链路 Trace 的收集策略

GV$OB_FLT_TRACE_CONFIG：展示租户或应用标识的全链路 Trace 的配置信息

SELECT * FROM GV$OB_FLT_TRACE_CONFIG;

GV$OB_PROCESSLIST：包含 Session 的全链路追踪配置信息

SELECT ID,SVR_IP,TENANT,DB,COMMAND,STATE,ACTION ,MODULE ,CLIENT_INFO ,"LEVEL" ,SAMPLE_PERCENTAGE, RECORD_POLICYFROM GV$OB_PROCESSLIST WHERE MODULE='backup';

展示全链路 Trace：SHOW TRACE 命令

当前会话打开在线全链路 Trace 的搜集后，可以使用 SHOW TRACE 命令来展示上一条 SQL 执行的全链路追踪信息。

打开 SQL Trace 功能：SET ob_enable_show_trace = 1;

手动执行需要查询的 SQL 语句：select/*+parallel(2)*/ count(*) from t1;

通过 SHOW TRACE 语句查看全链路耗时信息；SHOW TRACE；

注意：在线收集的的全链路信息存放在内存中，不会写入日志文件。

展示全链路 Trace：obdiag 工具

obdiag 是 OceanBase 设计的一款黑盒诊断工具。它的功能覆盖了对 OceanBase 日志、SQL 审计以及 OceanBase 进程堆栈等信息的扫描、收集和分析，包括全链路 Trace 的收集与展示。

使用 OCP、OBD 或手工安装 obdiag 工具 v1.5.0 + 版本，并配置 OceanBase 集群的相关信息（详细步骤请参考官网文档）

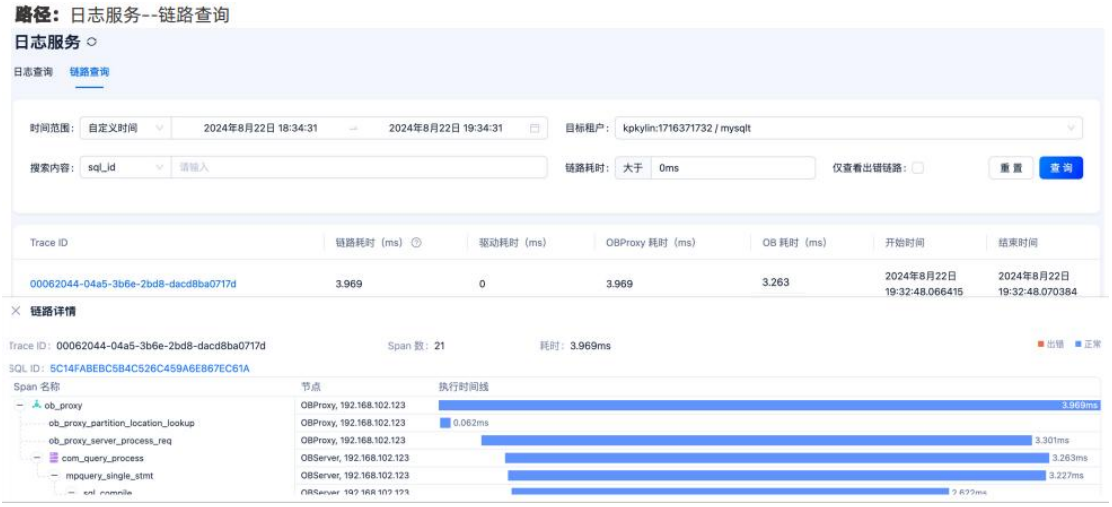
（可选）设置租户或应用的全链路 Trace 策略

执行 SQL，获取 SQL 执行的全链路 Trace ID：SELECT query_sql, flt_trace_id FROM oceanbase.GV$OB_SQL_AUDIT WHERE query_sql LIKE 'select * from tx%';

执行 obdiag 命令，收集指定 flt_trace_id 的全链路 trace，并图形化展示

展示全链路 Trace：OCP

OCP 使用 opensearch 插件实现对 OBServer、OBProxy 日志的汇总和检索，也实现了全链路 Trace 的收集与展示功能。

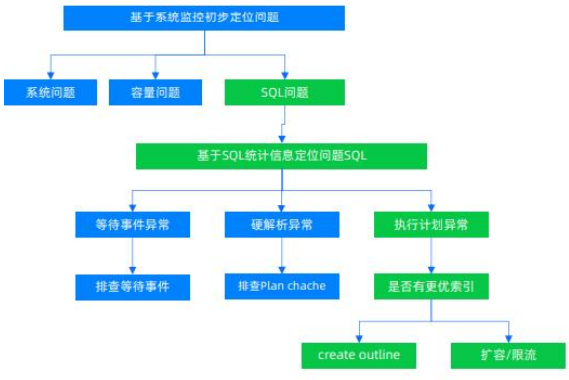


SQL 调优

SQL 执行监控

SQL 监控与调优概述

基于 SQL 统计的诊断流程



视图名称	功能描述
GV$OB_PROCESSLIST	展示所有会话的执行状态信息
GV$OB_SQL_AUDIT	展示每一次 SQL 执行的统计信息
GV$OB_PLAN_CACHE_PLAN_STAT	展示 Plan Cache 中计划的执行统计信息
GV$OB_PLAN_CACHE_PLAN_EXPLAIN	展示 Plan Cache 中计划的物理执行计划
GV$OB_SQL_PLAN	展示 Plan Cache 中计划的逻辑执行计划
GV$SQL_PLAN_MONITOR	展示每个算子的实时执行统计信息
GV$OB_PS_STAT	展示 PS Cache 的整体状态
GV$OB_PS_ITEM_INFO	展示 PS Cache 中 PS 的信息
DBA_OB_OUTLINES	展示系统中创建的 Outline

查看正在执行中的 SQL 语句

视图 GV$OB_PROCESSLIST 用于展示租户所在的所有 OBServer 节点的会话信息，可以用来查看正在执行的 SQL 语句

SELECT svr_ip, tenant, db, user, id, state, command, sql_id, trace_id, time, total_time, info FROM GV$OB_PROCESSLISTWHERE state = 'active' ORDER BY TIME DESC LIMIT 1\G

SVR_IP：当前会话所在的服务器 IP

ID：会话 ID

SQL_ID：SQL text 的 md5 值，唯一标识一条 SQL

TRACE_ID：SQL 执行的跟踪 ID，唯一标识一次执行

STATE：会话状态，比如 ACTIVE、SLEEP

TIME：当前命令执行时间，单位是秒。如果命令发生重试，会清零后重新计算

TOTAL_TIME：当前命令执行总时间，单位是秒

COMMAND：当前执行的命令类型

INFO：当前正在执行的命令或 SQL

也可以使用 SHOW PROCESSLIST 命令查看活跃会话的信息，其展示内如与视图(G)V$OB_PROCESSLIST 基本一致

查看执行中的 SQL 的物理执行计划与实时统计查看执行中的 SQL 的物理执行计划与实时统计

视图 GV$SQL_PLAN_MONITOR 实时统计 SQL 执行的信息，包括 SQL 的物理执行计划、算子吐行数、算子开始和结束时间点以及各个执行线程执行算子的执行状态等

查看执行中的 SQL 的逻辑执行计划和实时统计

在 MySQL 模式和 Oracle 模式，均可以调用 DBMS_XPLAN.DISPLAY_ACTIVE_SESSION_PLAN 获取活跃会话正在执行的 SQL 的逻辑执行计划和执行统计

注意：

Session ID 、SVR_IP、SVR_PORT 可以从 GV$OB_PROCESSLIST 中获取，Session ID 即视图中的 ID 字段。

如果执行 SQL 的 Session 和执行 DBMS_XPLAN 命令的 Session 连接的 SVR_IP 相同，DISPLAY_ACTIVE_SESSION_PLAN 可以仅指定 Session ID

查看已执行 SQL 的执行统计

视图 GV$OB_SQL_AUDIT 用于展示租户的已执行完成的 SQL 的执行统计，可以从该视图中查看 SQL 的执行时间、SQL_ID、PLAN_ID 等信息

SVR_IP：SQL 执行所在的数据库节点 IP

REQUEST_TIME：数据库接收到 SQL 请求的时间

SID：执行 SQL 的服务器会话 ID

TRACE_ID：SQL 执行的跟踪 ID，唯一标识一次执行

SQL_ID：SQL text 的 md5 值，唯一标识一条 SQL

QUERY_SQL：SQL 语句文本

PLAN_ID：执行计划 ID
PLAN_TYPE：执行计划类型
ELAPSED_TIME：本次执行的总耗时
QUEUE_TIME：SQL 在队列中等待的时间
EXECUTE_TIME：执行计划的总执行时间
TOTAL_WAIT_TIME_MICRO：执行过程中的各类等待事件
RETURN_ROWS：查询返回的记录数
AFFECTED_ROWS：DML 修改的记录数
RET_CODE：执行结果返回码
注意：Oracle 模式没有 USEC_TO_TIME 函数来将 REQUEST_TIME 转化为可读时间，建议使用 SCN_TO_TIMESTAMP(REQUEST_TIME*1000) 来转化

查看已执行 SQL 的物理执行计划

在 OceanBase 数据库中，Plan Cache 自动保留已执行 SQL 的物理执行计划，可以使用系统视图 (G)V$OB_PLAN_CACHE_PLAN_EXPLAIN 来查看 SQL 的物理执行计划。

GV$OB_PLAN_CACHE_PLAN_EXPLAIN 视图基于 KV 结构的内存虚拟表构建，查询该视图必须提供完整的 Key 值，即四元组(tenant_id, ip,port, plan_id)，否则结果返回空值。

GV$OB_PLAN_CACHE_PLAN_EXPLAIN 视图基于 KV 结构的内存虚拟表构建，查询该视图必须提供完整的 Key 值，即四元组(tenant_id,ip, port, plan_id)，否则结果返回空值。

查看已执行 SQL 的逻辑执行计划

OceanBase 数据库 V4.2 版本自动保存所有执行过的 SQL 的逻辑执行计划到内部表中，可以使用 DBMS_XPLAN.DISPLAY_CURSOR 来查看 SQL 执行的逻辑执行计划

查看 PLAN 的执行统计

同一条 SQL 反复在一个 OBServer 节点执行时，大概率会复用 Plan Cache 中的执行计划，视图 (G)V$OB_PLAN_CACHE_PLAN_STAT 从执行计划 PLAN_ID 的维度统计了一个 PLAN 多次执行的信息

SVR_IP：PLAN 所在的数据库节点 IP
SQL_ID：SQL text 的 md5 值，唯一标识一条 SQL
PLAN_ID：执行计划 ID
TYPE：执行计划类型
STATEMENT：参数化后的 SQL 语句文本
QUERY_SQL：生成该执行计划时的 SQL 语句文本（参数化前）
LAST_ACTIVE_TIME：最近一次执行该 PLAN 的 SQL 请求时间
AVG_EXE_USEC：PLAN 的平均执行时间
SLOWEST_EXE_TIME：PLAN 最慢一次执行的 SQL 请求时间
SLOWEST_EXE_USEC：PLAN 最慢一次执行的耗时
EXECUTIONS：PLAN 的执行次数

ELAPSED_TIME：PLAN 所有执行次数的的总耗时

CPU_TIME： PLAN 所有执行次数的的总 CPU 时间

OUTLINE_ID：PLAN 匹配的 OUTLINE ID，-1 表示未使用 OUTLINE

HINTS_INFO：PLAN 使用的 SQL Hint

执行计划管理

CREATE OUTLINE 绑定执行计划

使用 Hint 指定执行计划需要修改 SQL 语句，通常需要应用开发人员修改程序。OceanBase 数据库同时提供了 Outline 从数据库侧为 SQL 绑定执行计划 Hint，而无需修改 SQL 语句。

CREATE OUTLINE otl1 ON SELECT/*+ INDEX(t1 idx_c2)*/ * FROM t1 WHERE c2 = 1;

OceanBase 数据库支持通过两种方式创建 Outline

一：使用 SQL_TEXT 创建 CREATE [OR REPLACE] OUTLINE outline_name ON stmt_with_hint [TO];

指定 OR REPLACE 后，可以对已经存在执行计划进行替换

stmt_with_hint 为一个带有 Hint 和原始参数的 DML 语句，OceanBase 会对 stmt 中的常量进行参数化处理。如果数据库接受的 SQL 参数化后与 stmt 去掉 Hint 参数化文本相同，则将该 SQL 绑定 stmt 中 Hint 生成执行计划

指定 TO target_stmt 表示该 Outline 只对 target_stmt 生效，不会使用 stmt_with_hint 的参数化文本做匹配

二：使用 SQL_ID 创建 CREATE OUTLINE outline_name ON sql_id USING HINT hint_text;

注意：CREATE OUTLINE 需要在 SQL 语句所在的 Schema 下执行，否则 Outline 不会生效！

获取 SQL_ID 与 OUTLINE_DATA

使用 SQL_ID 创建 Outline 时，可以通过以下方法获取 SQL 的 SQL_ID

通过查询 SQL 监控的系统视图，比如 GV$OB_PLAN_CACHE_PLAN_STAT 、GV$OB_SQL_AUDIT 获取

通过参数化的原始 SQL，使用 MD5 生成 SQL_ID

```
#!/usr/bin/python
import hashlib
sql_text='select * from t1 where c1=? and c2 = ?'
sql_id=hashlib.md5(sql_text.encode('utf-8')).hexdigest().upper()
print(sql_id)
```

CREATE OUTLINE 语句中的 hint 可以使用或参考 EXPLAIN EXTENDED 结果中的 outline data

```
CREATE OUTLINE otl_idx_c2 ON"ED570339F2C856BA96008A29EDF04C74" USING HINT/*+BEGIN_OUTLINE_DATAINDEX(@"SEL$1" "test.t1"@"SEL$1" "idx_c2")END_OUTLINE_DATA*/;
```

CREATE FORMAT OUTLINE

SQL 执行时会根据 SQL 文本 或者 SQL ID 与 创建 Outline 时使用的 SQL 文本或 SQL ID 严格匹配。例如，对“select * from A”创建了 Outline，在 SQL 执行时它的匹配结果如下：

SQL 文本	匹配结果
select * from A;	匹配
select * from A;	不匹配
select * from \t A;	不匹配

OceanBase 从 V4.2.2 版本开始，支持以模糊匹配的方式为 SQL 绑定 Outline，即 Format Outline

CREATE [OR REPLACE] FORMAT OUTLINE outline_name ON format_stmt_with_hint [TO];

CREATE FORMAT OUTLINE outline_name ON format_sql_id USING HINT hint_text;

使用 SQL text 时：数据库会对 SQL text 做快速词法解析（fast parser），生成 format SQL text；

使用 SQL ID 时：需要使用 format SQL text 对应的 SQL_ID，可以直接从 GV$OB_SQL_AUDIT 的 FORMAT_SQL_ID 字段获取，也可以使用 STATEMENT_DIGEST_TEXT 函数获取 format SQL text，然后通过 md5 运算得到；

确认 OUTLINE 是否生效

创建 Outline 后，可以通过系统视图 DBA_OB_OUTLINES 或 DBA_OB_FORMAT_OUTLINES 确认是否创建成功；

执行绑定的 SQL，从 Plan Cache 中判断 SQL 是否通过绑定的 Outline 生成了新执行计划；

如果 outline_id 与在 DBA_OB_OUTLINES 中查到的 outline_id 相同，则表示是按绑定的 Outline 生成的执行计划，否则不是

执行计划管理（SPM）

SQL Plan Management（SPM）是一种防止计划回退的机制，能够确保新生成的计划在经过验证后才被使用，以保证计划性能不断优化和更新。

SPM 的功能

捕捉基线计划：SPM 自动捕捉基线执行计划，并提供固定基线计划的功能，以防止执行计划的变动

执行计划演进：当因为统计信息变化等原因触发执行计划生成时，SPM 会自动进行执行计划的演进，总是在基线计划中保留更优的执行计划

SPM 的执行机制

当 SQL 没有可复现的 Baseline 计划时，将新生成的执行计划存入 Baseline

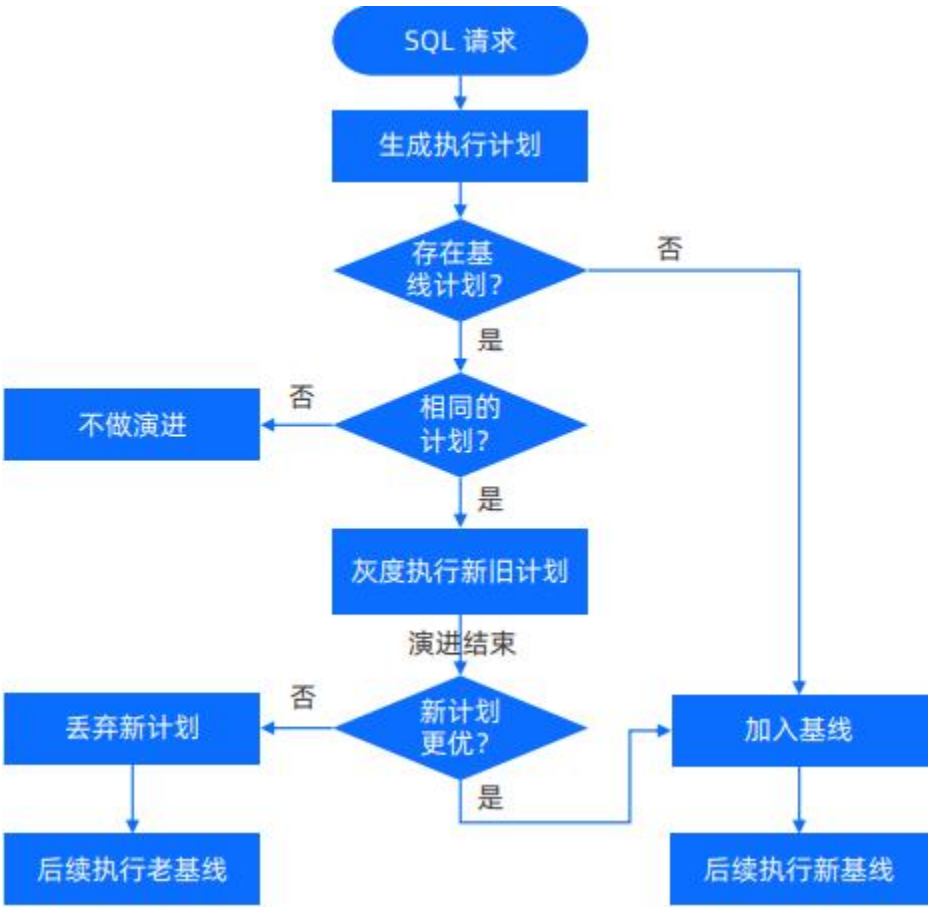
如果有 FIXED 且可复现的 Baseline 计划，优先使用 FIXED 计划，不做演进

如果存在可复现的 Baseline 计划，且 Baseline 不是 FIXED，则对新计划与 Baseline 进行

灰度演进

一部分 SQL 执行使用新计划，一部分使用 Baseline 计划

如果新计划更优，则将新计划存入 Baseline，并在 Plan Cache 中替代老计划



开启 SPM 功能

SPM 功能有其特定的使用场景，在 OceanBase V4.2 版本中 SPM 功能默认是关闭的。（注意：目前 OceanBase 数据库社区版暂不支持 SPM 功能）

需要设置以下系统变量来开启 SPM 功能：

optimizer_capture_sql_plan_baselines：表示当新的计划优于基线计划时，是否捕获新的计划作为基线，默认值为 false

SET [GLOBAL|SESSION] optimizer_use_sql_plan_baselines = on;

optimizer_use_sql_plan_baselines：表示当 SQL 生成新的计划时是否使用基线计划进行演进，默认值为 false

SET [GLOBAL|SESSION] optimizer_capture_sql_plan_baselines = on;

SPM 功能的典型使用场景

大小账号导致的计划走偏：业务表中数据存在倾斜时，优化器生成计划时可能因为使用了特定的大/小账号而导致计划走偏。如果业务大小账号的查询量是均匀的，使用 SPM 可以及时验证执行计划在小/大账号的场景下性能不优，从而避免使用错误的执行计划

分页查询导致的计划走偏：分页查询由于代价模型计算不准确，或者特定索引上的数据分布可以加快执行速度的场景下，优化器容易选错索引导致计划走偏。这种场景下，如

果基线计划是正确的索引，那么开启 SPM 可以避免因为计划走偏导致的分页查询性能下降

SPM 功能的使用限制：由于演进的结果是暂存在 OBServer 节点的 Local Cache 中，然后定期同步到 Inner Table 中，所以任何 OBServer 节点的演进结果对于其它的 OBServer 节点来讲是无法立即感知的

DBMS_SPM 包

OceanBase 提供了类似 Oracle 数据库的 DBMS_SPM 包来管理 SPM 和 Baseline 计划，在 Oracle 和 MySQL 租户均可以使用。

子程序名	类型	描述
LOAD_PLANS_FROM_CURSOR_CACHE	Function	从 Cursor Cache 中读取一条或多条 SQL 计划作为计划基线。
ALTER_SQL_PLAN_BASELINE	Function	修改基线中一个或一组计划的属性。
DROP_SQL_PLAN_BASELINE	Function	丢弃一个或多个计划基线。
ACCEPT_SQL_PLAN_BASELINE	Procedure	结束一个正在演进的任务，并将好的计划转为基线。
CANCEL_EVOLVE_TASK DROP_EVOLVE_TASK	Procedure	取消一个正在演进的任务。
CONFIGURE	Procedure	设置 SPM 的一些参数。

Function：返回值表示被修改的基线数量。可以直接使用 SELECT 语句调用执行。

SELECT DBMS_SPM.DROP_SQL_PLAN_BASELINE('testdb','521413248E4F1EC3F0FEDD9FF9E161C5','4392083053520233360');

Procedure：不返回值，可以使用 CALL 语句调用执行。

CALL DBMS_SPM.ACCEPT_SQL_PLAN_BASELIINE('testdb','521413248E4F1EC3F0FEDD9FF9E161C5','4392083053520233360');

查看 Baseline 计划

系统视图 DBA_SQL_PLAN_BASELINES 记录了 SQL 的基线计划的状态和执行统计信息。

DBA_SQL_PLAN_BASELINES

SQL_HANDLE SQL 的标识符，也就是 SQL_ID

SQL_TEXT SQL 文本

PLAN_NAME Plan Hash Value

ENABLED 是否启用该 Baseline 执行计划

FIXED 是否固定该 Baseline 执行计划

REPRODUCED 该 Baseline 计划是否依然有效

AUTOPURGE 是否允许自动清理该 Baseline 执行计划

EXECUTIONS Baseline 计划创建时计划的执行次数

ELAPSED_TIME Baseline 计划创建时计划的总耗时

CPU_TIME Baseline 计划创建时计划的总 CPU 时间

查看执行计划演进

系统视图 GV$OB_PLAN_CACHE_PLAN_STAT 记录 Plan Cache 中计划的详细信息及其执行统计信息，也包含演进中的执行计划的信息

GV$OB_PLAN_CACHE_PLAN_STAT

SQL_ID SQL 的标识符

STATEMENT SQL 参数化后的文本

PLAN_HASH Plan 的全局唯一识别号

AVG_EXE_USEC 该计划的平均执行时间

EXECUTIONS 该计划总的执行次数

EVOLUTION 该执行计划是否在演进中

EVO_EXECUTIONS 演进过程中的执行次数

EVO_CPU_TIME 演进执行的总 CPU 耗时

开启 SPM 并管理基线计划

本示例展示如何开启 SPM，以及如何干预演进任务、管理 Baseline 计划。本示例展示如何开启 SPM，以及如何干预演进任务、管理 Baseline 计划。

开启会话级别的 SPM 功能

SET optimizer_capture_sql_plan_baselines=on; //* 开启基线计划捕捉

SET optimizer_use_sql_plan_baselines=on; //* 开启执行计划演进

执行 SQL，检查 SPM 捕捉基线计划的结果

SELECT * FROM oceanbase.DBA_SQL_PLAN_BASELINES WHERE SQL_TEXT LIKE ...; //* 确认当前有 0 个基线计划

SELECT * FROM t1 WHERE C2='ABC'； //* 首次运行，生成计划，被捕捉为基线

SELECT * FROM oceanbase.DBA_SQL_PLAN_BASELINES WHERE SQL_HANDLE= ...; //* 确认当前有 1 个基线计划

创建索引，并将演进计划接受为基线计划

CREATE INDEX t1_i2 ON t1(c2); //* 确认没有基线计划

SELECT * FROM T1 WHERE C2='ABC'； //* 重复多次执行该 SQL

SELECT * FROM oceanbase.GV$OB_PLAN_CACHE_PLAN_STAT WHERE SQL_ID=...; //* 确认 Plan Cache 中两个计划在演进中

CALL DBMS_SPM.ACCEPT_SQL_PLAN_BASELINE('testdb','xxxx','xxxx'); //* 将演进中的新计划接受为基线计划

SELECT * FROM oceanbase.DBA_SQL_PLAN_BASELINES WHERE SQL_ID=...; //* 确认当前有 2 个基线计划

注意：在 MySQL 模式中，OceanBase 系统视图属于 oceanbase 数据库；在 Oracle 模式中，OceanBase 系统视图属于 SYS 用户

统计信息收集

统计信息概览

统计信息描述了数据库中数据的分布信息，是数据库优化器进行执行计划代价估算和执行计划选择的主要依据，对 SQL 性能优化至关重要。在 OceanBase V4 版本，数据库可以收集、使用的统计信息如下图所示

统计信息的收集

在 OceanBase V4 版本中，既可以由用户手动收集特定的数据库、表、分区的统计信息，也可以由数据库依据算法智能识别需要收集统计信息的对象，并自动收集

手动收集统计信息：DBMS_STATS

OceanBase 数据库优化器针对手动统计信息收集提供了两种方式：DBMS_STATS（推荐）、ANALYZE 命令行

DBMS_STATS 收集统计信息的程序主要有

GATHER_SCHEMA_STATS：搜集指定 Schema（Database）下所有表和索引的统计信息，也可以收集列的直方图信息

GATHER_TABLE_STATS：搜集指定表的统计信息，也可以收集表上指定列的直方图信息

GATHER_INDEX_STATS： 搜集指定索引的统计信息

```
PROCEDURE gather_table_stats (
    ownname VARCHAR2,
    tabname VARCHAR2,
    partname VARCHAR2 DEFAULT NULL,
    estimate_percent NUMBER DEFAULT AUTO_SAMPLE_SIZE,
    block_sample BOOLEAN DEFAULT FALSE,
    method_opt VARCHAR2 DEFAULT DEFAULT_METHOD_OPT,
    degree NUMBER DEFAULT NULL,
    granularity VARCHAR2 DEFAULT DEFAULT_GRANULARITY,
    cascade BOOLEAN DEFAULT NULL,
    stattab VARCHAR2 DEFAULT NULL,
    statid VARCHAR2 DEFAULT NULL,
    statown VARCHAR2 DEFAULT NULL,
    no_invalidate BOOLEAN DEFAULT FALSE,
    stattype VARCHAR2 DEFAULT 'DATA',
    force BOOLEAN DEFAULT FALSE
    );
```

method_opt：指定搜集列的直方图信息

no_invalidate：是否 invalidate Plan Cache 中缓存的执行计划

force：是否强制搜集统计信息，即使统计信息被锁定

```
CALL DBMS_STATS.GATHER_SCHEMA_STATS ('testdb',degree=>64,no_invalidate=>TRUE);
CALL DBMS_STATS.GATHER_TABLE_STATS('testdb', 'tb1',degree=>4, force=>TRUE);
CALL DBMS_STATS.GATHER_INDEX_STATS('testdb', 'tb1_ix1',degree=>4, tabname=>'tb1');
```

除非指定了 method_opt，搜集 schema 或 table 的统计信息时，默认不搜集列的直方图信息

```
method_opt=>
FOR ALL  [INDEXED | HIDDEN] COLUMNS [size_clause]
| FOR COLUMNS [size clause] column [size_clause] [,column [size_clause]...]
size_clause: SIZE integer | SIZE REPEAT | SIZE AUTO| SIZE SKEWONLY
```

SIZE integer：指定收集列的直方图桶的个数，范围在 [1-2048]

SIZE REPEAT：仅仅只收集已经有收集过的直方图的列的直方图；使用之前收集直方图设置的桶个数

SIZE AUTO：由数据库优化器来决定是否收集列的直方图，取决于列的使用情况，桶个数使用默认值为 254

SIZE SKEWONLY：仅仅只收集数据分布不均匀的列的直方图；直方图桶个数使用默认值为 254

```
CALL DBMS_STATS.GATHER_TABLE_STATS('testdb', 'tb1', degree=>64, method_opt=>'FOR ALL
```

COLUMNS SIZE SKEWONLY');
CALL DBMS_STATS.GATHER_TABLE_STATS('testdb', 'tb1', degree=>4, method_opt=>'FOR COLUMNS C2 SIZE 5');

手动收集统计信息： ANALYZE

ANALYZE 语法（Oracle 模式）：

ANALYZE TABLE table_name [PARTITION (part_name [,...])| SUBPARTITION(subpart_name [,...])]COMPUTE STATISTICS [analyze_for_clause]| ESTIMATE STATISTICS [analyze_for_clause] [SAMPLE INTNUM {ROWS | PERCENTAGE}]analyze_for_clause:FOR TABLE| FOR ALL [INDEXED | HIDDEN] COLUMNS [SIZE integer]| FOR COLUMNS [SIZE integer] column [SIZE integer] [,column [SIZE integer]...]

COMPUTE STATISTICS | ESTIMATE STATISTICS： 计算分析对象的精确统计信息，并将其存储在数据字典中

FOR TABLE：指定仅收集表和列的统计信息，不收集列的直方图信息

FOR ALL [INDEXED | HIDDEN] COLUMNS：收集表的统计信息，同时收集指定列（所有列、索引列或隐藏列）的直方图信息

SIZE：指定直方图中的最大存储桶数，默认值是 75

ANALYZE TABLE tbl1 COMPUTE STATISTICS FOR ALL INDEXED COLUMNS SIZE 100;

注意：MySQL 模式的 ANALYZE 命令功能比较简单，建议使用 DBMS_STATS 来进行统计信息的手动收集。

在线收集统计信息

在线统计信息收集指在批量插入或导入数据时，数据库优化器同时收集统计信息，不用手动调用系统包

支持在线统计信息搜集的场景主要有

CREATE TABLE AS SELECT：默认启动在线收集统计信息功能

INSERT INTO … SELECT：满足以下任一条件时，开启在线收集统计信息功能

使用 PDML（Parallel DML）

使用旁路导入（Append）

指定 Hint：GATHER_OPTIMIZER_STATISTICS

使用 PDML： INSERT /*+PARALLEL(4) ENABLE_PARALLEL_DML*/ INTO T1 SELECT * FROM FROM T2;

使用旁路导入： INSERT /*+APPEND PARALLEL(4) ENABLE_PARALLEL_DML*/ INTO T1 SELECT * FROM FROM T2;

指定 Hint： INSERT /*+GATHER_OPTIMIZER_STATISTICS*/ INTO T1 SELECT * FROM FROM T2;

自动收集统计信息

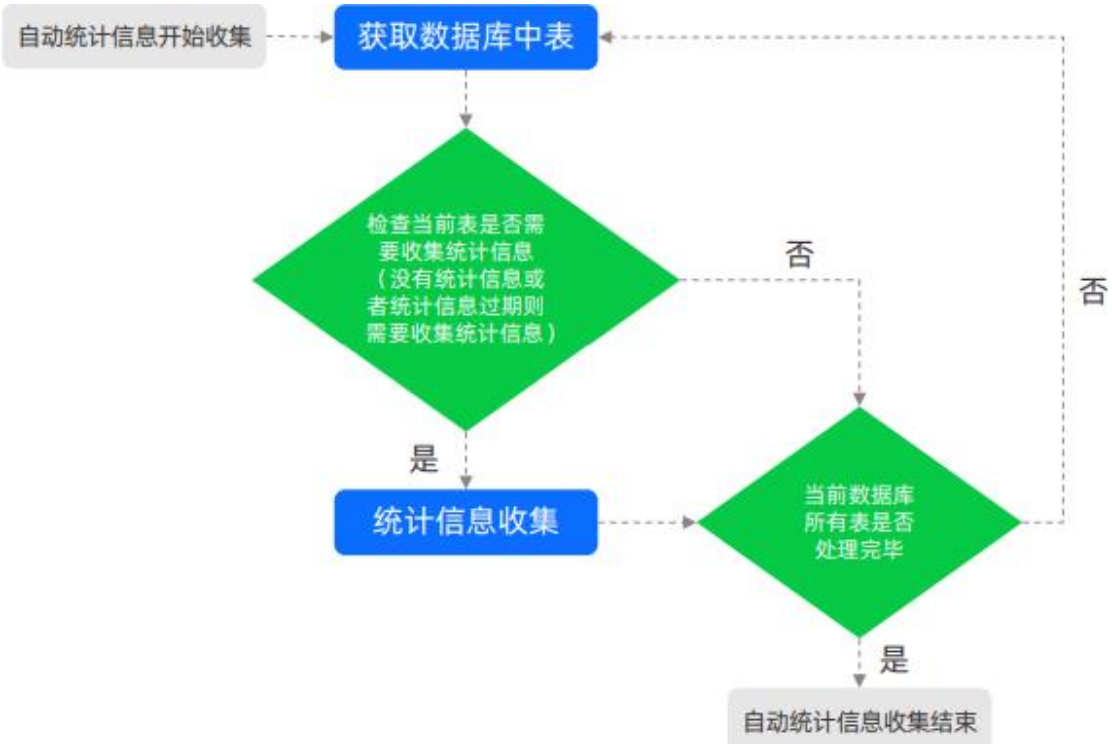
OceanBase V4 提供了自动收集统计信息的机制，数据库根据表的数据变化来判断其统计信

息是否过期，并在合适的时间窗口按照统计信息收集策略自动为统计信息过期的表收集统计信息

过期判断：从上一次收集统计信息后，该表或分区的增/删/改的比例达到阈值，则认为该表或分区的统计信息过期

收集策略：统计信息收集的粒度、范围以及并发度等，还有统计信息的过期阈值，以及历史统计信息的保留时间等

收集窗口：数据库优化器定义了周一到周日 7 个自动统计信息收集任务，按照配置的时间窗口进行统计信息的自动收集



数据变更的监控

自动统计信息收集中，判断一个表的统计信息是否过期，主要依据上一次统计信息收集时间到本次收集期间该表的做增/删/改的比例（默认 10%），OceanBase 数据库优化器也提供了相关视图用于查询一张表的增/删/改次数

模式	视图名称	描述
Oracle	ALL_TAB_MODIFICATIONS	查询表所有的 DML stats
Oracle	DBA_TAB_MODIFICATIONS	查询表所有的 DML stats
Oracle	USER_TAB_MODIFICATIONS	查询表所有的 DML stats
MySQL	oceanbase.DBA_TAB_MODIFICATIONS	查询表所有的 DML stats

字段说明

INSERTS/UPDATES/DELETES：统计自上一次自动收集统计信息以来的 DML 执行次数，

转储合并后显示的是 DML 修改的记录个数

TRUNCATED：指示上一次自动收集统计信息后该表、分区是否被 Truncate 过

DROP_SEGMENTS：自上次分析以来删除的分区和子分区段数

设置默认的收集策略

OceanBase 数据库提供了类似 Oracle 的 DBMS_STATS 系统包，用来收集、管理统计信息，以及配置统计信息的默认收集策略（Prefs）。DBMS_STATS 系统包在 MySQL 租户与 Oracle 租户均可使用。

查看 Prefs：

GET_PREFS：获取 Prefs 设置的 value

设置 Prefs：

SET_GLOBAL_PREFS：设置租户的 Prefs

SET_SCHEMA_PREFS：设置 schema 的 Prefs

SET_TABLE_PREFS：设置表的 Prefs

RESET_GLOBAL_PREF_DEFAULTS：还原租户的 Prefs

删除 Prefs：

DELETE_SCHEMA_PREFS：删除 schema 的 Prefs

DELETE_TABLE_PREFS：删除表的 Prefs

策略名称	描述
APPROXIMATE_NDV	是否使用估算方式计算 NDV，默认为 True
CASCADE	是否同时收集表和索引的统计信息，默认为 True
DEGREE	设置统计信息收集任务的并行度，默认为 1
ESTIMATE_PERCENT	设置数据采样比例，默认使用自适应的比例
GRANULARITY	设置统计信息收集时的分区粒度，默认为 AUTO
INCREMENTAL	是否采用增量收集策略，默认为 False
METHOD_OPT	设置收集直方图信息的默认选项
STALE_PERCENT	设置导致统计信息过期的数据变更百分比，默认 10%
STATS_RETENTION	设置历史统计信息的保留时间，默认 31 天

设置统计信息的过期策略

本示例展示在 Oracle 租户中使用 DBMS_STATS 系统包用来设置租户与表的统计信息过期策

略

查看过期策略（STATE_PERCENT）

SELECT DBMS_STATS.GET_PREFS('STALE_PERCENT') FROM DUAL;

SELECT DBMS_STATS.GET_PREFS('STALE_PERCENT', 'TEST', 'T1') FROM DUAL;

设置过期策略（STATE_PERCENT）：

CALL DBMS_STATS.SET_GLOBAL_PREFS('STALE_PERCENT', '15');

CALL DBMS_STATS.SET_TABLE_PREFS('TEST','T1','STALE_PERCENT','20');

删除过期策略（STATE_PERCENT）：

CALL DBMS_STATS.DELETE_TABLE_PREFS('TEST', 'T1', 'STALE_PERCENT');

CALL DBMS_STATS.DELETE_SCHEMA_PREFS('TEST', 'STALE_PERCENT');

自动收集窗口

OceanBase 数据库预定义周一到周日 7 个自动统计信息收集的任务执行窗口。

MAINTENANCE WINDOW：

维护窗口名字	开始时间/频率	最大收集时长
MONDAY_WINDOW	22:00/per week	4 hours
TUESDAY_WINDOW	22:00/per week	4 hours
WEDNESDAY_WINDOW	22:00/per week	4 hours
THURSDAY_WINDOW	22:00/per week	4 hours
FRIDAY_WINDOW	22:00/per week	4 hours
SATURDAY_WINDOW	6:00/per week	20 hours
SUNDAY_WINDOW	6:00/per week	20 hours

可以使用 DBA_SCHEDULER_JOBS 来查询 MAINTENANCE WINDOW 的信息

设置自动收集窗口

OceanBase 数据库基于 DBMS_SCHEDULER 系统包来管理 MAINTENANCE WINDOW，实现每日的自动统计信息收集。

禁止/启用指定时间窗口的自动统计信息收集任务：

DBMS_SCHEDULER.DISABLE($window_name)

DBMS_SCHEDULER.ENABLE($window_name)

设置自动统计信息收集任务在指定时间窗口的开始执行时间

DBMS_SCHEDULER.SET_ATTRIBUTE($window_name, 'NEXT_DATE', $next_time);

设置自动统计信息收集任务在指定时间窗口的执行时长：

DBMS_SCHEDULER.SET_ATTRIBUTE($window_name, 'JOB_ACTION', $job($duration));

总结

本课程基于 OceanBase V4.2 版本的特性，围绕 DBA 日常运维的内容，重点讲述了集群与租户管理，容灾与高可用，以及性能调优和监控诊断的常见操作和最佳实践。

配置项管理：配置项的级别，生效方式，配置方法，以及查看方法

系统变量管理：变量的级别，作用范围，设置方法，以及查看方法

集群运维： 状态与资源使用的监控，添加、删除节点和可用区的集群弹性扩缩容

租户运维： 租户 Locality、Primary Zone 的修改，修改资源规格与资源单元个数的租户弹性扩缩容

磁盘管理：磁盘 IO 性能数据的校准，日志盘的管理与写入限速，数据盘的动态扩展

用户与安全管理：MySQL 与 Oracle 两种模式下的用户与权限管理，以及网络访问控制

回收站与闪回查询：回收站的管理和对象闪回，多版本数据控制与闪回查询

物理备份与恢复：数据备份与日志归档的配置，物理备份与恢复的操作方法

租户级主备库：主备库的部署方式，日志传输服务，备租户的创建与同步，主备库的切换与解耦

监控诊断体系：系统日志，系统视图，全链路 Trace，WR 与 ASH 报告

系统性能诊断：WR 与 ASH 的配置，使用 ASH 报告分析系统性能问题

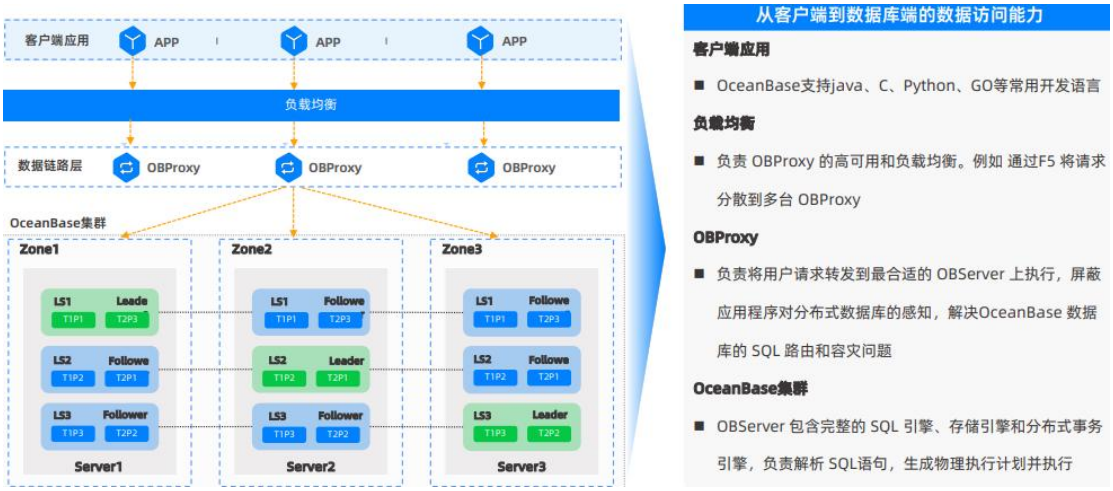
链路性能诊断：全链路跟踪的配置，全链路跟踪报告的生成方式

SQL 监控与性能优化：SQL 监控视图，执行计划管理，统计信息收集

数据链路管理

数据链路总览

数据链路总览



驱动程序简介

基于 OceanBase 数据库进行应用开发，需要根据租户的类型和应用开发语言，选择合适的驱动程序

MySQL 租户

Java 程序支持的驱动：OceanBase Connector/J、MySQL Connector/J

C 程序支持的驱动：OceanBase Connector/C、OceanBase Connector/ODBC

Python 程序支持的驱动：PyMySQL、MySQLClient

GO 程序支持的驱动：Go-SQL-Driver/MySQL

ORACLE 租户

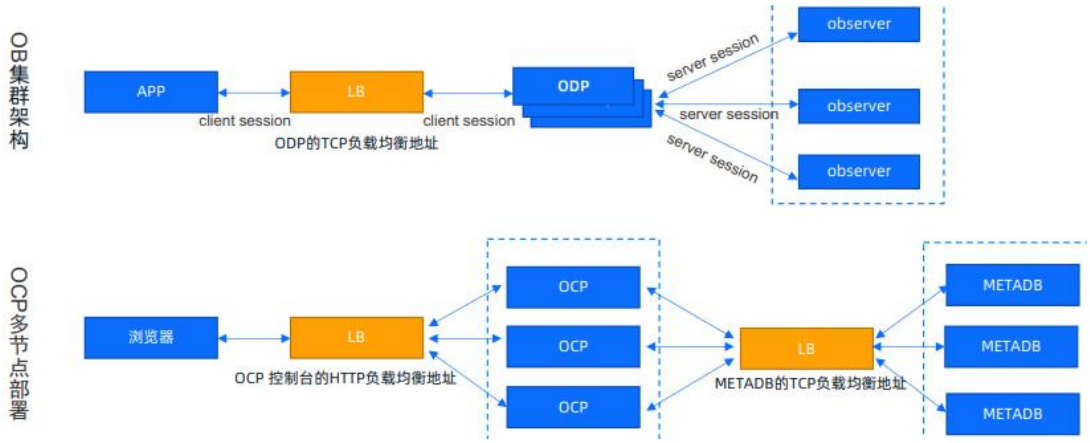
Java 程序支持的驱动：OceanBase Connector/J

C 程序支持的驱动：OceanBase Connector/C、OBCI（OceanBase Call Interface）、OceanBase Connector/ODBC

负载均衡简介

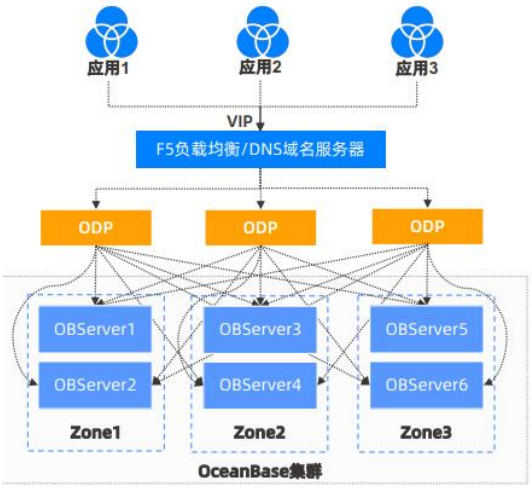
OBProxy（简称 ODP） 集群的各进程之间是相互独立的，例如修改一个 ODP 的配置不会影响其它 ODP 的配置。所以，为了实现 ODP 的高可用和负载均衡，通常会在 ODP 上再架一层 SLB/F5/OBLB 等负载均衡服务

OCP 多节点部署时，会在多个节点上部署 OCP 管理服务（OCP-Server）和元信息数据库（MetaDB）集群。通常需要在 OCP 管理服务和 METADB 集群前部署负载均衡服务，通过流量分配、故障切换来提升 OCP 系统的可靠性和整体性能



OBPrxoy 简介

OBProxy（简称 ODP） 是 OceanBase 数据库的反向代理层，提供了从用户端到数据库端的连接管理和路由选择功能，屏蔽应用程序对分布式数据库的感知，实现像使用单机数据库一样使用分布式数据库



连接管理

- **功能特性**：ODP 实现了很多的连接功能特性，如访问不同集群、不同租户，物理备库等，以及兼容 kill、show processlist 等命令。
- **高可用特性**：ODP 可以处理超时、机器状态变化、网络状态变化等问题，屏蔽后端异常，让用户无感知。

路由选择

- **路由特性**：ODP 充分考虑用户请求涉及的副本位置、用户配置的读写分离路由策略、OceanBase 多地部署的最优链路，以及 OceanBase 各机器的状态及负载情况，将用户的请求路由到最佳的 OBServer 节点，最大程度地保证了 OceanBase 整体的高性能运转。

驱动程序

JAVA 驱动程序

OceanBase 支持驱动类型

OceanBase 数据库支持 MySQL 模式和 Oracle 模式两种租户类型，在进行应用程序开发时，需要根据租户的类型，选择合适的驱动程序

Java 程序

MySQL Connector/J：是 MySQL 官方提供的 JDBC 驱动程序，支持连接 OceanBase 数据库的 MySQL 租户

OceanBase Connector/J：为基于 Java 开发的应用程序提供与 OceanBase 数据库的连接。支持连接 OceanBase 数据库的 MySQL 租户和 Oracle 租户

C 程序

C 驱动（OceanBase Connector/C）：也称为 libobclient，支持 C/C++ 程序访问 OceanBase 分布式数据库集群。支持 OceanBase 数据库的 MySQL 租户和 Oracle 租户

OceanBase Connector/ODBC：基于 MySQL ODBC 驱动进行了定制开发。支持连接 MySQL 租户和 oracle 租户

OBCI（OceanBase Call Interface）：与 Oracle OCI 完全兼容的 OceanBase C 语言接口。支持连接 Oracle 租户

Python 程序

Python 驱动（PyMySQL）：PyMySQL 是在 Python3.x 版本中用于连接 MySQL 服务器的一个库。支持连接 MySQL 租户

MySQLClient：是 Python 程序访问 MySQL 数据库的接口，基于 MySQL API，提供高性能的数据库访问。支持连接 MySQL 租户

GO 程序

Go-SQL-Driver/MySQL：用于 Go 语言的 MySQL 数据库驱动程序。支持连接 MySQL 租

户

JAVA 驱动简介

JAVA 程序支持通过以下两种驱动程序连接 OceanBase 数据库集群

MySQL Connector/J 是 MySQL 官方的 JDBC 驱动程序，支持连接 MySQL 租户

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
Connection conn =
DriverManager.getConnection(“jdbc:mysql://172.28.15.107:2883/test?user=root@mysql_test#obcp_cluster&password=******");
```

OceanBase Connector/J 是 OceanBase 官方实现的 JDBC 驱动程序， 完全兼容 MySQL JDBC 的使用方式，可以自动识别 OceanBase 数据库的运行模式是 MySQL 还是 Oracle，并在协议层同时兼容这两种模式

```
Class.forName("com.oceanbase.jdbc.Driver").newInstance();
Connection conn =
DriverManager.getConnection(“jdbc:oceanbase://172.28.15.107:2883/SYS?user=SYS@oracle_test#obcp_cluster&password=******");
```

Java 程序—OceanBase Connector/J URL 连接属性

OceanBase Connector/J 允许在 URL 中添加额外的连接属性，以保证应用和 OceanBase 数据库之间连接的稳定性，提升与数据库之间的交互性能

```
conn=jdbc:oceanbase://x.x.x.x(ip):xx(port)/xxxx(dbname)?rewriteBatchedStatements=true&allowMultiQueries=true&useLocalSessionState=true&characterEncoding=utf-8&socketTimeout=3000000
```

分类	参数名称	解释定义
网络连接	connectTimeout	● 定义连接超时时间，以毫秒为单位， 默认值：30000。
	socketTimeout	● 定义了网络 Socket 超时（SO_TIMEOUT），以毫秒为单位。值为 0 时将禁用此超时。
性能扩展	rewriteBatchedStatements	● 建议设置为true，使用addBatch方法把同一张表上的多条insert语句合在一起，做成一条insert语句里的多个values值的形式，提高batch insert的性能。 ● 注意：必须使用prepareStatement方式来把每条insert做prepare，然后再addBatch，否则不能合并执行
	allowMultiQueries	● 建议设置为True，JDBC驱动允许应用代码把多个SQL用分号（;）拼接在一起，作为一个SQL发给server端。
	useLocalSessionState	● 建议设置为true，避免交易频繁向OB数据库发送session变量查询SQL。 ● Tips：session变量主要为：autocommit，read_only和transaction isolation

Java 程序-OceanBase Connector/J 示例

```
String selectSql = "select host_ip() from dual;";
Connection conn = null;
Class.forName("com.oceanbase.jdbc.Driver").newInstance();
conn =
DriverManager.getConnection(“jdbc:oceanbase://172.28.15.107:2883/SYS?socketTimeout=3000
```

```
0&user=SYS@oracle_test#obcp_cluster&password=*******");
PreparedStatement ps = conn.prepareStatement(selectSql);
ResultSet rs = ps.executeQuery();
while (rs.next()){
   System.out.println(String.format("### %s ###", rs.getString("HOST_IP()")));
   }
ps.close();
export CLASSPATH=/home/obcp/oceanbase-client-2.4.0.jar:$CLASSPATH
javac TestJdbc.java
java TestJdbc
```

C 驱动程序

C 程序—OceanBase Connector/C 简介

OceanBase Connector/C 允许 C/C++ 程序以一种较为底层的方式访问 OceanBase 分布式数据库集群，以进行数据库连接、数据访问、错误处理和 Prepared Statement 处理等操作

OceanBase Connector/C 也称为 LibOBClient，用于应用程序作为独立的服务器进程通过网络连接与数据库服务器 OBServer 节点进行通信。支持 OceanBase 数据库的 MySQL 租户和 Oracle 租户

LibOBClient 安装步骤:

前提条件：GCC 版本为 3.4.6 及以上，推荐使用 4.8.5 版本、CMake 版本为 2.8.12 及以上

通过 github 获取 OceanBase Connector/C 安装包。下载地址：https://github.com/oceanbase/obconnector-c

安装 libobclient

sudo rpm -ivh libobclient-xx.x86_64.rpm

安装 obclient

sudo rpm -ivh obclient-xx.x86_64.rpm

C 程序—OceanBase Connector/C 示例

```
mysql_library_init(0, NULL, NULL);
MYSQL *mysql = mysql_init(NULL);
char* host_name = "172.28.15.107";//set your mysql host
char* user_name = "root@mysql_test#obcp_cluster"; //set your user_name
char* password = "******"; //set your password
char* db_name = "test"; //set your databasename
int port_num = 2883; //set your mysql port
.....
if (mysql_real_connect (mysql, host_name, user_name, password,
db_name, port_num, socket_name, CLIENT_MULTI_STATEMENTS) == NULL)
```

```
{
printf("mysql_real_connect() failed\n");
mysql_close(mysql);
exit(1);
}
status = mysql_query(mysql, "CREATE TABLE test_table(id INT,name varchar(24));");
status = mysql_query(mysql, "INSERT INTO test_table VALUES(10,'hangzhou'),(20,'shanghai');");
.....
mysql_close(mysql);
g++ -I/u01/obclient/include/ -L/u01/obclient/lib -lobclnt Test_Mysql.c -o Test_Mysql
export LD_LIBRARY_PATH=/u01/obclient/lib
./mysql_test
```

C 程序—OceanBase Connector/ODBC 简介

OceanBase Connector/ODBC 基于 MySQL ODBC 驱动进行定制开发的

开放数据库连接（Open Database Connectivity，ODBC）是为解决异构数据库间的数据共享而产生的，ODBC 为异构数据库访问提供统一接口，允许应用程序以 SQL 为数据存取标准，存取不同 DBMS 管理的数据；使应用程序直接操纵数据库中的数据，免除随数据库的改变而改变

OceanBase Connector/ODBC 驱动安装步骤

从官方网站下载 OceanBase Connector/ODBC 驱动相关的软件包

安装 RPM 软件包

rpm -ivh ob-connector-odbc-2.0.6-20221018193003.el7.alios7.x86_64.rpm

rpm -ivh libobclient-2.2.1-20221121105831.el7.alios7.x86_64.rpm

rpm -ivh ob-unixodbc-2.0.6-20221019102048.el7.alios7.x86_64.rpm

安装 Libtool 软件包

yum install libtool-ltdl-devel

C 程序—OceanBase Connector/ODBC 示例

配置 /etc/odbc.ini

```
[ODBC Data Sources]
data_source_name = odbctest
[odbctest]
Driver=Oceanbase
Description = MyODBC 5 Driver DSN
SERVER = 172.28.15.107
PORT = 2883
USER = root@test#obcp_cluster
Password = ******
Database = test
OPTION = 3
```

```
charset=UTF8
```

配置 /etc/odbcinst.ini

```
[Oceanbase]
Driver = /u01/ob-connector-odbc/lib64/mariadb/libobodbc.so
```

设置环境变量

```
export ODBCSYSINI=/etc
export ODBCINI=/etc/odbc.ini
export
LD_LIBRARY_PATH=/u01/mysql/lib:/usr/lib64:/u01/unixodbc/lib:/u01/obclient/lib:/u01/ob-
connectorodbc:$LD_LIBRARY_PATH
```

使用 odbcinst -j 命令查看配置是否正确

```
[root@obsvr01 obcp]# /u01/unix-odbc/bin/odbcinst -j
unixODBC 2.3.7
DRIVERS.....: /etc/odbcinst.ini
SYSTEM DATA SOURCES: /etc/odbc.ini
FILE DATA SOURCES..: /etc/ODBCDataSources
USER DATA SOURCES..: /etc/odbc.ini
```

ISQL 连接测试

```
[root@obsvr01 obcp]# /u01/unix-odbc/bin/isql -v odbctest
+---------------------------------------+
| Connected! |
| |
| sql-statement |
| help [tablename] |
| quit |
| |
+---------------------------------------+
```

程序示例

```
// Connect to data source
retcode = SQLDriverConnect(handler.hdbc, NULL,
(SQLCHAR*)"DSN=odbctest", SQL_NTS, connOut, MAX_NAME_LEN,
&len,SQL_DRIVER_NOPROMPT);
....
//insert
retcode = SQLPrepare(handler.hstmt, (SQLCHAR*)"INSERT INTO test
VALUES(?,'robin')", SQL_NTS);
.....
SQLINTEGER id = 2;
retcode = SQLBindParameter(handler.hstmt, 1, SQL_PARAM_INPUT,
SQL_C_LONG, SQL_INTEGER, 0, 0, &id, 0, NULL);
.....
retcode = SQLExecute(handler.hstmt);
.....
retcode = SQLEndTran(SQL_HANDLE_DBC, handler.hdbc, SQL_COMMIT);
```

```
gcc odbc_test.c -L/u01/unix-odbc/lib -lodbc -I/u01/unixodbc/include -oodbc_test
./odbc_test
```

C 程序—OBCI（OceanBase Call Interface）简介

OCI（Oracle Call Interface）是 Oracle 公司开发的一个应用程序开发工具，是一个通过访问 Oracle 数据库的服务器，通过控制各类 SQL 语句的执行，进而创建应用程序的的应用程序接口（API）

OBCI（OceanBase Call Interface）是与 Oracle OCI 兼容的 OceanBase C 语言接口调用工具，它提供了与 Oracle OCI 完全兼容的功能特性。支持 OceanBase 数据库的 Oracle 租户

OBCI 安装步骤:

前提条件

已部署 V2.2.76 或之后版本的 OceanBase 数据库（推荐使用 OceanBase 数据库 V4.1 或之后版本）

已部署 V1.8.7 或之后版本的 ODP

已安装 V2.2.6 或之后版本的 libobclient

已安装 Oracle 相关驱动文件 oracle-instantclient（从 OBCI V2.0.4 开始需要依赖 Oracle 头文件）

已部署 GCC 编译器

安装 libobclient 以及 oracle-instantclient 软件包

```
rpm -ivh libobclient-<version>86_64.rpm
rpm -ivh oracle-instantclient<version>-basic-<version>.x86_64.rpm
rpm -ivh oracle-instantclient<version>-devel-<version>.x86_64.rpm
```

安装 OBCI 软件包

```
rpm -ivh obci-<version>.x86_64.rpm
```

C 程序-OBCI（OceanBase Call Interface）示例

```
strcpy(strServerName, "172.28.15.107:2883");
strcpy(strUserName, "SYS@oracle_test#obcp_cluster");
strcpy(strPassword, "*******");
OCIServerAttach(srvhp, errhp, (text *)strServerName, (sb4)strlen(strServerName), OCI_DEFAULT);
OCIAttrSet(authp, OCI_HTYPE_SESSION, (text *)strUserName, (ub4)strlen(strUserName), OCI_ATTR_USERNAME, errhp);
OCIAttrSet(authp, OCI_HTYPE_SESSION, (text *)strPassword, (ub4)strlen(strPassword), OCI_ATTR_PASSWORD, errhp);
.....
static text* SQL_CREATE_TB = (text*)"create table person(personid number, name varchar(256), email varchar(256))";
OCIStmtPrepare(stmthp, errhp, SQL_CREATE_TB, strlen((char *)SQL_CREATE_TB),OCI_NTV_SYNTAX, OCI_DEFAULT);
OCIStmtExecute(svchp, stmthp, errhp, 1, 0, 0, 0, OCI_DEFAULT);
OCITransCommit(svchp, errhp, OCI_DEFAULT);
```

```
.....
OCISessionEnd(svchp, errhp, authp, (ub4)0);
OCIServerDetach(srvhp, errhp, OCI_DEFAULT);
OCIHandleFree((dvoid *)dschp, OCI_HTYPE_DESCRIBE);
OCIHandleFree((dvoid *)stmthp, OCI_HTYPE_STMT);
export LD_LIBRARY_PATH=/u01/obclient/lib:$LD_LIBRARY_PATH
gcc obci_oracle.c -I/u01/obclient/include -I/usr/include/oracle/11.2/client64/ -
L/u01/obclient/lib/ -
L/usr/local/lib64 -lobci -lobclnt -g -o obci_oracle
./obci_oracle
```

Python 驱动程序

OceanBase 数据库支持通过 Python 3.x 版本的 PyMySQL 驱动程序连接 MySQL 租户。PyMySQL 是在 Python3.x 版本中用于连接 MySQL 服务器的一个库，遵循 Python 数据库 API v2.0 规范，并包含了 pure-Python MySQL 客户端库。

PyMySQL 安装步骤

使用命令行安装

```
python3 -m pip install PyMySQL
```

源码编译安装

```
#注意，请根据 PyMySQL 的版本，选择对应的安装方式
git clone https://github.com/PyMySQL/PyMySQL
cd PyMySQL/
python3 setup.py install
```

程序示例

```
conn = pymysql.connect(host="172.28.15.107", port=2883,
user="root@mysql_test#obcp_cluster", passwd="******",
db="test")
cur = conn.cursor()
try:
#往 cities 表中插入两组数据
sql = "insert into cities values(1,'hangzhou'),(2,'shanghai')"
cur.execute(sql)
#查询 cities 表中的所有数据
sql = 'select * from cities'
cur.execute(sql)
ans = cur.fetchall()
print(ans)
finally:
cur.close()
conn.close()
#执行程序
python3 Test_PyMySQL.py
```

Go 驱动程序

GO 程序—MySQL 驱动（Go-SQL-Driver/MySQL）简介

Go-SQL-Driver/MySQL 驱动程序用于 Go 语言的 MySQL 数据库驱动程序。支持连接 MySQL 租户。

Go-SQL-Driver/MySQL 两种安装方式

方式 1: 通过 go get 安装（适用于 Go V1.13 - V1.16）

```
go get -u github.com/go-sql-driver/mysql
```

负载均衡

负载均衡原理

ODP 由多台节点组成的集群提供服务，用户请求具体会由哪个 ODP 节点来执行，由用户的 F5 或者其他负载均衡组件负责，同时 ODP 的某个节点故障，也应由负载均衡组件自动剔除，保证之后的请求不会再发送到故障节点上

负载均衡的工作原理

客户端向负载均衡设备（例如 F5）发起服务请求，该请求被发送到虚拟 IP 地址（VIP）

负载均衡 接收到请求后，将请求中的目的 IP 地址修改为选定的后端服务器 ODP 的地址，然后将数据包转发给 ODP

ODP 解析连接请求中的 SQL 信息，然后根据 OBServer 的状态以及用户配置的规则等信息， 将请求转发至选定的 OBServer 节点

OBServer 服务器接收到请求后，处理请求并生成响应数据包。服务器按照其路由规则将响应包发回给 ODP ，然后转发至负载均衡

负载均衡接收到响应包后，将响应包中的源 IP 地址改回 VIP 的地址，然后将数据包发送回客户端，完成一次标准的服务器负载均衡流程

负载均衡实现方式

负载均衡根据实现的方式，可以分为硬件负载均衡和软件负载均衡两种类型。在进行 OceanBase 集群部署时，建议采用硬件的负载均衡。

负载均衡在网络模型中的工作层次

四层负载均衡

实现基于 IP 地址+端口的负载均衡方式

分配用于 ODP 集群的 VIP

七层负载均衡

实现基于虚拟 URL 或主机 IP 的负载均衡方式

分配用于 OCP、ODC 等生态产品的 VIP

负载均衡设备需要开启源地址会话保持功能

负载均衡的实现方式

硬件负载均衡

专用的硬件设备进行负载均衡的工作

常见的硬件负载均衡包括：F5 Big-IP 、A10、以及国内相关厂家的设备

能够处理大量并发连接和高吞吐量，延迟低

软件负载均衡

OBLB：基于 iptables 的软负载均衡工具，可以为 ODP 集群配置 OBLB 以提升集群的负载均衡能力

OBDNS：基于容器部署的 DNS + Tengine 的负载均衡服务。可以为 OCP 等生态产品提供负载均衡和高可用能力

NLB：基于 OpenResty 的负载均衡器，以其高可用性、简单部署和低资源消耗为特点。NLB 为 OceanBase 生态产品如 OCP、ODC 和 OMS 提供负载均衡服务

OceanBase Connector/J 的 LoadBalance 模式：通过从 JDBC 连接的 URL 中，随机选择一个 ODP 主机来进行负载均衡

JDBC 负载均衡

OceanBase Connector/J 支持 LoadBalance 功能，通过 URL 或配置文件的方式，用户可指定以随机、加权或轮询方式，达到负载均衡建立连接的目的。

```
String url ="jdbc:oceanbase:loadbalance://172.28.15.107:2883:1,172.28.15.108:2883:1,172.28.15.109:2883:1/test?user=root@mysql_test#obcp_cluster&password=******&loadBalanceStrategy=serverAffinity" ;Connection = DriverManager.getConnection(url);
```

负载均衡策略：URL 实现方式中的 loadBlanceStrategy 配置项，有三种负载均衡策略：

算法	特性	备注
RANDOM	随机	默认算法
SERVERAFFINITY	加权	加权随机
ROTATION	轮询	轮询

JDBC LoadBalance 程序示例

```
String url =
“jdbc:oceanbase:loadbalance://172.28.15.107:2883:1,172.28.15.108:2883:1,172.28.15.109:2883:1/test?user=root@my
sql_test#obcp_cluster&password=******&loadBalanceStrategy=serverAffinity" ;
String selectSql = "select host_ip() from dual;";
Connection conn = null;
```

```
。 。 。 。 。 。
Class.forName("com.oceanbase.jdbc.Driver");
conn = DriverManager.getConnection(url);
PreparedStatement ps = conn.prepareStatement(selectSql);
ResultSet rs = ps.executeQuery();
while (rs.next()){
System.out.println(String.format("### %s ###", rs.getString("HOST_IP()")));
}



[root@obsvr01 obcp]# java -cp .:oceanbase-client-2.4.0.jar TestLB
### 172.28.15.109 ###
[root@obsvr01 obcp]# java -cp .:oceanbase-client-2.4.0.jar TestLB
### 172.28.15.108 ###
[root@obsvr01 obcp]# java -cp .:oceanbase-client-2.4.0.jar TestLB
### 172.28.15.107 ###
[root@obsvr01 obcp]# java -cp .:oceanbase-client-2.4.0.jar TestLB
### 172.28.15.109 ###
```

OBProxy 部署形态

ODP 概述

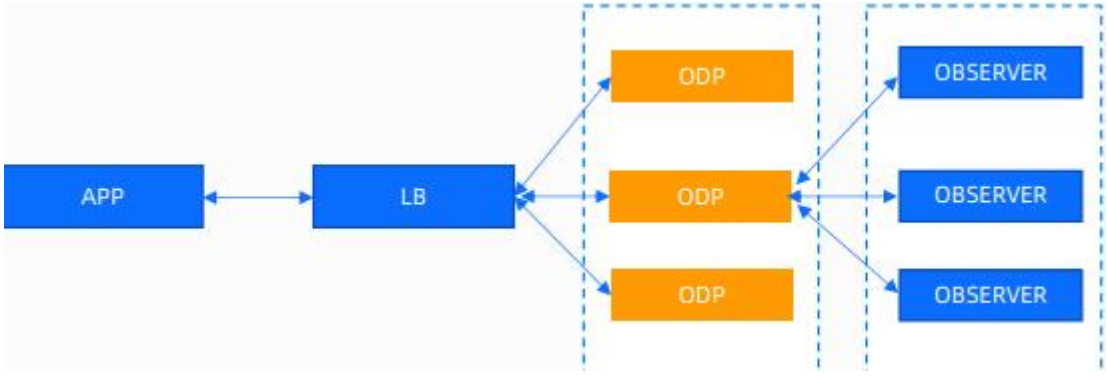
OBProxy（OceanBase Database Proxy，ODP） 概述

OceanBase Database Proxy（简称 ODP）

OceanBase 数据库专用的服务代理。它让用户像使用单机一样使用 OceanBase 数据库

用户访问数据库的代码无需改动就可以访问 OceanBase 数据库

用户无需关心一些分布式相关问题：如机器的添加删除、机器宕机、多租户的 unit 分布、每日合并等



ODP 特性

高性能转发：ODP 完整兼容 MySQL 协议，并支持 OceanBase 自研协议，采用多线程异步框架和透明流式转发的设计，保证了数据的高性能转发，同时确保了自身对机器资源的最小消耗。

最佳路由：充分考虑用户请求涉及的副本位置、用户配置的读写分离路由策略、OceanBase 数据库多地部署的最优链路，以及 OceanBase 数据库各机器的状态及负载情况，将用户的请求路由到最佳的 OBServer 节点上。

连接管理：针对一个客户端的物理连接，ODP 维持自身到后端多个 OBServer 节点的连接。采用基于版本号的增量同步方案，维持了每个 OBServer 节点连接的会话状态， 确保对 OBServer 节点的高效访问

专有协议：ODP 与 OBServer 节点默认采用了 OceanBase 专有协议：实现了报文的 CRC 校验，以确保和 OBServer 节点链路的正确性；增强了传输协议，以支持 Oracle 兼容性的数据类型和交互模型

安全可信：ODP 支持使用 SSL 协议对数据进行访问，并和 MySQL 协议做了兼容，满足客户对数据访问的安全合规需求

易运维：ODP 本身无状态，支持无限水平扩展， 支持同时访问多个 OceanBase 集群。同时也可以通过丰富的内部命令实现对自身状态的实时监控，提供极大的运维便利性

ODP 部署与启动

OBProxy（ODP）有两种安装方式：OCP 图形化安装、RPM 命令行安装。另外，ODP 在启动时根据获取集群信息方式的不同，可以分为依赖 Config Server 和依赖 RootServer 启动两种方式

部署方式（图形化部署）

建议企业版使用 OCP 进行环境部署

建议一台机器上只部署一个 ODP 服务，并且使用约定的 2883 端口

部署方式（命令行部署）

在未未安装 OCP 场景时，可以使用命令行安装 ODP 的 RPM 包 ，并进行 ODP 集群的配置

启动方式

依赖 Config Server 启动：在启动命令中指定 obproxy_config_server_url 参数项来查询获取 OceanBase 集群的 RootServer 信息。该方式需要配置 obproxy_config_server_url，故会依赖 Config Server 的启动。推荐使用该方式启动 ODP

```
./bin/obproxy -p 2883 -n test -o
obproxy_config_server_url='http://172.28.15.106:8080/services?User_ID=a
libaba&UID=test
```

依赖 RootServer 启动：在启动命令中指定 -r 参数来指定 OceanBase 集群的 RootServer 信息，该启动方式不需要额外配置，一般用于开发调试阶段

```
./bin/obproxy -p 2883
-r '172.28.15.106:2881;172.28.15.107:2881;172.28.15.108:2881' -n test -
c obproxy_cluster .....
```

ODP 部署方式

结合实际生产系统的物理环境和业务需求，ODP 集群在部署时有以下三种方式：

OBServer 端部署：将 ODP 集群部署在服务器端，是专有云常见的部署形态

独立部署：将 ODP 集群部署在独立的主机之上，目前公有云使用了该部署方式

客户端部署：将 ODP 安装在客户端服务器之上

OBServer 端部署介绍

部署在 OBServer 端是指在部署 OceanBase 数据库的机器上部署一个 ODP 进程，这样 OBServer 节点和 ODP 的数量满足 1:1 关系。此种方式下，ODP 数量和 APP 没有了对应关系。除了一个节点部署一个 ODP 这种情况外，也可以一个 Zone 内部署一个 ODP。

部署说明

LB 实现 ODP 的高可用和负载均衡

ODP 和 OBSERVER 部署在一台机器上（当压力很大时会存在资源竞争）

问题排查

APP 直接连接 OBSERVER，确认 OBSERVER 是否有问题

去掉 LB 确认是否 LB 有问题（常用的 F5 等一般问题都已经解决，主要是一些小众的 LB）

LB 后端只挂一个 ODP 或者直连 ODP，方便排查 ODP 问题

独立部署介绍

独立部署是指专门为 ODP 找一台机器部署。此时 ODP 的数量和 APP、OBServer 节点都没有关系，根据具体业务需求确定 ODP 的数量

部署说明

对于 ODP 的部署机型，一般推荐选择小机型，ODP 本身资源消耗较少，选择 4c/8c/16c/32c 的配置即可独立部署

OceanBase 数据库和 ODP 之间不存在资源抢占，可以更好地管理 ODP，将 ODP 做成资源池对外服务，目前公有云使用了该部署方式

客户端部署介绍

ODP 和 APP 应用服务器部署在同一个主机之上，APP 和 ODP 的数量满足 1:1 关系。ODP 和 OceanBase 数据库直连，中间没有负载均衡。

部署说明

当应用采用容器化部署时，由于 APP 和 ODP 的个数是 1:1 对应关系，这种部署方式会导致 ODP 数量较多。为了解决这个问题，可以在每个 APPServer 容器旁运行一个 ODP 容器，并将两者放在同一个 Pod 中。这种方式可以实现独立的升级、发布和运维操作

组织相关：这种形式涉及到业务和运维两个团队，需要全局把控

架构依赖：在客户成熟的业务服务器上，增加进来一个新的进程，可能会影响客户业

务的管理

OBProxy 连接管理

连接管理

连接管理概述

OBProxy（ODP） 为用户提供了数据库接入和路由功能，直接连接 ODP 就可以正常使用 OceanBase 数据库。

连接映射关系

当通过 ODP 与 OceanBase 数据库建立连接时，客户端与 ODP 之间存在一个物理连接，而 ODP 与 OBServer 节点间可能存在多个物理连接。其中 Client 与 ODP 的连接称为客户端连接，ODP 与 OBServer 节点间的连接称为服务端连接。

当客户端所访问的数据存在于不同 OBServer 节点上时，ODP 会与 OceanBase 数据库建立多个物理连接并管理复用这些连接，在客户端视角看来仅存在一个逻辑连接。ODP 基于此可以提供多种功能特性，比如主备库分离、读写分离、分区表数据路由、分布式 PS、屏蔽后端异常等等

连接管理与高可用机制

连接控制

白名单：控制客户端的访问白名单，目前该能力做到了租户级别，可以对每个租户单独设置

连接数配置：控制客户端连接 ODP 的连接数量

用户访问控制

root@proxysys 用户：是 ODP 的管理员,可以查询 ODP 内部状态，修改配置等

proxyro (Proxy Read Only)用户： 是 OceanBase 集群 sys 租户下的一个只读用户，拥有 OceanBase 数据库下所有虚拟表和视图的只读权限，ODP 使用该用户访问租户 Unit 分布、分区副本（或日志流）分布等信息

其他用户：通过“用户名@租户名#集群名”的方式访问业务租户及系统租户

高可用机制

TCP 探活：通过设置 TCP 的 no_delay 和 keepalive 探活属性，以及时发现连接是有效还是无效

故障探测：ODP 周期性的主动探测 OBServer 节点的状态，以及时发现故障的 OBServer 节点

黑名单：探测到 OBServer 故障后，将该节点加入黑名单，路由时选择其他健康节点

连接监控

客户端连接监控：支持对客户端至 ODP 之间的连接监控和管理

服务端连接监控：支持对 ODP 至 OBServer 之间的连接监控和管理

连接控制-白名单

OceanBase 数据库集群目前支持：ODP 级白名单、租户级白名单。

ODP 级白名单： 可以控制客户端访问 ODP 的白名单，目前该能力做到了租户级别，即在 ODP 中，控制访问租户的白名单。

租户级白名单： 在 OceanBase 集群中，设置允许哪些客户端访问当前租户。

连接控制-白名单举例

ODP 级白名单：在 root@proxysys 账户下，ODP 提供了内部表 white_list，里面保存了白名单信息，目前 ODP 支持白名单为 IP 或者 IP 和网络号，白名单功能在公有云上已经广泛使用。示例：ODP 中设置访问租户 mysql_test 的白名单

```
obclient[(none)]> replace into white_list(cluster_name, tenant_name, name,
value) values('obcp_cluster', 'mysql_test', 'ip_list',
'127.0.0.1,172.28.15.0/24');
Query OK, 0 rows affected (0.00 sec)
MySQL [(none)]> select cluster_name, tenant_name, name, value from
white_list;
+--------------+-------------+---------+-------------------------+
| cluster_name | tenant_name | name | value |
+--------------+-------------+---------+-------------------------+
| obcp_cluster | mysql_test | ip_list | 127.0.0.1,172.28.15.0/24 |
+--------------+-------------+---------+-------------------------+
```

租户级白名单：在租户的管理员账户下，通过系统变量 ob_tcp_invited_nodes 控制，是租户全局的白名单限制参数。默认值为 127.0.0.1,::1，表示仅允许本机的 IP 连接该租户

```
obclient [(none)]> SHOW VARIABLES LIKE 'ob_tcp_invited_nodes';
+----------------------+-------+
| Variable_name | Value |
+----------------------+-------+
```

| ob_tcp_invited_nodes | % |
+----------------------+-------+
1 row in set (0.01 sec)
MySQL [(none)]> SET GLOBAL
ob_tcp_invited_nodes='127.0.0.1,172.28.15.0/24';
Query OK, 0 rows affected (0.07 sec)

连接控制—连接数配置

OceanBase 数据库集群目前支持在 ODP、租户级别设置允许连接的最大连接数。

ODP 连接数参数配置

client_max_connections：单个 ODP 支持的最大客户端连接数,默认值为 8192

查看和修改连接参数（sys 租户或 proxysys 租户）：

```
show proxyconfig like '%max_connections%'\G
alter proxyconfig set client_max_connections=10000
```

租户级连接数配置

max_connections：系统变量，用于控制整个租户的最大连接数，默认值为 2147483647。仅适用于 MySQL 租户

max_user_connections：系统变量，用于控制租户内单个用户的最大并发连接数，默认值为 0，表示不限制连接数。仅适用于 MySQL 租户

_resource_limit_max_session_num：租户级隐藏配置项，用于控制用户租户内普通用户的最大并发连接数，该配置项修改后立即生效，不需要重启 OBServer 节点，其默认值为 0。适用于 MySQL 和 Oracle 租户

（注：对于某个普通用户，其并发连接数只要达到其中一个值，即无法再建立新的连接）

高可用机制—TCP 探活机制原理

探活机制：采用“轮询”的方式，定义一个时间段，在这个时间段内，如果没有任何连接相关的活动，TCP 保活机制会开始作用，每隔一个时间间隔，发送一个探测报文，如果连续几个探测报文都没有得到响应，则认为当前的 TCP 连接已经断开

高可用机制—ODP 的 TCP 探活机制

ODP 使用基于内核 TCP 协议栈的 keepalive 属性进行探活，以及时发现 OBServer 的故障，包含以下机制：

NO_DELAY 属性：通过禁用 TCP Nagle 算法解决延迟问题。Nagle 算法的设计目的是通过将小的数据包聚合成较大的数据包来减少网络上的小包数量，但这会引入延迟。禁用 Nagle 算法后，数据将被立即发送

KEEPALIVE 属性：用于故障探测。及时发现机器故障，将无效的连接关闭，是 TCP 层高可用一部分

参数名称	参数定义	详细描述

sock_option_flag_out/client_sock_option_flag_out	ODP 和 OBServer/ 客户端之间的 TCP 参数	1 表示启用 no_delay，2 表示启用 keepalive，3 表示启用 no_delay 和 keepalive 推荐 ODP 与客户端的 TCP 参数为 no_delay+keepalive，即 client_sock_option_flag_out=3 推荐 ODP 与 OBServer 的 TCP 参数为 no_delay+keepalive，推荐设置为 0，不做 ACK 确认超时处理，避免因为网络不稳定导致的意外断连即 sock_option_flag_out=3
server_tcp_user_timeout/client_ tcp_user_timeout	等待 TCP 层 ACK 确认消息的超时时长	推荐设置为 0，不做 ACK 确认超时处理，避免因为网络不稳定导致的意外断连

高可用机制—TCP 探活机制配置样例

TCP 连接参数：client 端

```
#启用 KEEPALIVE，不启用 TCP NO_DELAY
alter proxyconfig set client_sock_option_flag_out = 3;
#启动 KEEPALIVE 探活前的 idle 时间，5 秒
alter proxyconfig set client_tcp_keepidle = 5;
#两个 KEEPALIVE 探活包的发送时间间隔，5 秒
alter proxyconfig set client_tcp_keepintvl = 5;
#最多发送的 KEEPALIVE 探活包个数，2 个
alter proxyconfig set client_tcp_keepcnt = 2;
#等待 TCP 层 ACK 确认消息的超时时长，0 表示不做超时处理
alter proxyconfig set client_tcp_user_timeout = 0;
```

TCP 连接参数：server 端

```
#启用 TCP NO_DELAY 和 KEEPALIVE
alter proxyconfig set sock_option_flag_out = 3;
#启动 KEEPALIVE 探活前的 idle 时间，5 秒
alter proxyconfig set server_tcp_keepidle = 5;
#两个 KEEPALIVE 探活包的发送时间间隔，5 秒
alter proxyconfig set server_tcp_keepintvl = 5;
#最多发送的 KEEPALIVE 探活包个数，2 个
alter proxyconfig set server_tcp_keepcnt = 2;
#等待 TCP 层 ACK 确认消息的超时时长，0 表示不做超时处理
alter proxyconfig set server_tcp_user_timeout = 0;
```

高可用机制—故障探测及黑名单

ODP TCP 的探活机制只能探测 TCP 连接的状态，但是无法感知 OBServer 合并、升级、leader 切换、宕机、启动/停止等过程中的状态，为避免业务流量被硬件异常和内部流量影响，ODP 需要能够自适应处理这些过程中的访问控制

ODP 运行过程中会通过实时读取集群系统表或者通过发送探测 SQL 等手段，了解每个数据库节点的健康状态和分区的实时位置。当探测到 OBServer 不可用时，就将 OBServer 加入黑名单中，黑名单中的 OBServer 将被过滤不再访问，直到某次刷新后被洗白才恢复访问

根据不同的探测机制，ODP 实现了状态黑名单、探测黑名单和活不可用黑名单三种不同的黑名单

高可用机制—黑名单的工作机制

黑名单类型	功能描述	局限性
状态黑名单	依赖于 OBServer 节点的状态变更，周期性（20s/次）从 DBA_OB_SERVERS 和 DBA_OB_ZONES 视图中获取集群的机器状态信息	从 OceanBase 的总控服务节点 RS 处获得信息，这种信息有时不能反映 ODP 和 OBServer 节点之间的情况。例如：ODP 从 RS 处获得 OBServer1 节点状态为 ACTIVE，但 ODP 和 OBServer1 节点之间网络不通
探测黑名单	ODP 定时给 OceanBase 数据库的 sys 租户发送探测 SQL，并设置 5s 的超时时间，当连续超过 3 次探测失败，就设置 OBServer 节点状态为 DETECT_DEAD	有了状态黑名单和探测黑名单，ODP 已经可以很好地处理机器和进程故障了。为了使 ODP 可以更近一步感知每个 SQL 执行失败的原因，去处理更复杂的情况，因此实现了活不可用名单
活不可用黑名单	活不可用名单机制的核心是根据业务 SQL 的执行结果去定义 OBServer 的状态，触发拉黑和洗白操作	

高可用机制—故障探测及黑名单（查看黑名单）

黑名单查看步骤

在客户端中通过 ODP ，登录集群的 sys 租户

obclient -h172.28.15.107 -P2883 -uroot@sys#obcp_cluster –p******

执行以下命令，查看黑名单信息

SHOW PROXYCONGESTION [all] [clustername];

通过 OCP 停止 OBServer 的进程后，查看黑名单：

通过OCP停止OBServer的进程后，查看黑名单：

```
MySQL [(none)]> show proxycongestion 'obcp_cluster'\G
*************************** 1. row ***************************
        cluster_name: obcp_cluster
           zone_name: zone3
         region_name: BeiJing
          zone_state: ACTIVE
           server_ip: 172.28.15.109:2881
          cr_version: 2
        server_state: INACTIVE
     alive_congested: 0
last_alive_congested: 0
      dead_congested: 1
 last_dead_congested: 2024-08-06 15:44:38
 stat_alive_failures: 2
  stat_conn_failures: 0
 conn_last_fail_time: 0
 conn_failure_events: 0
alive_last_fail_time: 2024-08-06 15:44:38
alive_failure_events: 2
           ref_count: 2
    detect_congested: 0
last_detect_congested: 0
1 row in set (0.00 sec)
```

sever_state：表示OBServer的状态。

相关参数说明如下：

不指定可选项时，可查询所有集群的黑名单信息

仅指定 all 时，可显示所有集群的 OBServer 节点信息（包括黑名单中和非黑名单中的 OBServer 节点信息）

仅指定 clustername 时，可显示指定集群的黑名单信息

同时指定 all 和 clustername 时，可显示指定集群的 OBServer 节点信息

连接监控—客户端连接监控

客户端连接监控

一个 ODP 集群本质上是一个个独立的 ODP 进程，多个 ODP 之间的连接是互不感知的，组成集群只是为了方便进行运维管理和参数的修改

ODP 看到的连接是客户端至 ODP 之间的连接，而 OBServer 看到的连接是 ODP 至 OBServer 之间的连接，是两段不同的 TCP 连接

客户端连接监控：通过 ODP IP+端口（默认的端口号：2883）的方式连接 sys 租户，以查看某个 ODP 上所有的连接

方法 1: show processlist

方法 2: show proxysession

注：上述两种查看客户端连接方式的功能类似，show proxysession 是 ODP 专属命令。

连接监控—服务端连接监控

服务端连接监控

SHOW PROCESSLIST 语句的显示结果与连接数据库的方式有关。当通过 ODP 连接数据库时，显示的是对应的 ODP 节点上的会话信息；当通过直连方式连接 OBServer 时，显示的是租户的所有服务端会话信息

查看租户会话时，租户管理员可以查看当前租户内的所有会话信息。sys 租户可以查看集群范围内的所有连接

方法 1: 直连 OBServer，以租户管理员执行 show processlist

方法 2: 直连 OBServer 或者通过 ODP，以租户管理员执行 show full processlist

方法 3: 通过系统视图查看

MySQl 租户：SELECT * FROM oceanbase.GV$OB_PROCESSLIST

Oracle 租户：SELECT * FROM SYS.GV$OB_PROCESSLIST;

重要字段说明如下：

ID：表示该会话的 ID

HOST：表示发起该会话的客户端 IP 及端口号。如果是通过 ODP 连接的数据库，则表示 ODP 的主机 IP 及端口号

COMMAND：表示该会话正在执行的命令类型

STATE：表示该会话当前的状态

连接监控—OCP 管理租户会话

通过 OCP，也可以对租户的服务端连接进行监控和管理。

方式 1:登录 OCP，单击“自治服务”，在“集群详情”区域，单击需要查看的集群名称，进入“实时诊断”的“会话管理”页签

方式 2:登录 OCP，进入租户“概览”页面。在左侧单击“会话管理”，系统默认进入“租户会话”列表页面

连接监控—客户端连接和服务端连接的映射关系

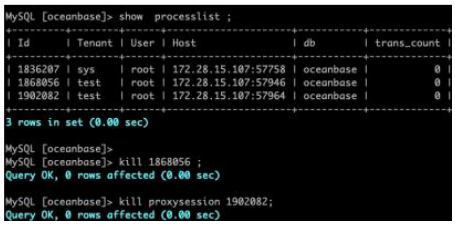
客户端与 ODP 之间的一个物理连接，可能会对应多个 ODP 与 OBServer 节点间的物理连接。 通过 ODP 登陆某个租户后，查看映射关系的步骤如下

连接监控—终止客户端连接

对于系统中存在的异常情况或性能问题，我们可以通过终止客户端连接或者服务端连接的手段快速的隔离和处理故障，以维护系统的稳定性和安全性。

终止客户端连接

1. 通过某个ODP连接sys租户。
2. 执行kill cs_id 或者 kill proxysession cs_id命令终止某个客户端连接。
   - SHOW PROXYSESSION 语句输出中的 id 即为 cs_id。
   - kill 命令用于断开一个客户端连接，客户端连接关闭后，ODP 也会关闭对应的服务端连接。

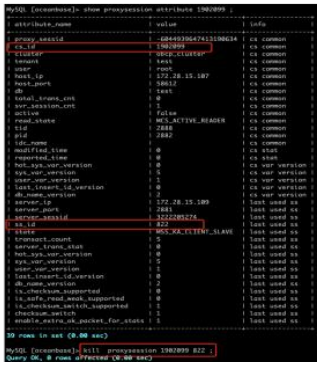


连接监控—终止服务端连接

对于系统中存在的异常情况或性能问题，我们可以通过终止客户端连接或者服务端连接的手段快速的隔离和处理故障，以维护系统的稳定性和安全性。

通过 ODP 终止服务端连接

1. 通过某个ODP连接sys租户。
2. show proxysession attribute cs_id 获取客户端和服务端连接的映射关系。
3. kill proxysession cs_id ss_id 语句终止指定的后端数据库连接。
   - ss_id 为 ODP 内部标记每个 Server 端会话（Server Session）的 ID 号



OBProxy 路由管理

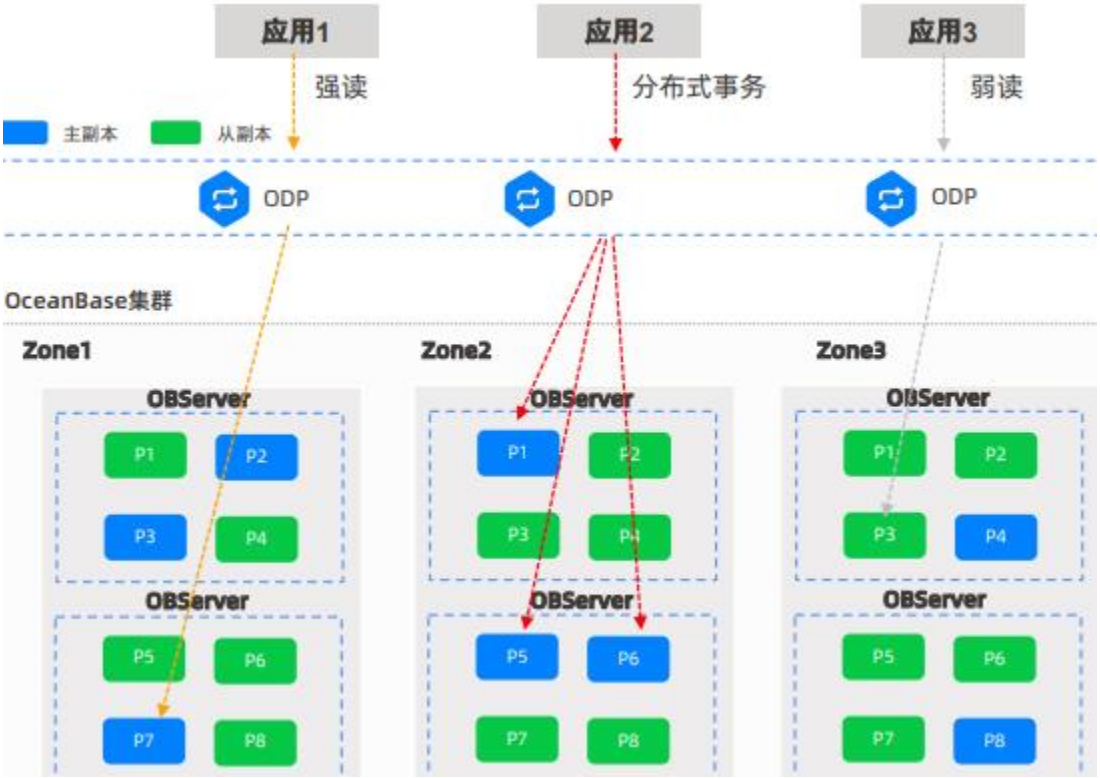
路由概述

路由原理

OceanBase 数据库的用户数据以多副本的形式存放在各个 OBServer 节点上，ODP 可以获取到 OBServer 节点中的数据分布信息，将用户 SQL 高效转发到数据所在机器，提升执行效率。

ODP 可以获取到 OBServer 节点中的数据分布信息，可以将用户 SQL 高效转发到数据所在机器，执行效率更高

最佳路由 ODP 会充分考虑用户请求涉及的副本位置、用户配置的读写分离路由策略、OceanBase 数据库多地部署的最优链路，以及 OceanBase 数据库各机器的状态及负载情况，将用户的请求路由到最佳的 OBServer 节点上，最大程度的保证了 OceanBase 数据库整体的高性能运转



路由功能分类

在分布式数据库系统中，面对海量数据和多租户需求，确保系统的高可用性、性能和资源隔离性至关重要。ODP 为了有效管理多集群、多租户以及租户内部的数据分布，实现了集群路由、租户路由和租户内路由三种关键的路由策略，以应对复杂的分布式环境挑战。

集群路由：ODP 用于在多个集群之间选择合适的集群来处理请求，且能够根据配置的

策略，将请求分配到不同的集群上，从而实现更好的资源利用和更高的系统稳定性

租户路由：OceanBase 支持多租户架构，每个租户相当于一个独立的数据库实例，具有自己的资源和权限。租户路由用于将请求路由到正确的租户上，以确保不同租户的数据和操作彼此隔离

租户内路由：在一个租户内，数据可能分布在多个物理节点上。租户内路由用于将请求路由到租户内的正确节点或数据分片上，以确保请求能够正确访问和操作数据

集群简介

集群路由是指 ODP 路由功能支持访问不同的集群，它的关键点在于获取集群名和 rslist 的映射关系

对于启动参数指定 rslist （RootServer 总控服务所在的机器）的启动方式，集群名和 rslist 的映射关系通过启动参数指定

对于启动参数指定 obproxy_config_server_url 的启动方式，集群名和 rslist 的映射关系通过访问 url 获取

ODP 是在用户登录首次访问集群时获取 rslist，并保存到内存中，后续再访问该集群，从 ODP 的内存中获取就可以了

ODP 路由功能支持访问不同的集群，所以在命令行或者 ODC 中通过 ODP （假 设 ODP 的 i p 和 port 分别为 172.28.15.106 和 2883）进行连接时，除了需要指定用户名 user_name 和租户名 tenant_name 外，还需要额外指定要连接的集群名 cluster_name

租户路由简介

OceanBase 数据库中，一个集群有多个租户，租户路由是指 ODP 路由功能支持访问不同的租户。在众多租户中，sys 租户比较特殊，类似于管理员租户，和集群管理相关。因此，租户路由根据租户的类型分为 sys 租户路由、普通租户路由两种。

sys 租户路由：

ODP 完成集群路由后可获得集群的 rslist，此时 ODP 会通过 proxyro@sys 账号登录 rslist 中的一台机器，查询系统视图 DBA_OB_SERVERS，获取集群的所有机器节点。返回的结果也就是 sys 租户的路由信息

ODP 会每 15 秒访问一次 DBA_OB_SERVERS，维护最新的路由信息，这样可以感知到

集群发生的节点变更

普通租户路由：

ODP 通过 sys 租户获得元数据信息，然后通过元数据信息获得租户的路由信息。普通租户路由信息是租户资源所在的机器

ODP 通过表 __all_virtual_proxy_schema 查询租户下的__all_dummy 表的副本位置来获取租户的节点列表并缓存起来

租户内路由实现了根据 OBServer 的数据分布精准访问到数据所在的机器。同时还可以根据一定的策略将一致性要求不高的读请求发送给副本机器，充分利用机器的资源。路由选择输入的是用户的 SQL、用户配置规则、和 OBServer 状态，路由选择输出的是一个可用 OBServer 地址。ODP 进行租户内路由的过程大致如下：

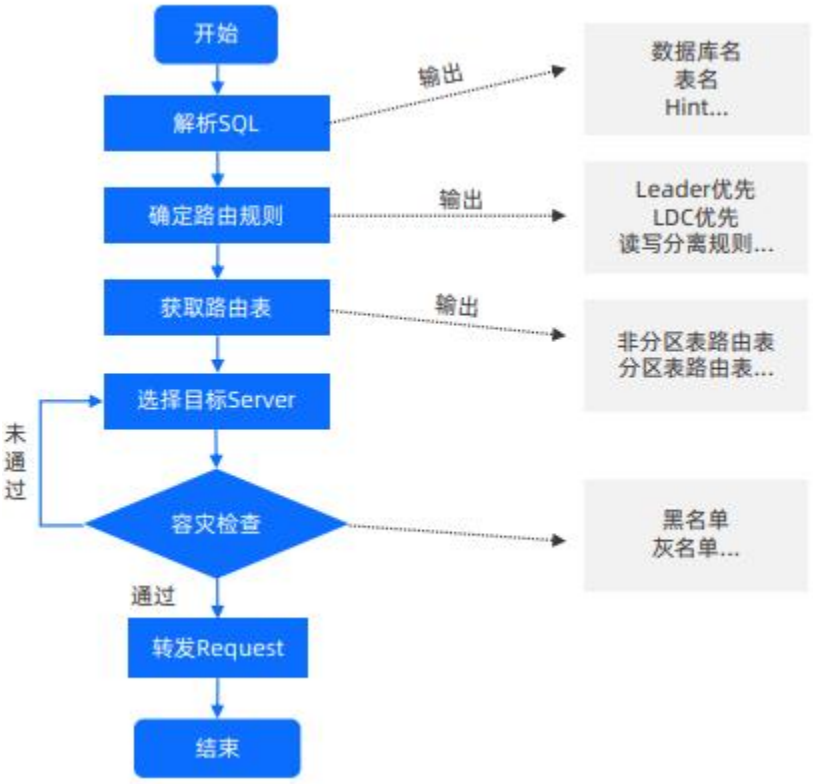
解析 SQL 模块：使用的是 ODP 自己定制的 Parser 模块，只需要解析出 DML 语句中的数据库名、表名和 Hint，不需要通过其他复杂的表达式推演

确定路由规则：ODP 需要根据不同情况确定最佳的路由规则，例如强一致性读、弱一致性读等

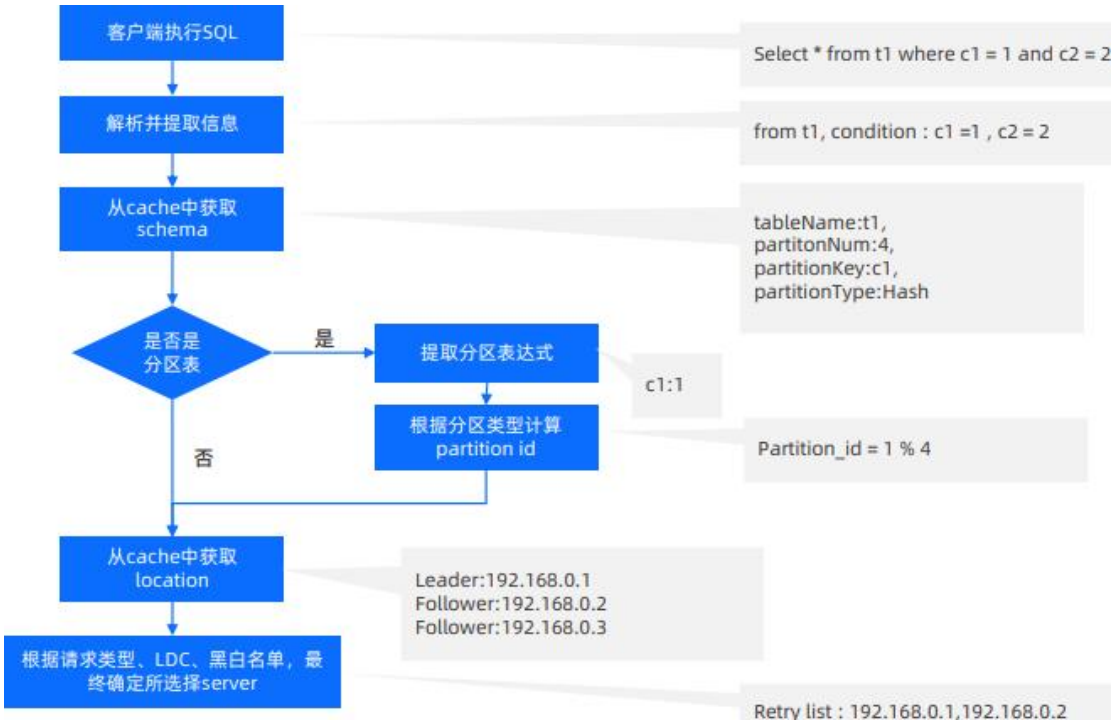
获取路由表： ODP 根据用户的请求 SQL 获取该 SQL 涉及的副本位置

选择目标 OBServer： 根据确定的路由规则从上一步获取的路由表中选择最佳的 OBServer 节点

容灾检查：在经过黑名单、灰名单检查通过后作为最终的目标 OBServer 节点进行请求转发



租户内路由—功能举例



租户内路由—路由策略

数据库系统在应用架构中承担了数据存储和查询的功能，为应用提供读写的服务。

写请求：ODP 会将写请求路由到表分区的主副本所在节点

读请求：根据请求数据的一致性要求，分为强一致性读和弱一致性读

强一致性读：由 OceanBase 数据库的 Leader 副本提供服务，适用于对读写一致性要求高的场景。ODP 默认使用此种路由策略

弱一致性读：由 Leader 副本和 Follower 副本提供服务

租户内路由—强一致性读（分区表路由）

对于非分区表，表只有一个分区，ODP 根据表名就可以获得副本位置信息。而对于分区表，ODP 需要将 SQL 请求中的分区键值或表达式、分区名等条件解析为分区 ID，通过分区 ID 查找到分区的副本位置，将请求路由到 Leader 副本。 示例中，T 表分区键为 C1，查询条件中可以指定：分区键键值（C1 = xxxx）、分区键列表达式（C1 = ABS(xxxx)）或者指定分区名（PARTITION(P0) ）。

登录 OceanBase 数据库，执行如下命令创建分区表

```
CREATE TABLE T(C1 INT) PARTITION BY HASH(C1)PARTITIONS 8;
```

使用 EXPLAIN ROUTE 命令查看 ODP 路由选取的过程

```
EXPLAIN ROUTE SELECT * FROM T WHERE C1=3;
```

1. 登录 OceanBase 数据库，执行如下命令创建分区表。

```
CREATE TABLE T(C1 INT) PARTITION BY HASH(C1)
PARTITIONS 8;
```

2. 使用 EXPLAIN ROUTE 命令查看 ODP 路由选取的过程。

```
EXPLAIN ROUTE SELECT * FROM T WHERE C1=3;
```

当查询条件不含分区键，但包括全局索引列时，ODP该如何路由呢？

租户内路由—强一致性读（全局索引表路由）

当查询分区表时，如果查询条件不包含分区键，但包括全局索引时，ODP 会将 SQL 语句中提供的索引值作为分区键来计算路由。通过全局索引路由将查询请求精确定位到目标数据所在的分区，从而避免扫描所有分区，提高查询速度。ODP 计算出全局索引的 partition id，把 SQL 转发到全局索引所在的 OBServer。

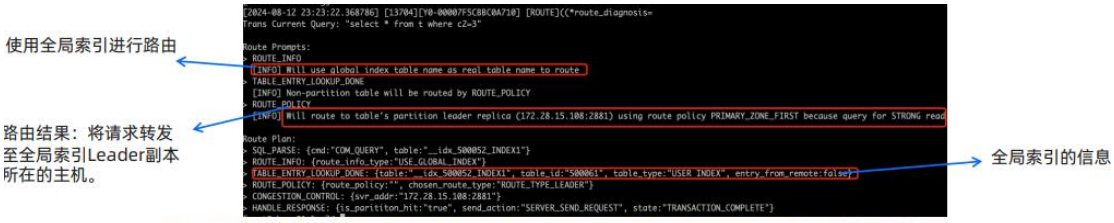
开启全局索引表路由（sys 租户）

```
ALTER PROXYCONFIG SET enable_reroute = true;
ALTER PROXYCONFIG SET enable_index_route = true;
ALTER PROXYCONFIG SET server_routing_mode = 'oceanbase';
ALTER PROXYCONFIG SET proxy_primary_zone_name ='';
```

全局索引表路由示例（普通租户）

```
CREATE TABLE T(C1 INT,C2 INT) PARTITION BY HASH(C1)
PARTITIONS 8;
CREATE INDEX INDEX1 ON T(C2) global ;
SELECT * FROM T WHERE C2=3;
```

ODP 诊断日志中的路由信息（修改配置项 route_diagnosis_level=4，打开路由的详细诊断信息）：



当ODP不能通过SQL请求中的条件计算出准确的分区位置信息时，该如何路由呢？

租户内路由—强一致性读（租户的 Primary Zone 路由）

默认情况，ODP 会将请求发送到租户的 primary zone 所在的机器上。但当 ODP 找不到表或者分区 Leader 副本时（分区计算失败、无法获取表名等原因），ODP 会通过租户的 Primary Zone 尽量将请求发往主副本所在的主机。

```
CREATE TABLE T1(C1 INT,C2 INT,C3 INT) PARTITION BY HASH(C1) PARTITIONS 8;
```

```
EXPLAIN ROUTE select * from t1 where c3=6 \G
```

租户内路由—强一致性读（复制表路由）

复制表是 OceanBase 数据库的一种特殊表。这种表可以在任意一个“健康”的副本上读取到数据的最新修改。对于写入频率要求较低、读操作延迟和负载均衡要求较高的用户来说，复制表是一种很好的选择。ODP 会将查询请求随机路由到复制表的任意一个副本。

创建测试的复制表

CREATE TABLE DUP(C1 INT) DUPLICATE_SCOPE = 'cluster';

执行 EXPLAIN ROUTE 命令查看 ODP 路由选取过程

EXPLAIN ROUTE SELECT * FROM test.dup WHERE C1=123\G

租户内路由—强一致性读（总结）

租户内路由—弱一致性读概述

和强一致性读对立的就是弱一致性读，弱一致性读不要求写后读立即可见。弱一致性读可以路由到分区的主副本和备副本节点。常见的弱一致性读设置有三种方式：会话级别设

置、SQL 级别设置、ODP 级别设置。

会话级设置

会话级别设置，通过 ob_read_consistency 系统变量指定,分为 Global 级别和 Session 级别

设置 Session 变量，影响当前 Session

```
SET ob_read_consistency = WEAK;
SELECT * FROM t1;
```

设置 Global 变量，影响之后新建的所有 Session

```
SET GLOBAL ob_read_consistency =WEAK;
```

SQL 级设置：在请求的 SQL 中，加上弱一致性读的 Hint，优先级高于 ob_read_consistency

select/*+READ_CONSISTENCY(WEAK)*/ *from t1;

ODP 配置设置：通过 Hint 的方式设置弱读需要修改 SQL，会比较麻烦，此时可以通过修改配置项 obproxy_read_consistency 的值设置弱读.

alter proxyconfig setobproxy_read_consistency = 1;(0:强度,1:弱读)

租户内路由—弱一致性读（读写分离）

某些场景下,我们希望将分析型的查询请求发送到从副本执行。此时就可以使用读写分离功能，将读请求发送到备副本，降低主副本的压力

设置路由策略（ODP 通过配置项 proxy_route_policy 设置读写分离的路由策略）

follower_first：弱读请求优先路由到备副本，如果备副本都不可用，弱读请求路由到主副本

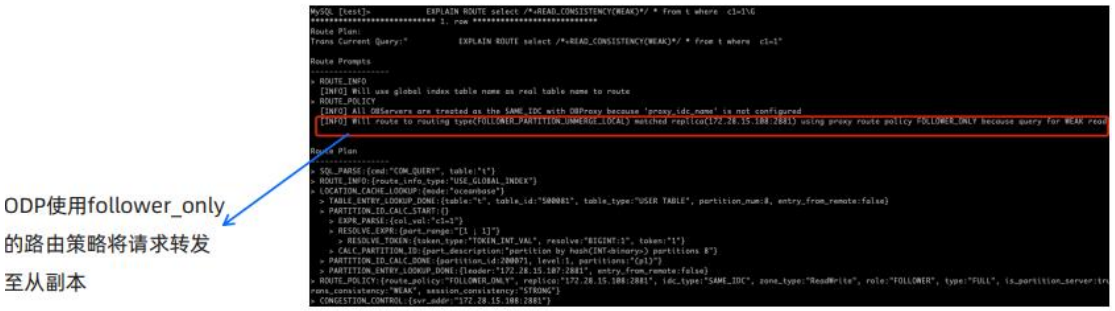
follower_only：弱读请求路由到备副本，如果备副本都不可用，断开和客户端的连接

unmerge_follower_first：优先选择不在集群合并状态的备副本

```
场景一：alter proxyconfig set proxy_route_policy='';
SELECT /*+READ_CONSISTENCY(WEAK) */ * FROM T;
ODP 在 z1、z2、z3 三个副本随机选择副本转发弱读请求
场景二：alter proxyconfig set proxy_route_policy='FOLLOWER_FIRST';
SELECT /*+READ_CONSISTENCY(WEAK) */ * FROM T;
ODP 优先选择 z2、z3 转发弱读请求，如果 z2、z3 都不可用，则转发到 z1
场景三：alter proxyconfig set proxy_route_policy='FOLLOWER_ONLY';
SELECT /*+READ_CONSISTENCY(WEAK) */ * FROM T;
ODP 只选择 z2、z3 转发弱读请求，如果 z2、z3 都不可用，则请求报错
```

租户内路由—弱一致性读（读写分离示例）

```
#sys租户配置proxy_route_policy参数
alter proxyconfig set proxy_route_policy='FOLLOWER_ONLY';
#普通租户创建测试表
CREATE TABLE T(C1 INT,C2 INT,C3 INT) PARTITION BY HASH(C1) PARTITIONS 8;
#弱读查询
EXPLAIN ROUTE select /*+READ_CONSISTENCY(WEAK)*/ * from t where c1=1\G
```

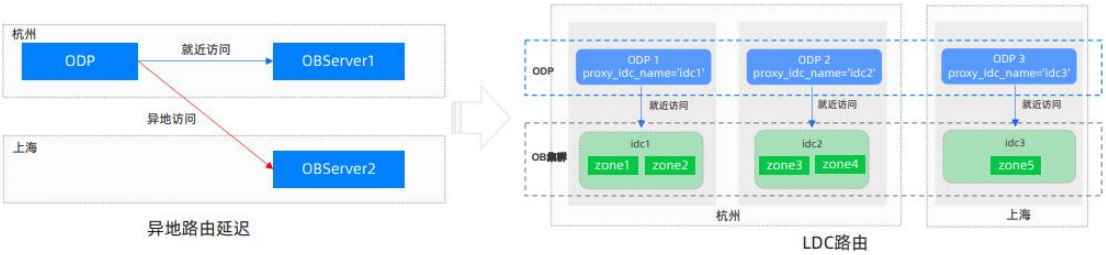


如果 OceanBase 集群是多地部署，ODP如何将收到的弱一致性请求路由至离它最近的OBserver呢?

租户内路由—弱一致性读（LDC 路由）

OceanBase 数据库作为典型的高可用分布式关系型数据库，使用 Paxos 协议进行日志同步，天然支持多地多中心的部署方式以提供高可靠的容灾保证。但当采用多地多中心的部署架构时，弱一致性读会均匀访问数据的所有主从副本，当访问非本地域机房上的副本时，系统响应时间将大大增加。逻辑数据中心（Logical Data Center，LDC）路由正是为了解决这一问题而设计的，通过给 OceanBase 集群的每个 Zone 设置 Region 属性和 IDC 属性，并给 ODP 指定 IDC 名称配置项后，当数据请求发到 ODP 时，ODP 将弱读请求按“同机房>同地区>异地”的优先级顺序进行 OBServer 的选取。

LDC 的路由设置需要三个步骤：配置 OceanBase 集群、配置 ODP、应用配置弱一致性读。



LDC 路由—OceanBase 集群的 LDC 配置

LDC 基本概念：

region 表示 Zone 所在的地域，通常设置为城市名（大小写敏感）

idc 代表该 Zone 所处的机房，通常设置为机房名（小写）

zone 为当前设置的 Zone 的名称

本地： 同城同机房（IDC 相同）

同城： 同城不同机房（IDC 不同，Region 相同）

异地： 不同的地域（Region 不同）

OB 集群 LDC 配置举例：

使用 root 用户登录集群的 sys 租户

登录后，执行以下示例语句配置 LDC

```
ALTER SYSTEM MODIFY zone "zone1" SET region = "SHANGHAI";
ALTER SYSTEM MODIFY zone "zone1" SET idc = "idc1";
```

检查 OBServer 节点中 LDC 设置内容是否生效

```
SELECT * FROM oceanbase.DBA_OB_ZONES;
```

LDC 路由—ODP 的 LDC 配置

通过配置 proxy_idc_name 指定 ODP 在 LDC 逻辑中所处的位置

sys 租户登陆，修改 ODP 的 proxy_idc_name 配置

```
ALTER PROXYCONFIG SET proxy_idc_name= idc1;
```

为 ODP 配置 LDC 后可通过下述语句检查配置是否生效

```
SHOW PROXYINFO IDC;
```

LDC 路由—LDC 弱一致性读示例

LDC 路由仅针对弱一致性读的场景才会生效。应用配置弱一致性示例如下：

在 SQL 中携带 HINT 来指定：SELECT /*+read_consistency(weak)*/ * FROM t1 where c1=1;

路由结果：将弱一致性读请求路由至和当前 ODP 在同一个 IDC 的 OBServer

租户内路由—弱一致性读（总结）

弱一致性读有两种路由策略，其中 LDC 路由的优先级高于读写分离，ODP 在选择副本的时候，先检查 LDC 路由，再走读写分离路由。

路由管理—分布式事务路由

OceanBase 数据库自 V4.1.0 起支持事务在多个节点执行，不受事务开启节点限制。当事务内出现多条读写语句，且其访问的数据不在同一分区时，ODP 可以将事务内的请求发往其数据所在的机器，而非事务开启节点，此功能可以有效的减少 OceanBase 数据库生成远程计划，提升事务性能。

分布式事务路由配置，默认情况下事务路由的功能是开启的，可以使用以下语句查询:

在客户端中使用 root 用户登录集群的 sys 租户

检查 OceanBase 2.0 协议的配置，确保 OceanBase 2.0 协议启用

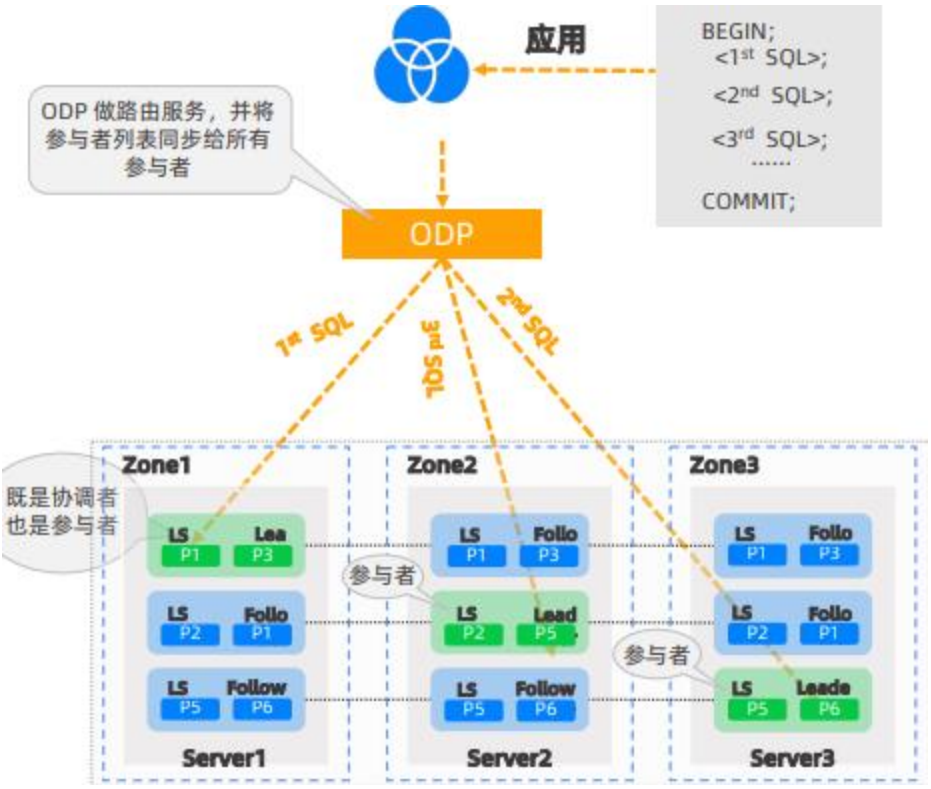
SHOW PROXYCONFIG LIKE enable_ob_protocol_v2;

ALTER PROXYCONFIG SET enable_ob_protocol_v2=True;

检查 ODP 的事务路由进行配置，确保事务路由功能打开

SHOW PROXYCONFIG LIKE enable_transaction_internal_routing;

ALTER PROXYCONFIG SET enable_transaction_internal_routing=True;



路由管理—分布式事务路由示例

登陆业务租户，创建测试表(确保租户的 primary zone 设置为多个 zone）

CREATE TABLE T1(C1 INT,C2 INT) PARTITION BY HASH(C1)PARTITIONS 2;

CREATE TABLE T2(C1 INT,C2 INT) PARTITION BY HASH(C1)PARTITIONS 2;

开启分布式事务

```
BEGIN;
INSERT INTO T1 VALUES(1,1);
```

```
INSERT INTO T1 VALUES(2,2);
INSERT INTO T2 VALUES(11,11);
INSERT INTO T2 VALUES(22,22);
COMMIT;
```

事务执行过程中，查看路由

```
BEGIN;
SELECT * FROM T1 WHERE C1=1;
EXPLAIN ROUTE INSERT INTO T1 VALUES(2,2)\G
```

路由管理—ODP 指定 Zone 路由

在业务并不关心 Leader 位置，需要路由到指定节点的场景下，我们可以通过 ODP 配置项配置指定 zone 路由（proxy_primary_zone_name），ODP 会将请求路由至固定的 OBServer 节点。

ODP 的配置项 proxy_primary_zone_name 优先级高于 OceanBase 集群租户的 Primary Zone。该配置项是强制性的路由。如果是交易支付等强读业务，希望路由到 Leader 的场景，不建议设置该路由策略，避免产生大量远程路由、二次路由等问题。

ODP 配置 primary zone 示例

```
#sys 租户配置 ODP 的 primary zone
ALTER PROXYCONFIG SET PROXY_PRIMARY_ZONE_NAME = 'zone2';
#普通租户创建测试表
CREATE TABLE T(C1 INT,C2 INT,C3 INT) PARTITION BY
HASH(C1) PARTITIONS 8;
```

```
#指定分区键进行查询
EXPLAIN ROUTE SELECT * FROM T WHERE C1=1;
```

路由管理—弱一致性读+ODP 指定 Zone 路由

对于 OLAP 类请求,我们可以将指定的 ODP 配置为弱读，并强制将请求只发送至某个从副本,以减轻主副本的压力。

设置路由策略

1. 设置弱读:

```
alter proxyconfig set obproxy_read_consistency='1';
```

2. 设置读写分离的路由策略

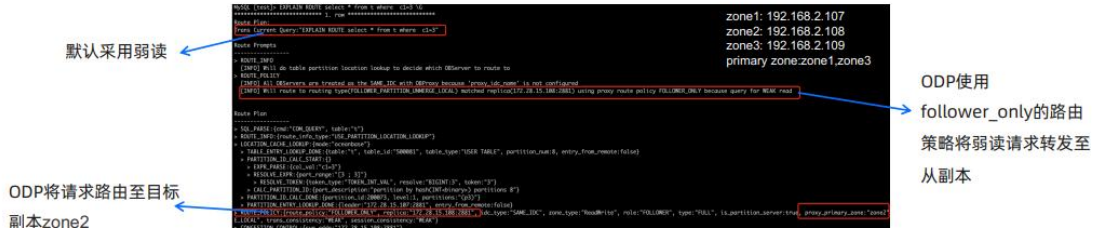
```
alter proxyconfig set
proxy_route_policy='FOLLOWER_ONLY';
```

3. 设置路由目标副本:

```
alter proxyconfig set proxy_primary_zone_name='zone2';
```

示例（接收弱一致性读请求的 ODP ）

```
#普通租户创建测试表
CREATE TABLE T(C1 INT,C2 INT,C3 INT) PARTITION BY
HASH(C1) PARTITIONS 8;
#弱读查询
EXPLAIN ROUTE select * from t where c1=3\G
```



OBProxy 监控诊断

监控与诊断概述

当用户发起请求后，经过负载均衡处理，该请求会被发送至 ODP。ODP 随后会根据路由策略，将请求转发到 OceanBase 集群中的某个 OBServer 节点进行处理。当用户请求出现错误或性能问题时，通常需要对每个相关组件进行问题诊断。ODP 提供了丰富的监控和诊断方法，以便快速识别和解决问题，保障系统的稳定性和高性能。

丰富的诊断日志：ODP 提供了详尽的诊断日志功能，涵盖错误日志、路由诊断和慢查询等日志

性能监控：通过对慢 SQL、连接数和内存等关键指标的监控，ODP 能够及时识别和优化性能瓶颈，保障系统的高效运行

连接问题诊断：ODP 支持深入分析和诊断连接问题，包括认证失败、SQL 执行失败等常见问题

路由问题诊断：ODP 提供的路由诊断工具，能够发现因路由错误导致的请求失败，优化系统的路由策略以提高整体稳定性

诊断日志

当使用 ODP 时遇到登录失败或 SQL 语句执行失败等问题，通常需要查看 ODP 的日志文件进行分析。ODP 的日志文件位于 ODP 安装目录下的 log 目录中，通过检查这些日志，可以帮助快速定位并解决问题。

分类	日志文件	记录内容
监控日志	obproxy_diagnosis.log	记录 ODP 登录、启动、断连接、路由错误等信息的诊断日志
	obproxy_digest.log	记录执行时间超过 query_digest_time_threshold 配置项的 SQL 和和错误响应请求。
	obproxy_error.log	记录执行失败的 SQL 相关信息。
	obproxy_slow.log	记录慢 SQL 请求信息的日志。
	obproxy_stat.log	统计日志，记录一段时间内的 SQL 执行情况
	obproxy_trace.log	记录全链路诊断信息的日志。
系统日志	obproxy.log	记录 ODP 详细信息的日志，内容最全面。
	obproxy.log.wf	WARN 以上级别的 obproxy.log 日志打印到一个单独的日志文件（enable_syslog_wf）

日志级别

当系统出现问题时，日志是我们追踪问题根源、分析故障原因的最直接且有效的工具。为了更高效地管理和利用日志，需要对不同重要性和优先级的信息进行分类，这就是日志级别的意义所在。ODP 将日志划分了七个日志级别，含义如下表所示。

日志级别	含义
ERROR	ODP 出现严重错误。用于记录 ODP 的故障信息，且必须进行故障排除，否则系统不可用。
WARN	出现非预期场景。ODP 能继续提供服务，但行为可能不符合预期，或可能将有严重错误发生，需进行故障排除。
INFO	系统状态变化信息。用于记录系统运行的当前状态，该信息为正常信息。
EDIAG	Error Diagnosis，协助故障排查的诊断信息，非预期的逻辑错误
WDIAG	Warning Diagnosis，协助故障排查的诊断信息，预期内的错误。
TRACE	任务级调试信息。
DEBUG	详细调试信息

对 ODP 进行问题诊断时，可以调低日志级别，以记录更加详细的信息，从而更全面地捕捉系统行为，便于问题的深入分析和定位。

日志级别配置

系统日志：syslog_leve 配置项：设置系统日志 obproxy.log 的默认日志级别。该配置项自 4.2.3 版本起默认值由 INFO 更新为 WDIAG 级别。

监控日志：monitor_log_level 配置项目：设置监控日志的默认日志级别。默认为 INFO 级别。监控日志包含 obproxy_digest.log、obproxy_trace.log、obproxy_diagnosis.log、obproxy_slow.log 和 obproxy_error.log 日志

日志清理策略

在 ODP 运行过程中，会生成大量的诊断日志。默认情况下，每个诊断日志文件的大小为 256MB，当日志文件超过此大小时，系统会自动进行存档。可以通过设置以下参数来控制 ODP 的日志清理策略。

max_log_file_size：用于设置 ODP 中单个日志文件的最大限制。默认值 256MB

log_dir_size_threshold： 用于设置 ODP 日志所在目录的最大可用空间阈值。默认值 10G

log_file_percentage ：用于设置 ODP 日志文件占用可用空间的百分比阈值。默认值 80%

log_cleanup_interval： 用于设置日志文件清理任务的间隔时间，默认值 1 分钟

日志清理策略： 当日志占用空间超过 log_file_percentage * log_dir_size_threshold 值后，每隔 1 分钟，清理任务会对日志进行清理。

性能监控

连接数监控

日常运维过程中，经常需要统计一个 ODP 集群中各个 ODP 的连接数，据此判断客户应用当前所有的连接是否均衡地发给了所有的 ODP，以及粗略判断当前系统的繁忙程度。

方法一：通过 ODP IP 连接到 root@sys 租户后，执行 show processlist 或者 show proxysession 命令。

方法二：登陆 OCP,进入“性能监控”页面，选择“ODP 集群”。

连接数达到上限制

当 ODP 的连接数超过 client_max_connections 参数指定的最大值时，会导致连接失败的问题。

```
[root@obsvr01 ~]# obclient -h172.28.15.107 -P2883 -uroot@test#obcp_cluster –p****** -A -cERROR 1203 (42000): Too many sessions
```

查看 client_max_connections 当前配置

show proxyconfig like '%client_max_connections%'

修改 client_max_connections 的配置

alter proxyconfig set client_max_connections = 10240;

慢 SQl

在数据库运行过程中，经常会出现耗时较长的 SQL 语句。通过设置 slow_query_time_threshold 配置项（默认 500ms），ODP 可以将执行时间超过该阈值的 SQL 语句记录在 obproxy_slow.log 日志中。

```
obclient [(none)]> show proxyconfig like '%slow_query_time_threshold%';
+---------------------------+-------+---------------------------------------+-------------+
| name                      | value | info                                  | need_reboot |
+---------------------------+-------+---------------------------------------+-------------+
| slow_query_time_threshold | 500ms | slow query time threshold, [0s, 30d] | false       |
+---------------------------+-------+---------------------------------------+-------------+
1 row in set (0.001 sec)
```

慢查询测试

1. 使用OBclient登陆业务租户，执行慢SQL语句，例如：

```
obclient [test]> select sleep(3);
```

2. 查看obproxy_slow.log日志中的慢SQL信息：

```
2024-08-18
10:36:11.588971,obpxy,,,,obcp_cluster:test:test,OB_MYSQL,,,COM_QUERY,SELECT,success,,select sleep(3),
3026283us,0us,0us,3017514us,Y0-00007FB4895BB710,YB42AC1C0F6B-00061D7E02AB4740-0-
```

obproxy_slow.log慢日志解析

```
2024-08-18 10:36:11.588971,          → 租户名
obcp_cluster:test:test
.....
success,,
select sleep(3),                     → 执行的SQL语句
3026283us,                           → SQL执行总耗时(us)
0us,                                 → ODP 处理时间
0us,                                 → 链接建立时间
3017514us,                           → OBServer 节点执行时间
Y0-00007FB4895BB710,
YB42AC1C0F6B-00061D7E02AB4740-0-0,
172.28.15.107:23804,,
172.28.15.107:2881
```

obproxy.log慢日志解析

SQL执行总耗时(us)

对于慢 SQL，obproxy.log 中也会有比较详细的记录，关键字是 " Slow Query "

```
[2024-08-18 10:36:11.588915] WDIAG [PROXY.SM] update_cmd_stats (ob_mysql_sm.cpp:11084) [22732][Y0-00007FB4895BB710] [lt=0] [dc=0] Slow Query: {client_ip=
{172.28.15.107:23804}, server_ip={172.28.15.107:2881}, obproxy_client_port={172.28.15.107:5300}, server_trace_id=YB42AC1C0F6B-00061D7E02AB4740-0-0, route_
type=ROUTE_TYPE_NONPARTITION_UNMERGE_LOCAL, user_name=root, tenant_name=test, cluster_name=obcp_cluster, logic_database_name=, logic_tenant_name=, ob_prox
y_protocol=0, cs_id=3420285, proxy_sessid=1240180442629360979, ss_id=23230, server_sessid=3221612702, sm_id=244627, cmd_size_stats={client_request_bytes:
20, server_request_bytes:55, server_response_bytes:0, client_response_bytes:63}, cmd_time_stats={client_transaction_idle_time_us=0, client_request_read_ti
me_us=0, client_request_analyze_time_us=0, cluster_resource_create_time_us=0, pl_lookup_time_us=0, pl_process_time_us=0, bl_lookup_time_us=0, bl_process_t
ime_us=0, congestion_control_time_us=0, congestion_process_time_us=0, do_observer_open_time_us=0, server_connect_time_us=0, server_sync_session_variable_t
ime_us=0, server_send_saved_login_time_us=0, server_send_use_database_time_us=0, server_send_session_variable_time_us=0, server_send_session_user_variable
_time_us=0, server_send_all_session_variable_time_us=0, server_send_start_trans_time_us=0, server_send_xa_start_time_us=0, server_send_init_sql_time_us=0,
 build_server_request_time_us=0, plugin_compress_request_time_us=0, prepare_send_request_to_server_time_us=0, server_request_write_time_us=0, server_proce
ss_request_time_us=3017514, server_response_read_time_us=0, plugin_decompress_response_time_us=0, server_response_analyze_time_us=0, ok_packet_trim_time_u
s=0, client_response_write_time_us=55, request_total_time_us=3026283, sql=select sleep(3), trans_internal_routing_state=not in trans}
```

请求的 SQL语句。

ODP 内存

ODP 中的 proxy_mem_limited 参数用于修改内存限制，该参数为 ODP 进程占用系统内存最大上限。默认是 2G 的大小。

ODP 会对自身进程的物理内存使用进行监控，并将使用率打印在 ODP 的系统日志 obproxy.log 中。

超过 80% 时，每 20s 打印一次日志

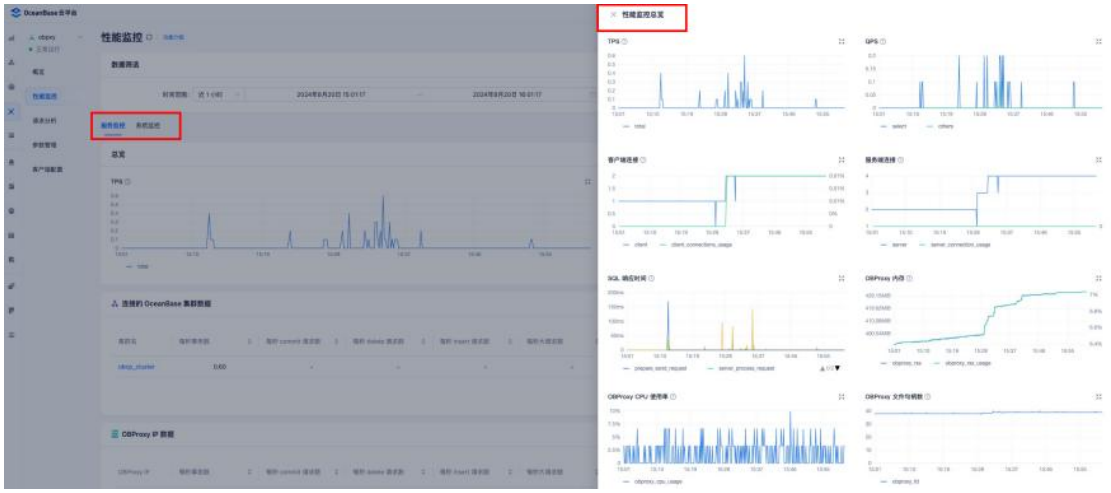
超过 90% 时，2s 打印一次日志

内存用尽时，2s 打印一次日志

当发现 ODP 内存不足时，如果所在服务器内存充裕的话，建议将该参数值调大，且该参数无需重启立刻生效

OCP 服务监控

OCP 支持查看 ODP 集群的性能监控信息，包括服务监控和系统监控。在“总览”区域，点击右上角的“查看全部”按钮，即可展示详细的监控信息。



连接诊断

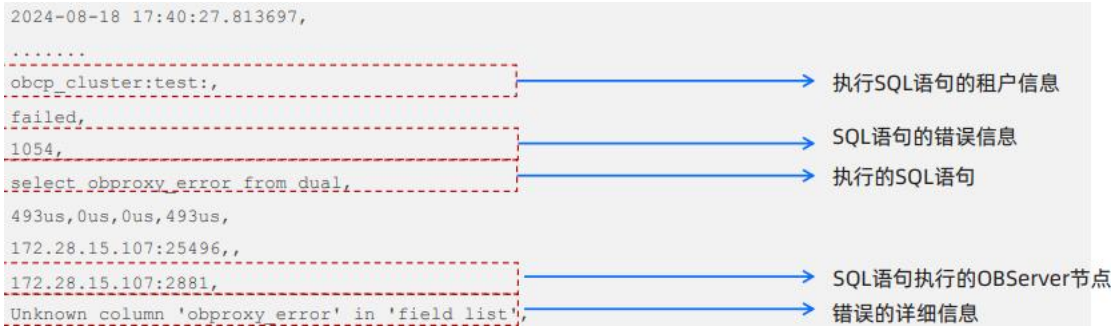
错误日志

obproxy_error.log 日志是 ODP 错误日志，执行错误的请求会打印到该日志中，包括 ODP 自身错误和 OceanBase 数据库返回错误。

SQL 语句执行错误

```
obclient [(none)]> select obproxy_error from dual;
ERROR 1054 (42S22): Unknown column 'obproxy_error' in 'field list'
```

错误日志解析



SQL 执行失败

当 SQL 语句执行失败时，通常会接收到来自 OBServer 端的错误信息。要诊断此类问题，通常需要从 ODP 和 OBServer 两端展开排查，以全面定位问题所在。

3. OBServer端问题诊断

- 根据ODP端日志中的路由信息(示例中为172.28.15.108主机），以及OBServer的Trace id信息，查看OBserver端的日志信息（observer.log）

- 根据冲突事务的ID信息，查看gv$ob_sql_audit获取事务的详细信息。

连接断开问题

当前使用 ODP 时，请求执行的链路主要为：客户端发送请求到 ODP --> ODP 将请求路由到对应的 OBServer 节点 --> OBServer 节点处理请求发送回包给 ODP --> ODP 回包给客户端。整条链路上都可能发生连接断开的场景，例如：请求处理时间较长导致客户端长时间没有收到回包而断开连接、用户登录信息填写错误的集群租户等导致无法登录、ODP 以及 OceanBase 数据库的内部错误导致断开连接等等。

常见的连接断开场景:

登录断连接：常见的场景包括登陆时指定了错误的集群名称、租户名称，客户端连接数达上限等。

超时断连接：当用户请求查询超时，用户事务执行超时，客户端连接长时间没有发生交互导致超时，常会出现连接断开。

客户端主动断连接：常见的场景包括驱动超时主动断开连接、Druid/Nginx 等中间件主

动断连接、网络抖动等

ODP 断连接：例如会话被 kill 掉或者 ODP 内部一些非预期的错误场景，可能会导致连接中断。

OceanBase 数据库断连接 ：例如数据请求时 CheckSum 校验出错，主备库切换时，可能会导致连接中断。

登陆断连接示例

当发生连接断开时，ODP 会将断连接的日志记录到 obproxy_diagnosis.log 日志文件中。该文件会详细记录包括 ODP 登录、启动、连接断开、路由错误等在内的相关信息，便于后续诊断和分析。

1. 使用错误的租户名称连接OB集群。

```
[root@obsvr01 ~]# obclient -h172.28.15.107 -P2883 -uroot@test1#obcp_cluster -p******  -A  -c
ERROR 1045 (42000): Access denied for user 'root@test1#obcp_cluster'@'172.28.15.107' (using password: YES)
```

2. 查看obproxy_diagnosis.log中的日志信息，分析错误原因。

登录断连接对应的 trace_type 为 LOGIN_TRACE

```
[2024-08-19 15:27:56.291279] [9923][Y0-00007F12BA960410] [LOGIN](trace_type="LOGIN_TRACE",
connection_diagnosis={cs_id:3437139, ss_id:0, proxy_session_id:0, server_session_id:0,
client_addr:"172.28.15.107:36974", server_addr:"*Not IP address [0]*:0", proxy_server_addr:"*Not IP address
[0]*:0", cluster_name:"obcp_cluster", tenant_name:"test1", user_name:"root", error_code:-4043,
error_msg:"dummy entry is empty, please check if the tenant exists", request_cmd:"COM_LOGIN",
sql_cmd:"COM_LOGIN", req_total_time(us):26758}{internal_sql:"", login_result:"failed"})
```

租户不存在

错误码-4043，表示租户名错误

3. 解决方法

使用系统租户（root@sys）直连 OBServer 节点后查询DBA_OB_TENANTS视图，查看集群中的所有租户。

超时断连接示例

ODP 超时断连接的错误信息记录在 obproxy_diagnosis.log 日志中。

1. Java应用程序连接断开，出现的错误信息如下

应用程序长时间未与数据库进行通信，超过了数据库interactive_timeout配置的闲置时长

```
com.mysql.cj.jdbc.exceptions.CommunicationsException: The last packet successfully received from the server was 20,027 milliseconds ago. The last packet sent successfully to the server was 20,028 milliseconds ago. is longer than the server configured value of 'interactive_timeout'. You should consider either expiring and/or testing connection validity before use in your application, increasing the server configured values for client timeouts, or using the Connector/J connection property 'autoReconnect=true' to avoid this problem.
        at com.mysql.cj.jdbc.exceptions.SQLError.createCommunicationsException(SQLError.java:174)
        at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMapping.java:64)
        at com.mysql.cj.jdbc.ConnectionImpl.commit(ConnectionImpl.java:814)
        at TimeOutTest.main(TimeOutTest.java:32)
```

2. 查看obproxy_diagnosis.log诊断日志，分析错误原因

超时断连接对应的 trace_type 为 TIMEOUT_TRACE

```
[2024-08-19 22:01:49.810232] [9923][Y0-00007F12BAB60510] [CONNECTION](trace_type="TIMEOUT_TRACE",
connection_diagnosis={cs_id:3433548, ss_id:0, proxy_session_id:-6044939647413190390, server_session_id:0,
client_addr:"172.28.15.107:30128", server_addr:"*Not IP address [0]*:0", proxy_server_addr:"*Not IP address
[0]*:0", cluster_name:"obcp_cluster", tenant_name:"test", user_name:"root", error_code:-10022,
error_msg:"OBProxy inactivity timeout", request_cmd:"COM_SLEEP", sql_cmd:"COM_END",
req_total_time(us):10819750}{timeout(us):10000000, timeout_event:"CLIENT_WAIT_TIMEOUT"})
```

10s内服务端没有收到客户端的交互信息

CLIENT_WAIT_TIMEOUT表示用户请求过程中，客户端连接长时间没有发生交互导致超时

3. 解决方法：

- 检查数据库变量interactive_timeout和wait_timeout的配置。
  - wait_timeout 用于指定服务器关闭交互式连接前等待活动的秒数。
  - interactive_timeout 用于指定服务器关闭非交互连接之前等待活动的秒数。

```
obclient [test]> show variables like '%interactive_timeout%';
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| interactive_timeout | 10    |
+---------------------+-------+
1 row in set (0.001 sec)

obclient [test]> show variables like 'wait_timeout%';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wait_timeout  | 10    |
+---------------+-------+
1 row in set (0.003 sec)
```

- 将变量interactive_timeout和wait_timeout的配置调大，例如恢复为默认值8小时。

```
set global wait_timeout=28800 ;
set global interactive_timeout=28800 ;
```

客户端主动断连接示例

比较常见的断连接问题有驱动超时主动断开连接、Druid/Nginx 等中间件主动断连接、网络抖动等，具体情况需要与客户端配合诊断。

1. Java应用程序连接断开，出现的错误信息如下。

```
java.sql.SQLNonTransientConnectionException: (conn=3421596) Connection timed out
        at com.oceanbase.jdbc.internal.util.exceptions.ExceptionFactory.createException(ExceptionFactory.java:122)
        at com.oceanbase.jdbc.internal.util.exceptions.ExceptionFactory.create(ExceptionFactory.java:202)
        at com.oceanbase.jdbc.OceanBaseStatement.executeExceptionEpilogue(OceanBaseStatement.java:312)
        at com.oceanbase.jdbc.JDBC4PreparedStatement.executeInternal(JDBC4PreparedStatement.java:242)
        at com.oceanbase.jdbc.JDBC4PreparedStatement.execute(JDBC4PreparedStatement.java:159)
        at com.oceanbase.jdbc.JDBC4PreparedStatement.executeQuery(JDBC4PreparedStatement.java:173)
        at test_jtimeout.select(test_jtimeout.java:18)
        at test_jtimeout.main(test_jtimeout.java:34)
Caused by: com.oceanbase.jdbc.internal.util.exceptions.OceanBaseSqlException: Connection timed out
        at com.oceanbase.jdbc.internal.protocol.AbstractQueryProtocol.exceptionWithQuery(AbstractQueryProtocol.java:197)
        at com.oceanbase.jdbc.internal.protocol.AbstractQueryProtocol.exceptionWithQuery(AbstractQueryProtocol.java:180)
        at com.oceanbase.jdbc.internal.protocol.AbstractQueryProtocol.executeQuery(AbstractQueryProtocol.java:372)
        at com.oceanbase.jdbc.JDBC4PreparedStatement.executeInternal(JDBC4PreparedStatement.java:233)
        ... 4 more
```

客户端主动断连接对应的 trace_type 为 CLIENT_VC_TRACE

可判断出客户端在 ODP 处理请求时断开连接。

2. 查看obproxy_diagnosis.log诊断日志，分析错误原因。

```
[2024-08-19 12:07:35.741769] [9923][Y0-00007F12BA9601D0] [CONNECTION](trace_type="CLIENT_VC_TRACE",
connection_diagnosis={cs_id:3421596, ss_id:11107, proxy_session_id:-6044939647413190433,
server_session_id:3221703115, client_addr:"172.28.15.107:39414", server_addr:"172.28.15.107:2881",
proxy_server_addr:"172.28.15.107:20910", cluster_name:"obcp_cluster", tenant_name:"test", user_name:"root",
error_code:-10011, error_msg:"An unexpected connection event received from client while obproxy handling
request", request_cmd:"COM_QUERY", sql_cmd:"COM_QUERY", req_total_time(us):5011201}{vc_event:"VC_EVENT_EOS",
user_sql:"select sleep(10) from dual;"})
```

错误码10011：ODP 处理请求时客户端断连接

客户端正在执行的SQL

请求总的执行时间5s。可以通过 total_time 去匹配客户端的超时参数

连接中断

3. 解决方法：

- 检查JDBC的连接超时参数socketTimeout的配置

  socketTimeout 是 socket 的超时时间，指的是业务的 APP 与后台 OBServer 或 ODP 之间进行 TCP 通信的网络超时时间，如果业务 APP 超过 socketTimeout 的时间还没有从 OBServer 或 ODP 收到反馈，JDBC 会发生网络异常，进而关闭该连接。socketTimeout 单位是毫秒。

```
String selectSql = "select sleep(10) from dual;";
Connection conn = null;
try
{
   Class.forName("com.oceanbase.jdbc.Driver").newInstance();
   conn = DriverManager.getConnection("jdbc:oceanbase://172.28.15.107:2883/test?socketTimeout=5000&user=root@test#obcp_cluster&password=aaAA11__");
   PreparedStatement  ps = conn.prepareStatement(selectSql);
   ResultSet rs = ps.executeQuery();
   while (rs.next()){
     System.out.println(String.format("###   %s   ###", rs.getString("HOST_IP()")));
   }
```

- 将socketTimeout调整为大于SQL语句执行时间的值。

ODP 断连接示例

ODP 引起的连接中断分为两种：未预期错误场景和符合预期的错误场景。对应的 trace_type 为 PROXY_INTERNAL_TRAC

符合预期的错误场景：例如客户端连接被人为 kill 的场景。

未预期错误场景：例如租户分区信息查询失败、ODP 部分内部请求执行失败等场景。

1. OBClient 客户端在执行查询时，连接中断

```
obclient [(none)]> select sleep(100) ;
ERROR 2013 (HY000): Lost connection to MySQL server during query
```

2. 查看obproxy_diagnosis.log诊断日志，分析错误原因。

ODP断链的trace_type对应PROXY_INTERNAL_TRACE

```
[2024-08-20 10:26:31.924565] [9923][Y0-00007F12BA9601D0] [CONNECTION](trace_type="PROXY_INTERNAL_TRACE",
connection_diagnosis={cs_id:3426114, ss_id:22165, proxy_session_id:-6044939647413190373,
server_session_id:3221578476, client_addr:"172.28.15.107:7168", server_addr:"172.28.15.107:2881",
proxy_server_addr:"172.28.15.107:50700", cluster_name:"obcp_cluster", tenant_name:"test", user_name:"root",
error_code:-5065, error_msg:"connection was killed by user session 3467674", request_cmd:"COM_QUERY",
sql_cmd:"COM_QUERY", req_total_time(us):19491762}{user_sql:"select sleep(100)"})
[2024-08-20 10:26:32.028228] [9933][Y0-00007F12BF91F380] [MEMORY]((hold=197132288, buf="
```

当前会话被kill了

路由诊断

路由诊断概述

路由是 ODP 的核心功能之一。ODP 提供了多种丰富的路由诊断工具，帮助用户深入分析和解决路由过程中出现的问题。通过这些诊断功能，可以清晰地了解一条 SQL 语句从进入 ODP 到被成功转发的整个过程中的各个环节与细节。

Explain route 命令行通过该命令可以获取 SQL 语句的路由状态信息。explain route

语句将会在 ODP 内部进行处理，执行正常的 ODP 转发流程，但不会真正转发给 OBServer 节点

route_diagnosis_level 配置项的可选值有 [0-4]，配置的值越大，上述命令展示的状态信息越详细

诊断日志

当 SQL 语句满足以下条件时，ODP 会将实际路由过程输出到 obproxy_diagnosis.log 日

志中

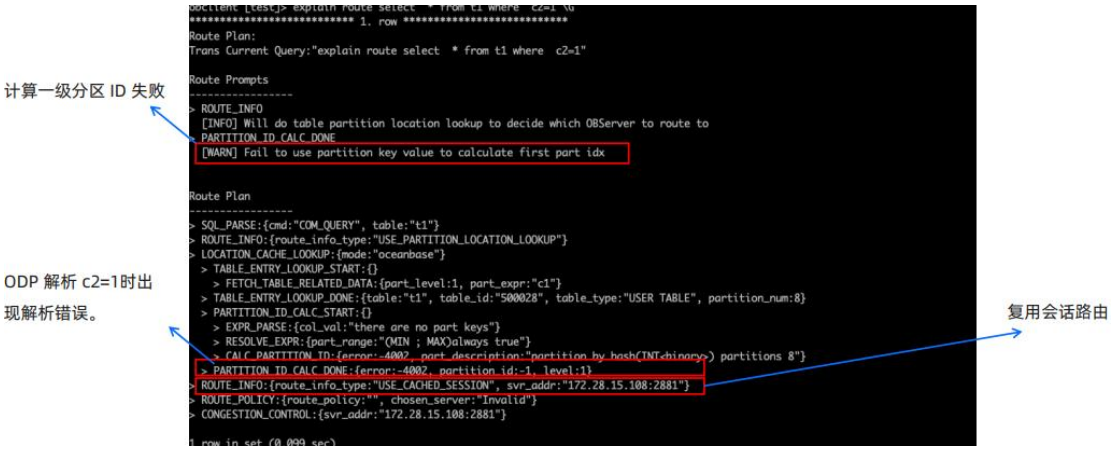
非 EXPLAIN ROUTE executable_sql 语句

设置 route_diagnosis_level = 4 和 monitor_log_level='TRACE' 配置项设置 route_diagnosis_level = 4 和 monitor_log_level='TRACE' 配置项

在 obproxy_diagnosis.log 中找到想诊断的日志行，过滤关键字得到该行，然后替换该日志行内的 '/n' 为 '\n'得到树状的诊断过程。例如

grep "2024-08-17 16:56:46.521180"obproxy_diagnosis.log | sed "s/\/n/\n/g"

命令行 explain route 路由诊断示例



诊断日志示例

设置 route_diagnosis_level = 4 和 monitor_log_level='TRACE'配置项，并执行 SQL 语句。在诊断日志中会生成如下的详细信息。

```
[root@obsvr01 log]# grep "2024-08-20 15:53:35.685965" obproxy_diagnosis.log | sed "s/\/n/\n/g"
[2024-08-20 15:53:35.685965] [10382][Y0-00007F9949F42650] [ROUTE]((*route_diagnosis=
Trans Current Query:"select  * from t1 where  c2=1"

Route Prompts
> ROUTE_INFO
  [INFO] Will do table partition location lookup to decide which OBServer to route to
> PARTITION_ID_CALC_DONE
  [WARN] Fail to use partition key value to calculate first part idx


Route Plan
> SQL_PARSE:{cmd:"COM_QUERY", table:"t1"}
> ROUTE_INFO:{route_info_type:"USE_PARTITION_LOCATION_LOOKUP"}
> LOCATION_CACHE_LOOKUP:{mode:"oceanbase"}
  > TABLE_ENTRY_LOOKUP_DONE:{table:"t1", table_id:"500167", table_type:"USER TABLE", partition_num:8, entry_from_remote:false}
  > PARTITION_ID_CALC_START:{}
    > EXPR_PARSE:{col_val:"there are no part keys"}
    > RESOLVE_EXPR:{part_range:"(MIN ; MAX)always true"}
    > CALC_PARTITION_ID:{error:-4002, part_description:"partition by hash(INT<binary>) partitions 8"}
  > PARTITION_ID_CALC_DONE:{error:-4002, partition_id:-1, level:1}
> ROUTE_INFO:{route_info_type:"USE_CACHED_SESSION", svr_addr:"172.28.15.108:2881"}
> ROUTE_POLICY:{route_policy:"", chosen_server:"Invalid"}
> CONGESTION_CONTROL:{svr_addr:"172.28.15.108:2881"}
> HANDLE_RESPONSE:{is_parititon_hit:"true", send_action:"SERVER_SEND_REQUEST", state:"TRANSACTION_COMPLETE"}
```

数据库开发设计与优化

分布式数据库对象设计

分区表

分布式场景下数据对象设计的思考

热点数据与负载不均衡的问题

问题现象：因不同数据的访问热度不同，特定节点的负载显著高于其他节点，整个集群因为局部节点的性能下降而导致整体性能和吞吐量降低，甚至可能导致服务不可用

解决方案：使用分区表，采用合适的分区键将热点数据分散到多个节点，从而解决高并发访问热点数据时的性能问题

分布式数据访问的 SQL 性能问题

问题现象：SQL 访问的数据分布在多个节点上时，需要跨多个节点进行数据的读写操作，节点间的数据交互导致 SQL 的性能下降

解决方案：可以将具有访问相关性、事务相关性或连接相关性的数据分配在同一个节点上，避免跨节点操作，从而避免分布式 SQL。例如本章后面讲到的表组、复制表等技术

多节点数据有序插入的问题

问题现象：应用需要数据按照插入的先后顺序生成全局一致的顺序号，要求分布式多节点并行插入保证顺序号单独递增

解决方案：使用序列、自增列来自动生成全局唯一、有序的顺序号

分区表概述

分区表的概念：OceanBase 数据库可以把普通的表数据按照一定的规则划分到不同的区块内，同一区块的数据物理上存储在一起。这种划分区块的表叫做分区表，其中的每一个区块称作分区。

分区表的适用场景：

并行处理：通过分区将一个表的数据分散到多个服务节点上，充分利用分布式多节点的处理能力，提升并行处理的性能

热点打散：通过分区将对热点数据的访问分散到多个服务节点上，让整个集群的负载更均衡，避免少数节点出现资源瓶颈

数据管理：通过分区将对表数据的管理从表的维度细分到分区的维度，比如使用 TRUNCATE 分区的操作来替代批量的数据删除

OceanBase 数据库中分区表的特点：

OceanBase 数据库自动对表、分区在多个节点间进行负载均衡和多副本容灾

OceanBase 可动态多机扩展，并在分区间并行查询

分区的分布对业务透明，可以取代 MySQL 的“分库分表”方案

单表的最大分区数

MySQL 模式由租户配置项 max_partition_num 控制，默认值 8192

Oracle 模式是 65536

单机分区数支持上限：根据租户内存大小来预估，每 1GB 内存约支持 20000 分区

分区策略

分区键：数据表中每一行中用于计算这一行属于哪一个分区的列的集合叫做分区键

分区策略:

OceanBase 支持的分区策略包括范围（Range）分区、列表（List）分区和哈希（Hash）分区

OceanBase 支持组合分区，即先使用一种分区策略，然后在子分区再使用另外一种分区策略

分区策略	MySQL 模式	Oracle 模式
范围分区	RANGE、RANGE COLUMNS	RANGE
列表分区	LIST、LIST COLUMNS	LIST
哈希分区	HASH、KEY	HASH

不同分区策略下的分区键选择规则：

分区键必须是主键或唯一键的子集。只要表有主键，分区键就必须是主键的子集

MySQL 模式下，RANGE、LIST、HASH 的分区键必须是单个列或表达式，且必须是整数类型或 YEAR 类型；RANGE COLUMNS、LIST COLUMNS、KEY 的分区键可以是多个列，但不支持表达式，并支持除 BLOB、TEXT 外的大部分数据类型

Oracle 模式下，RANGE、LIST、HASH 的分区键规则与 MySQL 模式下 RANGE COLUMNS、LIST COLUMNS、KEY 相似

范围分区

范围分区是业界使用最广泛的分区类型，可以按用户指定的表达式范围将每一条记录划分到不同分区

范围分区支持将最高分区的上限定义 MAXVALUE，表示一个虚拟无限值，其排序高于分区键的任何其他可能值，包括空值

适用场景： 按时间字段分区。例如：可以将历史流水表按日/周/月分区，实现批量数据的清理、备份、导入导出

Oracle 模式创建范围分区的示例

```
CREATE TABLE parttb_range(id INT, gmt_create TIMESTAMP, info VARCHAR(20), PRIMARY KEY (gmt_create))
PARTITION BY RANGE(gmt_create)
( PARTITION p0 VALUES LESS THAN ('2025-01-01 00:00:00'),
PARTITION p1 VALUES LESS THAN (MAXVALUE) );
```

MySQL 模式创建范围分区的示例：

```
CREATE TABLE parttb_range_1(id INT, gmt_create TIMESTAMP, info VARCHAR(20), PRIMARY KEY (gmt_create))
PARTITION BY RANGE(UNIX_TIMESTAMP(gmt_create))
( PARTITION p0 VALUES LESS THAN (UNIX_TIMESTAMP('2025-01-01 00:00:00')),
PARTITION p1 VALUES LESS THAN (MAXVALUE) );
CREATE TABLE parttb_range_2(id INT, gmt_create TIMESTAMP, info VARCHAR(20), PRIMARY KEY (gmt_create))
PARTITION BY RANGE COLUMNS(gmt_create)
( PARTITION p0 VALUES LESS THAN ('2025-01-01 00:00:00'),
PARTITION p1 VALUES LESS THAN (MAXVALUE) );
```

注释：

1. gmt_create 字段类型是 TIMESTAMP，在 MySQL 模式下如果使用 RANGE 分区，需要先使用 UNIX_TIMESTAMP 函数将其转化为整型。
2. MAXVLUE 分区包含从倒数第二个分区上限之后的所有数据

列表分区

列表分区为每一个分区指定特定的值的列表，每一个分区仅包含分区键值属于对应列表的记录

列表分区支持将最后一个分区指定为 DEFAULT，这个值没有具体的数值，代表所有不在其他分区的列表中的值

适用场景:

List 分区的优点是可以方便的对无序或无关的数据集进行分区

可以将数据按照某种业务逻辑进行集中，如按照子公司进行分区

Oracle 模式创建列表分区的示例：

```
CREATE TABLE parttb_list (id INT, flag VARCHAR(2))
PARTITION BY LIST(flag)
( PARTITION p0 VALUES ('00','01'),
PARTITION p1 VALUES ('02','03'),
PARTITION p2 VALUES (DEFAULT) );
```

MySQL 模式创建列表分区的示例：

```
CREATE TABLE parttb_list (id INT, flag VARCHAR(2))
PARTITION BY LIST COLUMNS(flag)
( PARTITION p0 VALUES IN ('00','01'),
PARTITION p1 VALUES IN ('02','03'),
PARTITION p2 VALUES IN (DEFAULT) );
```

注释：

1. flag 字段类型是 VARCHAR，在 MySQL 模式下需使用 LIST COLUMNS 分区。
2. DEFAULT 分区包含其余分区之外的所有记录，比如 '04‘、’ab' 等

哈希分区

哈希分区通过 Hash 函数计算分区键的 Hash 值，并将不同 Hash 值的记录均匀分散到不同的分区中。通常，当分区数量是 2 的幂次方时，分区的数据分布比较均匀.

适用场景：哈希分区常用于解决满足以下特征的热点数据访问的问题

热点表没有合适的字段提供范围分区，或者按范围分区会导致不同范围的分区数据分布严重倾斜

热点表没有定期按时间范围清理陈旧数据的需求，因为 Hash 分区不能提供定期按数据范围对陈旧分区做清理的功能

Oracle 模式创建哈希分区的示例：

```
CREATE TABLE parttb_hash (c1 INT, c2 VARCHAR(10))
PARTITION BY HASH(c2) PARTITIONS 8;
```

MySQL 模式创建哈希分区的示例：

```
CREATE TABLE parttb_hash (c1 INT, c2 VARCHAR(10))
PARTITION BY HASH(c1+1) PARTITIONS 8;
CREATE TABLE parttb_key (c1 INT, c2 VARCHAR(10))
PARTITION BY KEY(c2) PARTITIONS 8;
```

注释：1. 在 MySQL 模式下 HASH 分区的分区键只支持整数和 YEAR 类型，且可以使用表达式

组合分区

组合分区按照两个维度来对数据分区：先按一种分区策略拆分一级分区，然后再按另一种分区策略将每一个一级分区拆分成二级分区

范围分区、列表分区和哈希分区都可以作为组合分区表的一级分区与二级分区策略

适用场景：适用于依靠单一分区策略无法满足的场景

比如，用户账单表既需要考时间范围分区来管理历史数据，又需要对当前时间范围的分区内的账号按照哈希分区来消热点

组合分区的示例：

```
CREATE TABLE history_t (user_id INT,
gmt_create DATETIME, info VARCHAR(20),
PRIMARY KEY(user_id, gmt_create))
PARTITION BY RANGE COLUMNS (gmt_create)
SUBPARTITION BY HASH(user_id) SUBPARTITIONS 3
(PARTITION p0 VALUES LESS THAN ('2024-11-11'),
PARTITION p1 VALUES LESS THAN ('2025-11-11'),
PARTITION p2 VALUES LESS THAN ('2026-11-11'),
PARTITION p3 VALUES LESS THAN ('2027-11-11') );
```

☐ 思考：Range + Hash 的组合分区 和 Hash + Range 的组合分区分别适用于什么场景？

分区管理-添加分区

添加分区是指为已经分区的表添加后续分区

Oracle 模式中：

对于一级分区表，当前仅支持对 Range 分区和 List 分区执行添加分区操作，暂不支持对 Hash 分区执行添加分区操作

对于二级分区表，当前仅支持对 Range/List 类型（组合）的分区执行添加分区操作

仅支持向非模板化二级分区表中添加二级分区

MySQL 模式中：

暂不支持向表中添加二级分区

对于一级分区表，支持 Range / Range Columns / List / List Columns 执行添加分区操作，暂不支持对 Hash 分区和 Key 分区执行添加分区操作

```
ALTER TABLE table_name ADD PARTITION (partition_option);
```

分区管理-删除分区

根据业务需要，删除分区表中的分区。删除分区时，会同时删除该分区的定义和数据

根据业务需要，删除分区表中的分区。删除分区时，会同时删除该分区的定义和数据

Oracle 模式中：

对于一级分区表，当前仅支持对 Range 分区和 List 分区执行删除分区操作，暂不支持对 Hash 分区执行删除分区操作。对于二级分区表，当前仅支持对 Range/List 类型（组合）的分区执行删除分区操作

对于有全局索引的一级分区表或二级分区表，删除分区时，需要添加 UPDATE GLOBAL INDEXES 来更新全局索引信息。如果未添加，删除分区后该分区表上的全局索引会处于不可用状态

MySQL 模式中：

对于一级分区表，支持 Range / Range Columns / List / List Columns 执行删除分区操作，暂不支持对 Hash 分区和 Key 分区执行删除分区操作

```
MySQL 模式：
ALTER TABLE table_name DROP PARTITION partition_name[, partition_name ...];
Oracle 模式：
ALTER TABLE table_name DROP PARTITION partition_name_list [UPDATE GLOBAL INDEXES];
```

分区管理-Truncate 分区

可以根据业务需要，清空分区表中分区的数据，保留分区结构

Truncate 分区时，尽量保证待 Truncate 的分区上不存在活动的事务或查询，否则可能会导致 SQL 语句报错，或者出现一些异常情况。可在 sys 租户下通过视图 oceanbase.GV$OB_TRANSACTION_PARTICIPANTS 查询当前还未结束的事务上下文状态

Oracle 模式中：

对于一级分区表，仅支持对 Range 分区和 List 分区执行 Truncate 分区操作，暂不支持对 Hash 分区执行 Truncate 分区操作。对于二级分区表，仅支持对 Range/List 类型（组合）

的分区执行 Truncate 分区操作

对于有全局索引的一级分区表或二级分区表，Truncate 分区时，需要添加 UPDATE GLOBAL INDEXES 来更新全局索引信息。如果未添加，Truncate 分区后该分区表上的全局索引会处于不可用状态

MySQL 模式中：

对于一级分区表，支持 Range / Range Columns / List / List Columns 执行 Truncate 分区操作，暂不支持对 Hash 分区和 Key 分区执行 Truncate 分区操作

```
MySQL 模式：
ALTER TABLE table_name TRUNCATE PARTITION partition_name[, partition_name ...];
Oracle 模式：
ALTER TABLE table_name TRUNCATE PARTITION partition_name_list [UPDATE GLOBAL INDEXES];
```

分区表索引

分区表索引概述

OceanBase 数据库主要涉及如下索引类型：

局部分区索引：在创建索引时指定关键字 LOCAL 的索引为局部索引，局部索引无须指定分区规则，它的分区属性和主表的属性一致，也会跟随主表的分区操作而发生变更

全局索引：在创建索引时指定关键字 GLOBAL 的索引为全局索引，全局索引可以按照与主表不一样的分区规则进行分区。但如果主表没有必要进行分区，通常来说全局索引也就没有必要做分区

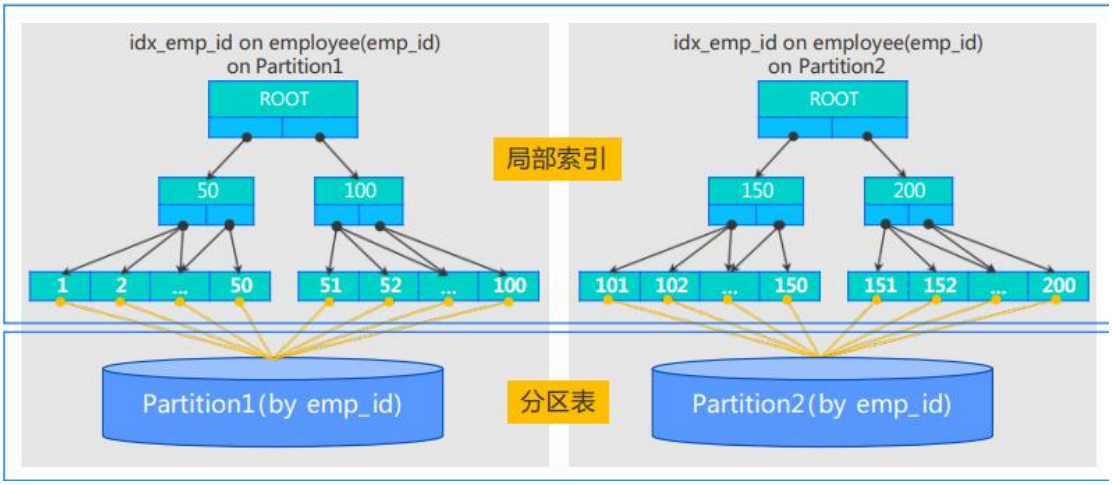
唯一索引：索引键值具有唯一性，用关键字 UNIQUE 表示

前缀索引：当分区索引表的分区键是索引列的左前缀时，该索引称为前缀索引。这不同于 MySQL 长字符串的前缀索引。例如，索引表 idx 建立在 c1 和 c2 列上，如果该索引表的分区键为 c1，那么该索引称为前缀索引；如果该索引表的分区键为 c2 或者其他列，则称为非前缀索引

非前缀索引：相对于分区前缀索引而言，如果分区索引不是前缀索引，那么称为非前缀索引

局部索引

分区表的局部索引和非分区表的索引类似，索引的数据结构还是和主表的数据结构保持一对一的关系，但由于主表已经 做了分区，主表的“每一个分区”都会有自己单独的索引数据结构。局部索引的结构如下图所示：



局部索引根据分区键划分为局部前缀索引和局部非前缀索引

MySQL 模式默认创建的是局部索引。Oracle 模式默认创建的是全局索引，且索引表只有一个分区。

局部前缀索引：分区键是索引的左前缀，并且索引包含二级分区键。局部前缀索引可以是唯一索引或者非唯一索引。唯一索引必须包括表分区函数中的所有列。

指定索引键的查询利用局部前缀索引可以唯一定位到一个索引分区，非常适合于结果集比较小，但是需要进行分区裁剪的场合

例如，表 A 上有个局部索引 idx(c1,c2,c3)，主表根据 c1 进行分区，由于索引表和主表的分区方式一样， idx 也以 c1 为分区键，当处理指定索引键的查询时，可以通过分区键 c1 的值定位到唯一一个索引分区，大大减少了索引分区的访问

局部非前缀索引：如果索引表不是局部前缀索引，那就是局部非前缀索引。可能的情况是，索引表的分区键不是索引的左前缀，或者索引没有包含二级分区键

如果分区键不是索引的子集，那么局部非前缀索引不能是唯一索引

指定索引键的查询利用局部非前缀索引，需要访问所有索引分区，因此比较适合访问数据量比较大，注重并发的场景

例如，表 A 上有个局部索引 idx(c1,c2,c3)，主表根据 c4 列进行分区，当用户查询指定索引键时，不能通过分区键定位到索引分区，因此需要访问所有的索引分区才能获得结果

局部前缀索引示例：

```
obclient> CREATE TABLE tbl2_f_rl(col1 INT,col2 INT)
PARTITION BY RANGE(col1) SUBPARTITION BY LIST(col2)
(PARTITION p0 VALUES LESS THAN(100)
(SUBPARTITION sp0 VALUES IN(1,3),
SUBPARTITION sp1 VALUES IN(4,6)),
PARTITION p1 VALUES LESS THAN(200)
(SUBPARTITION sp3 VALUES IN(1,3),
SUBPARTITION sp4 VALUES IN(4,6)));
obclient> CREATE UNIQUE INDEX tbl2_f_rl_idx1 ON tbl2_f_rl(col1,col2) LOCAL;
```

局部非前缀索引示例：

```
obclient> CREATE TABLE tbl2_f_rl(col1 INT,col2 INT,col3 INT)
PARTITION BY RANGE(col1) SUBPARTITION BY LIST(col2)
(PARTITION p0 VALUES LESS THAN(100)
(SUBPARTITION sp0 VALUES IN(1,3),
```

```
SUBPARTITION sp1 VALUES IN(4,6)),
PARTITION p1 VALUES LESS THAN(200)
(SUBPARTITION sp3 VALUES IN(1,3),
SUBPARTITION sp4 VALUES IN(4,6)));
obclient> CREATE UNIQUE INDEX tbl2_f_rl_idx1 ON tbl2_f_rl(col1,col3) LOCAL;
```

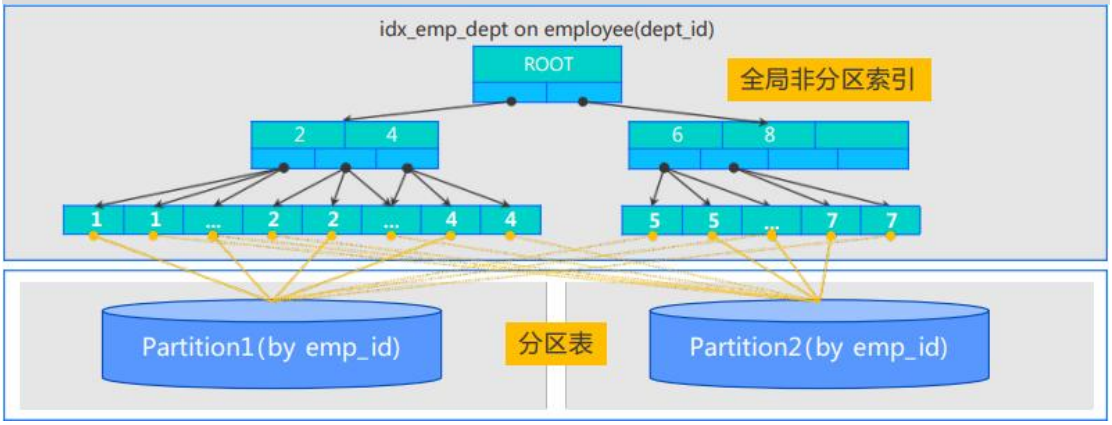
全局索引

分区表的全局索引不再和主表的分区保持一对一的关系，而是将所有主表分区的数据合成一个整体来建立全局索引。全局索引可以定义自己独立的数据分布模式，既可以选择非分区模式也可以选择分区模式：

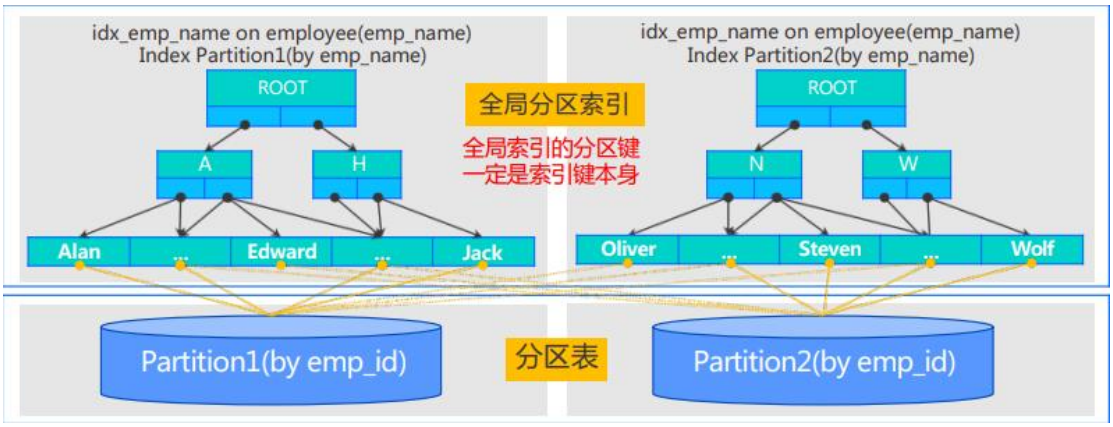
全局非分区索引（Global Non-Partitioned Index）

全局索引创建示例

```
CREATE TABLE tbl1_h(col1 INT PRIMARY KEY,col2 INT) PARTITION BY HASH(col1) PARTITIONS 5;
CREATE INDEX tbl1_h_idx1 ON tbl1_h(col2) GLOBAL;
```



全局分区索引（Global Partitioned Index）




```
CREATE TABLE tbl1_h(col1 INT PRIMARY KEY,col2 INT) PARTITION BY HASH(col1) PARTITIONS 5;
CREATE INDEX tbl1_h_idx1 ON tbl1_h(col2) GLOBAL
PARTITION BY RANGE(col2)
(PARTITION p0 VALUES LESS THAN(100),
PARTITION p1 VALUES LESS THAN(200),
PARTITION p2 VALUES LESS THAN(300)
```

```
);
```

索引创建策略

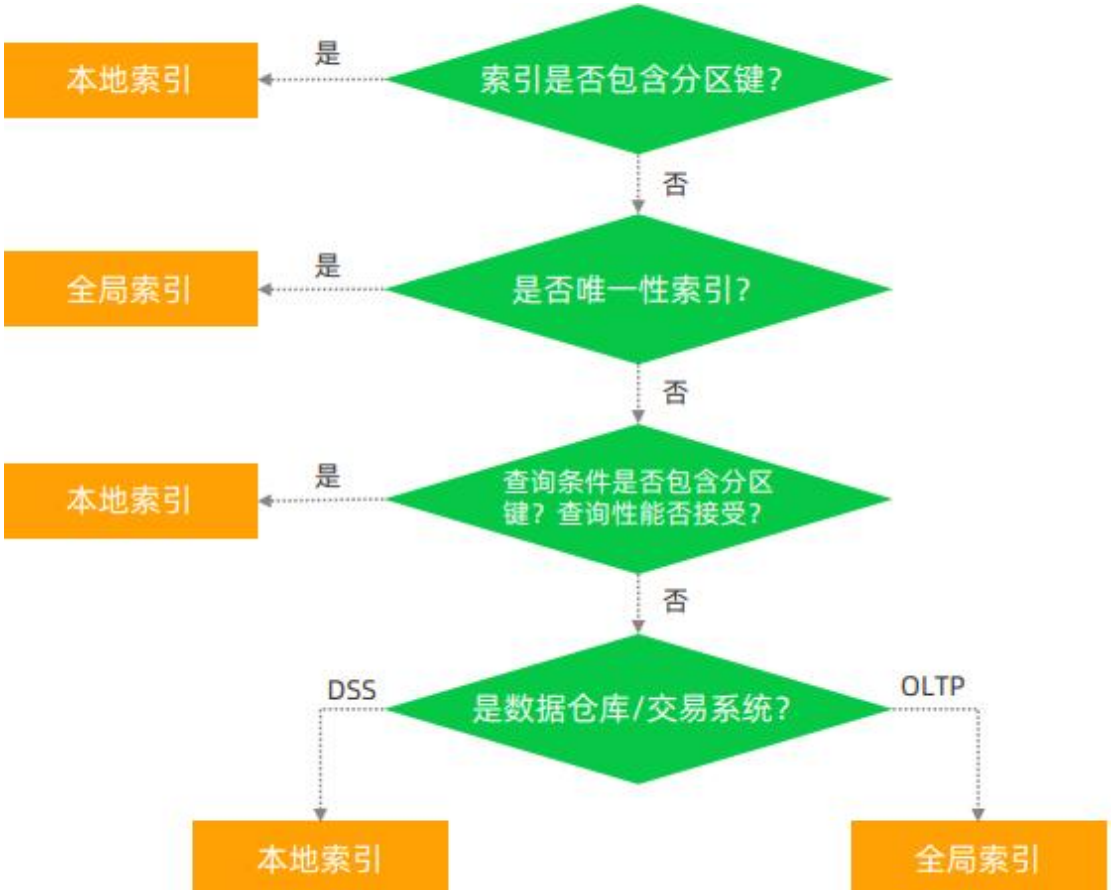
创建索引需要考虑用户访问数据方式、性能、索引可维护性等方面

如果索引键包含主表所有分区键，推荐使用局部索引

如果索引键不包含主表的分区键，且是唯一索引，需要采用全局索引

查询条件包含分区键，且查询性能能够接受，建议优先采用局部索引

如果主表存在大量 DML 的情况下，使用全局索引，会导致更多跨分区的分布式事务，这种场景不建议使用全局索引



表组

表组概述

表组的概念：表组（Table Group）是一个逻辑概念，表示一组表的集合。OceanBase 依据表组的 SHARDING 属性控制组内表和分区的物理分布，使表组内的表或不同表的相同分区聚合在相同的服务节点上

表组的适用场景：通过表组将业务相关联的表按照一定的规则进行物理上的聚合，避免跨节点的数据访问，避免分布式事务，提升查询和交易的性能

SHARDING	表组内表的分区要求	表组内的表、分区的分布规则
NONE	无要求：不同的表可以使用不同的分区规则，允许有的表分区，有的表不分区	表组内的所有表、分区均聚集在同一个节点上
PARTITON	所有表的一级分区的分区规则相同	第一张表的一级分区平均分布到所有的节点上 其他表的相同分区号的一级分区与第一张表聚集 若表有二级分区，则一级分区下的所有二级分区聚集在一起
ADAPTIVE	所有表的分区规则完全相同，要么都不分区，要么都是相同的一级分区，或者都是相同的组合分区	如果仅有一级分区，则与SHARDING=PARTITION 相同 如果都是组合分区，则按二级分区打散，相同的二级分区聚集在相同的节点上

创建与管理表组

创建表组的语法 :CREATE TABLEGROUP tablegroup_name SHARDING = 'NONE'|'PARTITION'|'ADAPTIVE' ;

将表加入表组的方法有三种:

创建表时，直接指定其所属的表组：CREATE TABLE ... TABLEGROUP = tablegroup_name;

创建表后，修改表来指定表组：ALTER TABLE ... TABLEGROUP = tablegroup_name;

创建表后，修改表组来添加表： ALTER TABLEGROUP tablegroup_name ADD table_name [, table_name...];

```
CREATE TABLEGROUP tbg1 SHARDING = 'NONE';
CREATE TABLE tb1 (col INT) TABLEGROUP ='TBG1';
ALTER TABLE tb2 TABLEGROUP ='TBG1';
ALTER TABLEGROUP tbg1 ADD tb3;
```

将表从表组中移除：

```
ALTER TABLE table_name DROP TABLEGROUP;
ALTER TABLE table_name [SET] TABLEGROUP='';
```

删除表组的语法:

```
DROP TABLEGROUP tablegroup_name ;
```

注释：仅当表组为空时，即没有任何表指定 TABLEGROUP 属性为该表组时，才可以删除该表组。

表组查询

系统租户和 MySQL 用户租户可以分别通过视图 CDB_OB_TABLEGROUPS 和 视图 DBA_OB_TABLEGROUPS 来查看表组的 SHARDING 属性及表组内表的分区信息。Oracle 用户租户可以通过视图 DBA_OB_TABLEGROUPS 来查看表组的 SHARDING 属性及表组内表的分区信息

（1）系统租户:SELECT * FROM oceanbase.CDB_OB_TABLEGROUPS WHERE tablegroup_name = 'tblgroup1';

（2）MySQL 用户租户:SELECT * FROM oceanbase.DBA_OB_TABLEGROUPS WHERE tablegroup_name = 'tblgroup1';

（3）Oracle 用户租户:SELECT * FROM SYS.DBA_OB_TABLEGROUPS WHERE tablegroup_name = 'TBLGROUP1';

还可以通过 SHOW TABLEGROUPS 语句查看表组内表的所属数据库及表组的 SHARDING 属性

表组属性修改

创建表组后，根据业务调整，可以使用 ALTER TABLEGROUP 语句修改表组的 SHARDING 属性。

ALTER TABLEGROUP tablegroup_name SHARDING = 'NONE'|'PARTITION'|'ADAPTIVE' ;

修改规则:

将表组的 SHARDING 属性修改为 NONE 时，可直接修改

将表组的 SHARDING 属性修改为 PARTITION 时：

如果表组内的表都是非分区表，可以直接修改

如果表组内同时有非分区表和分区表，无法修改

如果表组内的表都是一级分区表或者组合分区表，要求所有表的一级分区对应的分区规则完全相同，否则无法修改

将表组的 SHARDING 属性修改为 ADAPTIVE 时:

如果表组内的所有表都是非分区表，可以直接修改

如果表组内同时有非分区表和分区表，无法修改

如果表组内同时有一级分区表和组合分区表，无法修改

如果表组内全部是一级分区表或者全部是组合分区表，要求所有表的分区规则完全均相同，否则无法修改

注意：OceanBase 数据库升级过程中，禁止使用 ALTER TABLEGROUP 语句修改表组的 SHARDING 属性。

复制表和 BUFFER 表

复制表概述

复制表的概念：复制表是 OceanBase 数据库中一种特殊类型的表。复制表会在租户的每一个服务节点上创建一份数据副本，OceanBase 可以在任意一个“健康”的副本上读取到数据的最新修改，而普通表只有主副本才能保证读到最新的数据

复制表的适用场景：

复制表适用于读多写少且数据量较小的表，比如参数表

复制表用于解决跨机读取数据的性能问题，通过在每个服务节点上复制一份数据，让读请求都可以在本地完成

OceanBase V4.2 版本中复制表的特点：

复制表会在租户的每一个 UNIT 内各创建一个副本，以满足大量并发的读请求

复制表只有一个 Leader 副本，可以接受写请求；复制表的所有“健康”副本都能接受强一致性读请求

OceanBase 基于分区的可读版本号校验以及基于日志流的 Lease 授予机制，用于保证强一致性读的正确性

如何使用复制表

创建复制表（仅用户租户可以创建复制表， sys 租户无法创建复制表。）：

CREATE TABLE table_name ... DUPLICATE_SCOPE='none | cluster';

none：表示该表是一个普通表，none 是默认值

cluster：表示该表是一个复制表

修改表的复制表属性：V4.2 版本暂时不支持修改表的复制表属性

系统视图 DBA_OB_TABLE_LOCATIONS 的 DUPLICATE_SCOPE 字段标识了表的复制表属性：

database_name	table_name	partition_name	duplicate_scope
test	table_dup	null	cluster
test	table_dup2	null	none

复制表创建成功后，可以与普通表一样进行插入和读写操作。不同的是，对于读请求，如果使用 Proxy 的方式连接数据库，则读请求可能会路由到任意一个 OBServer 节点执行；如果是通过直连方式连接数据库，则只要本地副本可读，系统就会在直连的 OBServer 节点上执行读请求

Buffer 表概述

Buffer 表的概念：Buffer 表又被称为 Queuing 表，意为业务上“像使用 buffer 一样使用一张表”。通常情况下，Buffer 表实际的数据量不大，但在业务执行过程中会有大量的更新操作或者插入再增删操作

Buffer 表效应：Buffer 表上的查询性能随着更新或者增删次数的增长而越来越差。Buffer 表效应是基于 LSM-Tree 机制实现的存储引擎的通用问题

在 Memtable 中，记录的多版本记录链表随着更新次数的增加变得越来越长

默认设置下，删除行的操作只是在内存里打个删除标记，数据到合并时才会真正被删除。当增量数据中积累了大量标记删除的数据时，从上层应用视角实际存在的行很少，但范围查询时可能需要处理较多的标记删除的数据，从而导致 SQL 耗时不够理想

同时 Buffer 表场景下也容易导致优化器生成非最优执行计划

OceanBase 的 Buffer 表解决方案：

自适应合并：OceanBase 4.2+ 支持分区级的自适应合并，数据库自动检测 Buffer 表行为，并主动发起分区级的合并。通过合并来消除 Buffer 表中的删除记录空洞和多版本数据，缩小 Buffer 表的实际大小，提升查询性能

表级的合并策略：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 版本开始支持表级的合并策略。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 表的性能问题

指定 Buffer 表的合并策略（V4.2.3+）

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

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

示例：指定 Buffer 表的合并策略

```
CREATE TABLE tbl1 (c1 int, c2 double) TABLE_MODE = 'queuing';
ALTER TABLE tbl1 SET TABLE_MODE = 'moderate';
```

注意：

1. 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

2. 在 V4.2 版本中，表的 table_mode 属性只能通过 SHOW CREATE TABLE 的方式查看，系统表暂不提供 table_mode 的查询

分布式只读外表

外表介绍

数据库中的表数据存放在数据库的存储空间中，而外表的数据存储在外部存储服务中
创建外表时，需要定义数据文件路径和数据文件的格式
创建成功后，可以通过外表从外部存储服务中读取文件中的数据
外表与普通表的差异如下：
外表的数据存储在外部文件中，普通表的数据存储在数据库中
外表是只读的，可以在查询语句使用，但不能执行 DML 操作
外表不支持添加约束和创建索引
外表的访问速度会慢于普通表

创建外表的语法

```
CREATE EXTERNAL TABLE table_name
( col_name col_type [AS (metadata$filecol{N})]
[ , col_name col_type [AS (metadata$filecol{N})] ]
[ , ... ] )
LOCATION = '<string>'
FORMAT = (
TYPE = 'CSV'
LINE_DELIMITER = '<string>' | <expr>
FIELD_DELIMITER = '<string>' | <expr>
ESCAPE = '<character>' | <expr>
FIELD_OPTIONALLY_ENCLOSED_BY = '<character>' | <expr>
ENCODING = 'charset'
NULL_IF = ('<string>' | <expr>, '<string>' | <expr> ...)
SKIP_HEADER = <int>
SKIP_BLANK_LINES = { TRUE | FALSE }
TRIM_SPACE = { TRUE | FALSE }
EMPTY_FIELD_AS_NULL = { TRUE | FALSE }
)
[ PATTERN = '<regex_pattern>' ]
```

AS (metadata$filecol{N})：当外部文件中的列顺序与外表中定义的列顺序不一致时，可以通过形如 metadata$filecol{N} 的伪列来指定外表的列对应外部文件中的第 N 列

LOCATION = '<string>'：用于指定外部文件存放的路径
路径中不能指定单独的文件，必须是文件所在的上层目录
在创建外表时，外表会自动收集该目录中的所有文件
既可以是 OceanBase 服务器的本地文件路径，也可以对象存储上的远程文件路径
FORMAT：用于定义外部文件的格式
PATTERN：用于指定正则模式串，过滤 LOCATION 目录下的文件

对每个 LOCATION 目录下的文件，如果能够匹配该模式串，则外表就可以访问该文件，否则外表会跳过该文件。如果不指定该参数，则默认可以访问 LOCATION 目录下的所有文件

创建本地文件外表的示例

假设本地机器的 /home/admin/oceanbase/ 下存放了一个 data.csv 文件，文件中的内容如下。

```
1,"lin",98
2,"hei",90
3,"ali",95
```

在 OBserver 节点上，租户管理员通过本地 Unix Socket 连接集群的 MySQL 租户;

```
obclient -S /home/admin/oceanbase/run/sql.sock -uroot@sys -p********
```

配置数据库可以访问的路径 /home/admin/，配置完成后退出登录

```
SET GLOBAL secure_file_priv = "/home/admin/";
```

重新连接数据库后，创建外表 ext_t3

```
CREATE EXTERNAL TABLE ext_t3
(id int, name char(10),score int)
LOCATION = '/home/admin/oceanbase/'
FORMAT = (
TYPE = 'CSV'
FIELD_DELIMITER = ','
FIELD_OPTIONALLY_ENCLOSED_BY ='"'
)
PATTERN = 'data.csv';
```

查询外表;

注意：使用本地文件创建外表前，需要通过系统变量 secure_file_priv 配置 OceanBase 数据库有权限访问的文件路径。要求 secure_file_priv 必须是 local_file_path 的上层目录，而 local_file_path 是本地外表文件的上级目录，即 secure_file_priv 是本地外表文件的祖父目录。

创建远程文件外表的示例

假设外表文件位于阿里云 OSS 对象存储中：所在桶为 obcp，所在目录是 ob421，文件名 data.csv，文件内容如下。

```
1,"lin",98
2,"hei",90
3,"ali",95
```

创建外表，通过列映射的方式重新排列文件中的列;

```
CREATE EXTERNAL TABLE ext_t2 (
id int AS (metadata$filecol1),
score int AS (metadata$filecol3)，
name char(10) AS (metadata$filecol2) )
```

```
LOCATION = 'oss://$ACCESS_ID:$ACCESS_KEY@$HOST/obcp/ob421/'
FORMAT = (
TYPE = 'CSV'
FIELD_DELIMITER = ','
)
PATTERN = 'data.csv';
```

查询外表.

注意：如果需要手动定义列映射，则自动列映射功能将会失效，所有列都需要手动定义映射

分布式外表文件管理

外表创建时，系统会将 LOCATION 中指定路径下匹配 PATTERN 的文件列表保存在 OceanBase 数据库的系统表中，外表扫描时会根据该列表来访问外部文件

在使用本地外表文件时，Oceanbase 支持分布式外表。当多台 OBServer 节点上的 secure_file_priv 目录均有匹配外表文件 PATTERN 的文件时，一张外表会同时关联多个节点上的外表文件

可通过系统视图 DBA_OB_EXTERNAL_TABLE_FILES 或 ALL_OB_EXTERNAL_TABLE_FILES 来查看外表的文件信息

外表创建后，如果外部目录新增了其他文件（该文件在 LOCATION 中指定的路径下且匹配 PATTERN），需要执行更新外表文件的操作，才能通过外表访问新增的文件

ALTER EXTERNAL TABLE table_name refresh;

注意：

1. 查询外表时，如果外表所访问的外部文件已删除，系统不会报错，会返回空行。
2. 由于外表所访问的文件由外部存储系统进行管理，当外部存储不可用时，查询外表将会报错。

序列与自增列

序列（SEQUENCE）是一个独立的数据对象，用于提供全局唯一的数值

默认设置下，序列仅保证单节点内单调有序，不保证全局有序

序列提供两个伪列：CURRVAL 是序列的当前值，NEXTVAL 是序列的下一个值

每个序列可以单独定义其 CACHE、ORDER 等属性

MySQL 模式和 Oracle 模式均支持序列，且实现机制相同

自增列（AUTO INCREMENT）是隶属于表的列，具备单调有序且唯一的属性

默认设置下，自增列保持全局有序

插入数据时不指定自增列的值，由数据库来自动生成单条有序的数值

每个表可以单独定义其自增列的 ORDER 属性，自增列的 CACHE 等属性则通过租户变量参数来控制

仅 MySQL 模式支持自增列

OceanBase 数据库 V4 版本中，自增列与序列的实现机制不同。自增列可以使用集中缓存来提供全局有序的递增值，而序列在使用缓存时无法提供全局有序的值

创建与管理序列

创建序列语法格式如下：

```
CREATE SEQUENCE sequence_name
[MINVALUE value | NOMINVALUE]
[MAXVALUE value | NOMAXVALUE]
[START WITH value]
[INCREMENT BY value]
[CACHE value | NOCACHE]
[ORDER | NOORDER]
[CYCLE | NOCYCLE];
ALTER EXTERNAL TABLE EXT_T4 REFRESH;
```

MINVALUE 和 MAXVALUE 指定最小值和最大值

START WITH 指定起始值

START WITH 指定起始值

CACHE 是为了性能缓存部分序列值，在并发高的时候使用。默认的 CACHE 大小为 20

CYCLE 指定序列值是否循环。如果循环，需要指定最大值或最小值

查看序列的定义：

MySQL 模式下，可以通过 DBA_SEQUENCES 或 DBA_OB_SEQUENCE_OBJECTS 视图查看创建的序列；

在 Oracle 模式下，可以通过 DBA_SEQUENCES、USER_SEQUENCES、ALL_SEQUENCES 视图查看创建的序列 。

使用 ALTER SEQUENCE 语句修改序列的属性，使用 DROP SEQUENCE 语句删除序列

序列的使用示例

创建序列，指定模式为 CACHE + NOORDER;

```
CREATE SEQUENCE seq1 CACHE 10000 NOORDER;
```

使用系统表 DBA_SEQUENCES 查看序列的定义，其中字段 LAST_NUMBER 为下一批 CACHE 的起始值;

```
SELECT * DBA_SEQUENCES WHERE SEQUENCE_NAME ='SEQ1';
```

引用序列

```
CREATE TABLE t1(id BIGINT);
INSERT INTO t1 (id) VALUES (seq1.nextval);
INSERT INTO t1 (id) VALUES (seq1.nextval);
INSERT INTO t1 (id) VALUES (seq1.nextval);
INSERT INTO t1 (id) VALUES (seq1.nextval);
```

注意：

1. 查看序列的 CURRVAL 之前，当前 SESSION 须先访问过该序列对象的 NEXTVAL，否则会报“sequence is not yet defined in this session” 错误。
2. 每次查询 NEXTVAL 都会推进 CURRVAL 值。

与 Oracle 数据库序列的差异

模式	Oracle 数据库	OceanBase 数据库
NOCACHE with ORDER	所有 Instance 不缓存任何序列，使用时从全局 CACHE 中获取，CACHE 根据请求的先后顺序返回序列值	所有 Instance 不缓存任何序列，使用时从全局 CACHE 中获取，CACHE 根据请求的先后顺序返回序列值。
NOCACHE with NOORDER	所有 Instance 不缓存任何序列，使用时从全局 CACHE 中获取，CACHE 可以根据繁忙程度延迟满足一些请求，导致 NOORDER	仅语法兼容，实际效果与 NOCACHE with ORDER 相同。
CACHE with ORDER	每个 Instance 都缓存相同的序列，使用之前需要通过全局锁来同步序列中的下一个可用位置	仅语法兼容，实际效果与 NOCACHE with ORDER 相同。
CACHE with NOORDER	每个 Instance 都缓存不同的序列，并且不需要全局同步 CACHE 状态。此时的值自然是 NOORDER	每个 Instance 都缓存不同的序列，并且不需要全局同步 CACHE 状态。此时的值自然是 NOORDER。

序列的使用建议

序列模式的选择建议：

OceanBase 数据库中，序列使用 ORDER with NO CACHE 模式时，每次调用 NEXTVAL 都会触发一次内部表 SELECT 与 UPDATE 操作，会影响数据库的性能

出于性能考虑，我们不建议使用默认的 CACHE with NOORDER 模式，不建议使用 ORDER with NO CACHE 模式

CACHE 大小的设置建议：

在创建序列时，由于默认的 CACHE 值只有 20，按照序列的使用频率，合理分配 CACHE 的大小。建议每日 CACHE 刷新的次数控制在 100 次以内。例如，对于单机 TPS 为 100 时，CACHE SIZE 建议设置为 360000

序列值跳变的原因：

缓存在 CACHE 中的序列值在发生服务器重启或者切主等操作时会失效，OceanBase 数据库在服务器重启或者切主后会重新分配、缓存序列值，导致序列值的跳变

创建自增列

在创建表时，如果为列指定 AUTO_INCREMENT 属性，则该列为自增列，自增列仅在 MySQL 模式可用

```
CREATE TABLE t1( id BIGINT NOT NULL AUTO_INCREMENT, name VARCHAR(50), gmt_create TIMESTAMP)
AUTO_INCREMENT = 1 AUTO_INCREMENT_MODE = 'ORDER';
```

AUTO_INCREMENT：设置列为自增列，每张表仅可定义一个自增列。建议使用 BIGINT 类型

AUTO_INCREMENT：自增列的起始值

AUTO_INCREMENT_MODE：自增列的自增模式，ORDER 为全局有序递增，NOORDER 为全局无序递增

除此之外，MySQL 租户提供了 3 个租户变量来控制自增列的自增起始值、自增步长和自增列缓存大小

系统变量	说明
auto_increment_cache_size	Global 级变量，用于设置自增的缓存个数，取值范围为 [1, 100000000]，默认值为 1000000
auto_increment_increment	Session 级变量，用于设置自增步长，取值范围为 [1, 65535]，默认值为 1
auto_increment_offset	Session 级变量，用于确定 AUTO_INCREMENT 列值的起点，取值范围为 [1, 65535]，默认值为 1

自增模式

OceanBase 数据库 V4 版本的自增列支持两种自增模式:

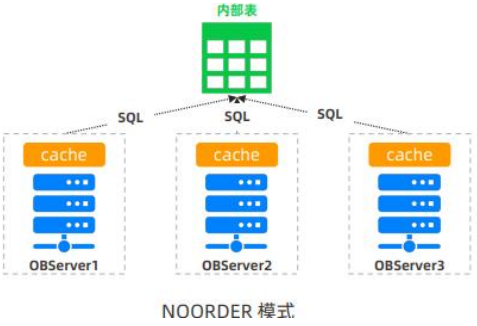
ORDER 模式（默认）：基于集中缓存的自增列。设置为该模式后，自增列的值全局递增且唯一

NOORDER 模式：基于分布式缓存的自增列，设置为该模式后，仅保证自增列的值全局唯一

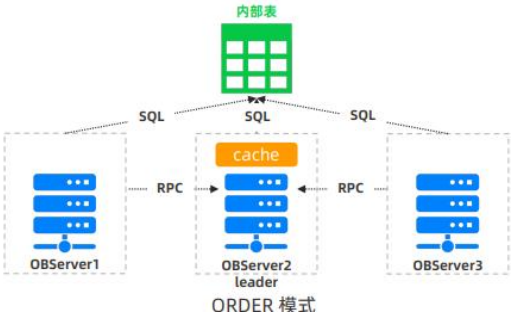
设置自增模式的方法：

设置表的属性为 auto_increment_mode 为 order 或 noorder

通过租户级配置项 default_auto_increment_mode 指定自增列的默认自增模式，其默认值为 order



NOORDER 模式



ORDER 模式

自增列的填充

在 INSERT 包含自增列的表时，可以用户指定自增列的值，也可以由 OceanBase 自动填充自增列的值

根据 INSERT 时指定的自增列值的不同，OceanBase 有不同的填充方式：

值为 NULL 或没有指定值，OceanBase 自动填充一个值，保证该值比该分区内已经分配的最大值还大

值为 0，且系统变量 SQL_MODE 中未设置 NO_AUTO_VALUE_ON_ZERO，OceanBase 视同插入了 NULL 值处理

值比分区内已分配的最大值小，在满足约束要求下插入成功，不影响自增列的下一个值的计算

值比分区内已分配的最大值大，则将插入的最大值作为自增列的当前最大值列来进行下一个的计算

管理与查看自增列

将普通列修改为自增列：

```
ALTER TABLE table_name MODIFY column_name data_type AUTO_INCREMENT;
```

注意：修改为自增列时，可以同时修改该列的数据类型，但需要确保修改后的数据类型是允许自增的数据类型

修改自增列的自增模式：

```
ALTER TABLE table_name AUTO_INCREMENT_MODE = 'NOORDER'|'ORDER';
```

修改自增列的当前起始值：

```
ALTER TABLE table_name AUTO_INCREMENT = next_value;
```

注意：

1. next_value 需要大于当前自增列已分配的最大值，否则设置不会生效;
2. next_value 设置完成，自增列下次分配的值就是 next_value

查看本 Session 最后一次插入的自增列的值

```
SELECT LAST_INSERT_ID();
SHOW VARIABLES LIKE 'LAST_INSERT_ID';
```

自增列的跳变

场景 1：在多机多分区环境下，NOORDER 的自增列无法保证全局有序，数据插入时存在自增列值跳变的现象

1. OBServer1 向内部表申请一段自增区间 [1,100]，插入记录时生成一个自增值 1
2. OBServer2 向内部表申请一段自增区间 [101,200]，插入记录时生成一个自增值 101
3. OBServer3 向内部表申请一段自增区间 [201,300]，插入记录时生成一个自增值 201
4. OBServer1 使用缓存 [2,100] 生成自增值 2
5. OBServer2 使用缓存 [102,200] 生成自增值 102

场景 2：通过 INSERT 语句插入指定的最大值

1. 当前自增列的已分配最大值是 1000
2. 新的 INSERT 指定了自增列的值为 2000

场景 3：机器重启或者发生切主

重启或切主时，原来的 OBServer 上的自增列缓存被丢弃，重启后重新缓存一段新的值

分布式数据库 SQL 调优

执行计划解读

执行计划调优概述

OceanBase 数据库优化器基于规则和代价模型，生成最优的执行计划。规则只适用于单表访问的索引选择，更多时候是基于代价的选择

基表访问的代价通常取决于扫描的行数、回表的行数

关联查询的代价通常取决于连接算法、连接的结果集大小

执行计划调优是人为优化或干预优化器的执行计划选择，常见的调优手段有：

优化索引：创建最优索引，提供更好的性能

搜集统计信息：提供全面、及时的统计信息，帮助优化器更准确地估算扫描和返回的数据量

指定执行计划：使用 Hint 或 Outline 来指定执行计划

执行计划演进：使用 SPM 来管理和演进执行计划

使用并行查询：通过并行处理来提高 AP 类查询的性能

改写 SQL：比如提供更优的过滤条件，避免使用复杂谓词条件，或者将标量子查询改写为 Join

执行计划

执行计划是对一条 SQL 查询语句在数据库中执行过程的描述。用户可以通过 EXPLAIN 命令查看优化器针对指定 SQL 生成的逻辑执行计划。

```
EXPLAIN [BASIC | EXTENDED ] explainable_stmt;
```

BASIC：用于最基本的计划展示，例如算子名称、所访问的表和索引名，以及过滤信息（Filter）等。BASIC 是默认的输出模式

EXTENDED：用于最详细的计划展示，可以帮助你理解优化器的执行计划选择，从而解决 SQL 执行计划问题

```
==========================================================
|ID|OPERATOR                  |NAME    |EST.ROWS|EST.TIME(us)|
----------------------------------------------------------
|0 |PX COORDINATOR            |        |1       |21          |
|1 |└─EXCHANGE OUT DISTR      |:EX10000|1       |21          |
|2 |  └─NESTED-LOOP JOIN      |        |1       |21          |
|3 |    ├─PX BLOCK ITERATOR   |        |1       |2           |
|4 |    │ └─TABLE FULL SCAN   |pt      |1       |2           |
|5 |    └─DISTRIBUTED TABLE GET|np     |1       |18          |
==========================================================
```

列名	含义
ID	执行树按照前序遍历的方式得到的编号（从0开始）
OPERATOR	操作算子的名称
NAME	对应表操作的表名（索引名）
EST. ROWS	估算的该操作算子的输出行数
EST.TIME	该操作算子的执行代价（微秒）

执行计划的总代价是构成执行计划的每个算子的代价的总和。在 EXPLAIN 信息中，上层父算子的代价包含下层所有子算子的代价，0 号算子的代价通常是整个执行计划的总代价

EXPLAIN 解读：Outputs & filters

```
CREATE TABLE p_tb(c1 INT PRIMARY KEY, c2 INT, KEY i_c2(c2)) PARTITION BY HASH(c1) PARTITIONS 4;
CREATE TABLE np_tb(c1 INT PRIMARY KEY, c2 INT, KEY i_c2(c2));
EXPLAIN SELECT /*+ parallel(8) use_nl(pt,np)*/ * FROM p_tb pt JOIN np_tb np ON pt.c1=np.c1 and np.c2=5;
```

```
==========================================================
|ID|OPERATOR                  |NAME    |EST.ROWS|EST.TIME(us)|
----------------------------------------------------------
|0 |PX COORDINATOR            |        |1       |21          |
|1 |└EXCHANGE OUT DISTR       |:EX10000|1       |21          |
|2 |  └NESTED-LOOP JOIN       |        |1       |21          |
|3 |    ├PX BLOCK ITERATOR    |        |1       |2           |
|4 |    │ └TABLE FULL SCAN    |pt      |1       |2           |
|5 |    └DISTRIBUTED TABLE GET|np      |1       |18          |
==========================================================
  0 - output(...), filter(nil), rowset=16
  1 - output(...), filter(nil), rowset=16 dop=8
  2 - output(...), filter(nil), rowset=16
      conds(nil), nl_params_([pt.c1(:0)]), use_batch=true
  3 - output(...), filter(nil), rowset=16
  4 - output(...), filter(nil), rowset=16
      access(...), partitions(p[0-3])
      is_index_back=false, is_global_index=false,
      range_key([pt.c1]), range(MIN ; MAX)always true
  5 - output(...), filter([np.c2 = 5]), rowset=16
      access(...), partitions(p0)
      is_index_back=false, is_global_index=false,
      filter_before_indexback[false],
      range_key([np.c1]), range(MIN ; MAX),
      range_cond([:0 = np.c1])
```

- rowset：rowset>1表示算子使用向量化计算。
- dop：dop>1表示算子使用并行执行。
- use_batch：nested loop join 是否使用batch join（一次 join 多条记录）。
- partitions：表扫描的分区范围。
- filter：过滤条件。
- is_index_back：是否需要回表扫描。
- range_key：使用的索引可以提供扫描范围的列，是一个数组。
- range：与 range_key 对应的每一个 key 的扫描范围，MIN~MAX 表示全部范围。
- range_cond：索引扫描的范围。

EXPLAIN 解读：Extended Info

```
EXPLAIN EXTENDED SELECT /*+ parallel(8) use_nl(pt,np)*/ * FROM p_tb pt JOIN np_tb np ON pt.c1=np.c1 and np.c2=5;
```

```
Used Hint:
------------------------------------
  /*+
      USE_NL("np")
      PARALLEL(8)
  */
```

生效的Hint

```
Qb name trace:
------------------------------------
  stmt_id:0, stmt_type:T_EXPLAIN
  stmt_id:1, SEL$1 > SEL$C6D21C0F
```

Query Block 信息
可用于Hint和Outline绑定

```
Outline Data:
------------------------------------
  /*+
      BEGIN_OUTLINE_DATA
      LEADING(@"SEL$C6D21C0F" ("test"."pt"@"SEL$1" "test"."np"@"SEL$1"))
      USE_NL(@"SEL$C6D21C0F" "test"."np"@"SEL$1")
      PQ_DISTRIBUTE(@"SEL$C6D21C0F" "test"."np"@"SEL$1" NONE ALL)
      PARALLEL(@"SEL$C6D21C0F" "pt"@"SEL$1" 8)
      FULL(@"SEL$C6D21C0F" "pt"@"SEL$1")
      FULL(@"SEL$C6D21C0F" "np"@"SEL$1")
      USE_DAS(@"SEL$C6D21C0F" "np"@"SEL$1")
      OUTER_TO_INNER(@"SEL$1")
      PARALLEL(8)
      OPTIMIZER_FEATURES_ENABLE('4.2.1.7')
      END_OUTLINE_DATA
  */
```

Outline 信息
可用于Outline绑定

```
Optimization Info:
------------------------------------
pt:
    table_rows:1
    physical_range_rows:1
    logical_range_rows:1
    index_back_rows:0
    output_rows:1
    table_dop:8
    dop_method:Global DOP
    avaiable_index_name:[i_c2, p_tb]
    pruned_index_name:[i_c2]
    stats version:0
    dynamic sampling level:1
np:
    table_rows:1
    physical_range_rows:1
    logical_range_rows:1
    index_back_rows:0
    output_rows:1
    table_dop:1
    dop_method:DAS DOP
    avaiable_index_name:[i_c2, np_tb]
    pruned_index_name:[i_c2]
    stats version:0
    dynamic sampling level:1
```

优化器估算信息

```
Plan Type:
    DISTRIBUTED
```

执行计划类型

```
Note:
    Degree of Parallelism is 8 because of hint
```

备注信息

常见执行算子

类型	算子
单表扫描	TABLE FULL SCAN， TABLE RANGE SCAN， TABLE SKIP SCAN， TABLE GET
两表关联	NESTED LOOP JOIN， HASH JOIN， MERGE JOIN

排序	SORT，TOP-N SORT
聚合	MERGE GROUP-BY，HASH GROUP-BY，WINDOW FUNCTION
分布式	DAS，EXCHANGE IN/OUT, PX
集合	UNION，EXCEPT，INTERSECT，MINUS
其他	LIMIT，MATERIAL，SUBPLAN，EXPRESSION，COUNT

单表扫描

Table Scan（单表扫描）算子是存储层和 SQL 层的接口，用于展示优化器选择哪个索引来访问数据

Table Scan 算子分为以下几类：

TABLE FULL SCAN（全表扫描）：完整扫描整个表或分区，该扫描方式不会使用索引过滤

TABLE RANGE SCAN（范围扫描）：扫描特定范围的记录，该扫描方式通常使用索引或主键来过滤扫描范围

该扫描方式通过匹配索引或主键的前缀列，来确定扫描的范围（query range）

TABLE SKIP SCAN（跳跃扫描）：跳跃地方式扫描不连续的几个范围，该扫描方式通常使用索引或主键来执行跳跃

该扫描方式通过匹配索引或主键的后缀列，来确定跳跃扫描的范围（skip scan range）

TABLE GET（主键唯一匹配）： 直接用主键唯一查询，匹配 0 行或者 1 行数据

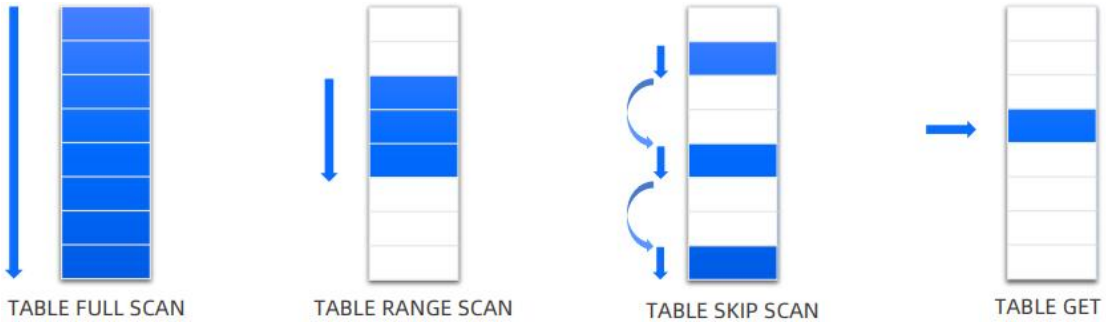


TABLE SKIP SCAN 的使用场景

OceanBase 数据库在满足如下限制条件时会尝试选择 TABLE SKIP SCAN：

表上有收集过统计信息

查询条件中包含联合索引的后缀列，并且不是其他索引最左前缀

前缀列包含的不同值的个数（NDV）较少

优化器比较 TABLE SKIP SCAN 和全表扫描的代价，发现 TABLE SKIP SCAN 代价更低

场景示例：假如索引定义为：CREATE INDEX I1 ON T1(C1,C2,C3)，以下场景下可以使用

TABLE SKIP SCAN。

索引扫描（等值匹配后缀列）：

SELECT * FROM T1 WHERE C2=1 AND C3=1

SELECT * FROM T1 WHERE C1<100 AND C2=1

计算索引前缀的唯一值及聚合：

SELECT DISTINCT(C1) FROM T1；

SELECT COUNT(DISTINCT(C1)) FROM T1；

计算索引后缀中第一列的最大、最小值：

SELECT MIN (C2) FROM T1

SELECT C1,C2,MIN (C3) FROM T1 GROUP BY C1,C2

注意：TABLE SKIP SCAN 是 OceanBase V4 的新功能，优化器默认关闭该路径的选择。用户可以在 SQL 中使用 INDEX_SS Hint 显式打开该路径的选择.

单表扫描的优化

相对于全表扫描，高效的索引扫描可以大幅降低扫描的记录数，提升单表扫描和关联查询的性能

最佳的索引设计应该满足以下特征：

最好的过滤性能：索引列能匹配最好的过滤条件，尽量少地扫描索引表，并且索引扫描后返回尽量少的记录

覆盖索引：索引列尽可能包含 SQL 查询所需的所有过滤列和投影列，避免回表访问

避免排序：索引列的顺序与 Order By 或者 Group By 的列顺序一致，避免对查询结果进行排序

在上面的示例中，为 Query 4 设计最佳索引时，需要按照索引的匹配规则以及查询条件的过滤率来确定索引列的顺序

Query 编号	SQL 语句	最佳索引
1	SELECT name,id FROM customers WHERE id=123456789 and name like'%ack%'	（id,name）
2	SELECT country,name,id FROM customers WHERE id=123456789 and namelike '%ack%'	（id,name,country）
3	SELECT country,name,count(*) FROM customers WHERE country='US' and name like '%ack%' GROUP BY country,name ORDER BY 2 DESC	（country, name）
4	SELECT c1,c2,c3 FROM t1 WHERE c1=1 and c2>100 and c3<10	?

在上面的示例中，为 Query 4 设计最佳索引时，需要按照索引的匹配规则以及查询条件的过滤率来确定索引列的顺序

索引的匹配规则

在 OceanBase 数据库中，索引是一个 B+ 树结构，索引扫描遵循 B+ 索引的匹配规则：

条件的先后顺序不影响索引能效，如 where A = ? and B = ? 和 where B = ? and A = ? 效

果相同

按照索引字段的顺序匹配，一旦前面的索引字段缺失，后面的索引字段无法参与匹配

遇到第一个范围查询字段后，后续的字段不参与匹配

查询条件中的索引字段如果不能参与匹配，依然可以提供扫描过滤

示例：假设存在索引 I1(A,B,C)，则以下查询条件的索引匹配字段和 Query Range 如表格所示

查询条件	索引匹配字段	Query Range
where A=? and B=? and C=?	A, B, C	A=? and B=? and C=?
where A=? and B>? and C=?	A, B	A=? and B>?
where A=? and C=?	A	A=?
where A>? and B=? and C=?	A	A>?
where A LIKE '%ABC' and B=?	无	(min,max)

上页的示例中，Query 4 的 c2>100 和 c3 <10 都是范围查询条件，假如 c3<10 的过滤性更好，则最佳索引设计为 （c1, c3, c2），而不是（c1, c2, c3）

两表关联

JOIN 操作合并来自两个数据源（例如表或视图）的输出，并返回一个数据源。多表 JOIN 的类型定义在 SQL 语句的 WHERE（非 ANSI）或 FROM ... JOIN (ANSI) 子句中。只要 FROM 子句中存在多个表，数据库就会执行连接

根据两表连接的结果集的保留规则，一般将连接分为以下四种类型

联接类型	英文表示形式	说明
内联接	INNER JOIN	内联接，结果为两个联接表中的匹配行的联接
左（外）联接	LEFT [OUTER] JOIN	结果包括左表（出现在 JOIN 子句最左边）中的所有行，不包括右表中的不匹配行
右（外）联接	RIGHT [OUTER] JOIN	结果包括右表（出现在 JOIN 子句最右边）中的所有行，不包括有左表中的不匹配的行
全（外）联接	FULL [OUTER] JOIN	结果包括所有联接中的所有行，不论他们是否匹配

JOIN 算法

JOIN 算法是将两个表的数据进行关联的执行方式，OceanBase 数据库支持的 JOIN 算法主要有以下三种：

算法	Nested Loop Join(NLJ)	Merge Join(MJ)	Hash Join(HJ)
执行方式	从外部表读取一行或一批记录； 根据外部表传递的连接键值匹配内部表的记录，并返回关联结果； 重复第一步，直到匹配完所有记录	同时读取外部表和内部表的数据； 同时对内外部表的数据按照连接键进行排序； 对排序后的内外部数据执行合并操作	读取外表表的数据； 将外部表的数据按照连接键进行 Hash 运算并放入 Hash 表中； 读取内部表的数据，并以 Loop 循环的方式去匹配 Hash 表
连接条件	可以用于任何连接条件	只能用于等值的连接条件	只能用于等值的连接条件

为关联查询创建索引

好的索引决定了单表访问的性能，也对关联查询至关重要。

在 Nested-Loop Join 中，如果外部表的数据集较大，而内部表上缺乏高效的索引提供关联字段的查询，则 Loop 循环的方式关联内部表的性能将为非常差

在 Merge Join 中，如果没有索引提供关联字段的排序，大数据集的排序将会是非常耗时的操作

示例：EMPLOYEE 表与 DEPARTMENT 表的关联查询，两张表在关联字段 DEPARTMENT_ID 上均没有创建索引， EMPLOYEE 表的主键是 EMPLOYEE_ID

```
SELECT E.FIRST_NAME,E.LAST_NAME,D.DEPARTMENT_NAME
FROM EMPLOYEE E JOIN DEPARTMENT D ON E.DEPARTMENT_ID=D.DEPARTMENT_ID
WHERE E.EMPLOYEE_ID <=10;
```

DEPARTMENT 表在关联字段上未创建索引时的执行计划

DEPARTMENT 表在关联字段上创建索引后的执行计划

执行计划管理

使用 HINT 指定执行路径

通常，优化器会选择最佳的执行计划，不需要用户使用 Hint 指定。但在某些场景下，优化器生成的执行计划可能不满足用户的要求，这时就需要用户使用 Hint 来指定生成某种执行计划

Hint 是一种特殊的 SQL 语句注释，它以一种固定的格式和位置出现在 SQL 语句文本中，从而影响优化器对执行计划的选择

{ DELETE | INSERT | SELECT | UPDATE | REPLACE } /*+ [hint_text][,hint_text]... */

HINT 的位置必须紧跟在 DELETE、INSERT、MERGE、SELECT 或 UPDATE 关键字后面

与普通 SQL 注释所不同的是，HINT 需要在注释的左标记后（'/*' 符号）增加了一个'+’

如果一次使用多个 Hint，则 Hint 间需要用空格或者逗号分隔

HINT 只影响优化器生成计划的逻辑，而不影响 SQL 语句的语义

如果使用 MySQL 的 C 客户端执行带 Hint 的 SQL 语句，需要使用 -c 选项登录，否则 MySQL 客户端会将 Hint 作为注释从用户 SQL 语句中去除，导致系统无法收到用户 Hint

Hint 示例：指定 SQL 超时时间为 60000000us，即 60s

```
EXPLAIN EXTENDED SELECT/*+ query_timeout(60000000) */ * FROM gtx_t1;
```

关于表扫描的 HINT

可以使用 Hint 指定表扫描的执行方式，比如使用哪个索引

Hint 语法	Hint 语义
INDEX(table_name index_name)	设置表索引，index_name 为 PRIMARY 时表示主键
FULL(table_name)	设置表执行计划为主表（主键），等价于 INDEX(TBL_NAME PRIMARY)
INDEX_SS(table_name index_name)	设置表执行计划为 TABLE SKIP SCAN

关于关联的 HINT

可以使用 Hint 指定表关联的 Join 算法和 Join 顺序

Hint 语法	Hint 语义
ORDERED	设置按照 SQL 中的顺序进行联接
LEADING(table_name_list)	设置联接顺序
USE_NL(table_name_list)	设置指定表在作为右表时使用 Nested Loop Join
USE_MERGE(table_name_list)	设置指定表在作为右表时使用 Merge Join
USE_HASH(table_name_list)	设置指定表在作为右表时使用 Hash Join

Hint 示例：指定两表关联的 Join 算法为 Nested Loop Join，并且 Join 顺序为 t2->t1

```
SELECT/*+ USE_NL(t1),LEADING(t2 t1) */* FROM tb1 t1,tb2 t2 WHERE t1.c1=t2.c1 AND t1.c2>100;
```

关于 SQL 执行的 HINT

除了使用 Hint 影响执行计划的选择外，还可以使用 Hint 控制 SQL 语句的执行，常见的 Hint 如下

Hint 语法	Hint 语义
QUERY_TIMEOUT(int)	设置 SQL 的最长执行时间。

PARALLEL(N)	设置 SQL 的并行执行线程数。
MAX_CONCURRENT(N)	设置 SQL 的最大并发执行线程数。
READ_CONSISTENCY(weak|strong)	设置读取的模式为弱一致或强一致性读。

Hint 示例：设置 SQL 的并行执行线程数为 5

SELECT /*+ PARALLEL(5) */ last_name FROM emp;

CREATE OUTLINE 绑定执行计划

使用 Hint 指定执行计划需要修改 SQL 语句，通常需要应用开发人员修改程序。OceanBase 数据库同时提供了 Outline 从数据库侧为 SQL 绑定执行计划 Hint，而无需修改 SQL 语句

CREATE OUTLINE otl1 ON SELECT/*+ INDEX(t1 idx_c2)*/ * FROM t1 WHERE c2 = 1;

OceanBase 数据库支持通过两种方式创建 Outline

一种是使用 SQL_TEXT 创建:CREATE [OR REPLACE] OUTLINE outline_name ON stmt_with_hint [TO

];

指定 OR REPLACE 后，可以对已经存在执行计划进行替换

stmt_with_hint 为一个带有 Hint 和原始参数的 DML 语句，OceanBase 会对 stmt 中的常量进行参数化处理。如果数据库接受的 SQL 参数化后与 stmt 去掉 Hint 参数化文本相同，则将该 SQL 绑定 stmt 中 Hint 生成执行计划

指定 TO target_stmt 表示该 Outline 只对 target_stmt 生效，不会使用 stmt_with_hint 的参数化文本做匹配

一种是使用 SQL_ID 创建

CREATE OUTLINE outline_name ON sql_id USING HINT hint_text;

注意：CREATE OUTLINE 需要在 SQL 语句所在的 Schema 下执行，否则 Outline 不会生效！

获取 SQL_ID 与 OUTLINE_DATA

使用 SQL_ID 创建 Outline 时，可以通过以下方法获取 SQL 的 SQL_ID：

通过查询 SQL 监控的系统视图，比如 GV$OB_PLAN_CACHE_PLAN_STAT 、GV$OB_SQL_AUDIT 获取

通过参数化的原始 SQL，使用 MD5 生成 SQL_ID

```
#!/usr/bin/python
import hashlib
sql_text='select * from t1 where c1=? and c2 = ?'
sql_id=hashlib.md5(sql_text.encode('utf-8')).hexdigest().upper()
print(sql_id)
```

CREATE OUTLINE 语句中的 hint 可以使用或参考 EXPLAIN EXTENDED 结果中的 outline data

```
CREATE OUTLINE otl_idx_c2 ON
```

```
"ED570339F2C856BA96008A29EDF04C74" USING HINT
/*+
BEGIN_OUTLINE_DATA
INDEX(@"SEL$1" "test.t1"@"SEL$1" "idx_c2")
END_OUTLINE_DATA
*/;
```

CREATE FORMAT OUTLINE

SQL 执行时会根据 SQL 文本 或者 SQL ID 与 创建 Outline 时使用的 SQL 文本或 SQL ID 严格匹配。例如，对“select * from A”创建了 Outline，在 SQL 执行时它的匹配结果如下

SQL 文本	匹配结果
select * from A;	匹配
select * from A;	不匹配
select * from A;	不匹配

OceanBase 从 V4.2.2 版本开始，支持以模糊匹配的方式为 SQL 绑定 Outline，即 Format Outline

```
CREATE [OR REPLACE] FORMAT OUTLINE outline_name ON stmt_with_hint [ TO <target_stmt> ];
CREATE FORMAT OUTLINE outline_name ON sql_id USING HINT hint_text;
```

使用 SQL text 时：数据库会对 SQL text 做快速词法解析（fast parser），生成 format SQL text

使用 SQL ID 时：需要使用 format SQL text 对应的 SQL_ID，可以直接从 GV$OB_SQL_AUDIT 的 FORMAT_SQL_ID 字段获取，也可以使用 STATEMENT_DIGEST_TEXT 函数获取 format SQL text，然后通过 md5 运算得到

确认 OUTLINE 是否生效

创建 Outline 后，可以通过系统视图 DBA_OB_OUTLINES 或 DBA_OB_FORMAT_OUTLINES 确认是否创建成功

```
SELECT * FROM DBA_OB_OUTLINES WHERE OUTLINE_NAME = 'otl_idx_c2'\G
*************************** 1. row ***************************
        tenant_id: 1001
      database_id: 1100611139404776
       outline_id: 1100611139404777
    database_name: test
     outline_name: otl_idx_c2
visible_signature: SELECT * FROM t1 WHERE c2 = ?
         sql_text: SELECT/*+ index(t1 idx_c2)*/ * FROM t1 WHERE c2 = 1
   outline_target:
      outline_sql: SELECT /*+ BEGIN_OUTLINE_DATA INDEX(@"SEL$1" "test.t1"@"SEL$1" "idx_c2") END_OUTLINE_DATA*/*
FROM t1 WHERE c2 = 1
```

执行绑定的 SQL，从 Plan Cache 中判断 SQL 是否通过绑定的 Outline 生成了新执行计划

```
SELECT SQL_ID, PLAN_ID, STATEMENT, OUTLINE_ID, OUTLINE_DATA FROM oceanbase.GV$OB_PLAN_CACHE_PLAN_STAT WHERE
SQL_ID='ED570339F2C856BA96008A29EDF04C74'\G
*************************** 1. row ***************************
      sql_id: ED570339F2C856BA96008A29EDF04C74
     plan_id: 17225
   statement: SELECT * FROM t1 WHERE c2 = ?
  outline_id: 1100611139404777
outline_data: /*+ BEGIN_OUTLINE_DATA INDEX(@"SEL$1" "test.t1"@"SEL$1" "idx_c2") END_OUTLINE_DATA*/
```

如果 outline_id 与在 DBA_OB_OUTLINES 中查到的 outline_id 相同，则表示是按绑定的 Outline 生成的执行计划，否则不是

执行计划管理（SPM）

SQL Plan Management（SPM）是一种防止计划回退的机制，能够确保新生成的计划在经过验证后才被使用，以保证计划性能不断优化和更新

SPM 的功能

捕捉基线计划：SPM 自动捕捉基线执行计划，并提供固定基线计划的功能，以防止执行计划的变动

执行计划演进：当因为统计信息变化等原因触发执行计划生成时，SPM 会自动进行执行计划的演进，总是在基线计划中保留更优的执行计划

SPM 的执行机制

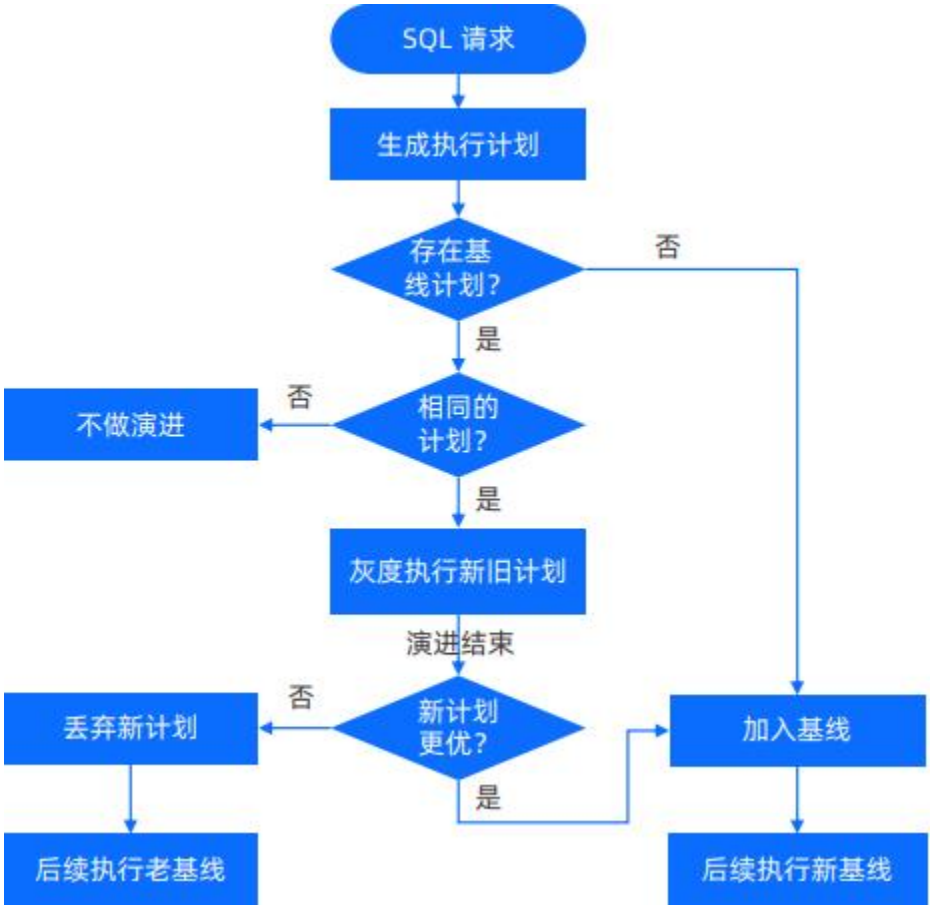
当 SQL 没有可复现的 Baseline 计划时，将新生成的执行计划存入 Baseline

如果有 FIXED 且可复现的 Baseline 计划，优先使用 FIXED 计划，不做演进

如果存在可复现的 Baseline 计划，且 Baseline 不是 FIXED，则对新计划与 Baseline 进行灰度演进

一部分 SQL 执行使用新计划，一部分使用 Baseline 计划

如果新计划更优，则将新计划存入 Baseline，并在 Plan Cache 中替代老计划



开启 SPM 功能

SPM 功能有其特定的使用场景，在 OceanBase V4.2 版本中 SPM 功能默认是关闭的.

需要设置以下系统变量来开启 SPM 功能：

optimizer_capture_sql_plan_baselines：表示当新的计划优于基线计划时，是否捕获新的计划作为基线，默认值为 false

optimizer_use_sql_plan_baselines：表示当 SQL 生成新的计划时是否使用基线计划进行演进，默认值为 false

```
SET [GLOBAL|SESSION] optimizer_use_sql_plan_baselines = on;
SET [GLOBAL|SESSION] optimizer_capture_sql_plan_baselines = on;
```

SPM 功能的典型使用场景：

大小账号导致的计划走偏：业务表中数据存在倾斜时，优化器生成计划时可能因为使用了特定的大/小账号而导致计划走偏。如果业务大小账号的查询量是均匀的，使用 SPM 可以及时验证执行计划在小/大账号的场景下性能不优，从而避免使用错误的执行计划

分页查询导致的计划走偏：分页查询由于代价模型计算不准确，或者特定索引上的数据分布可以加快执行速度的场景下，优化器容易选错索引导致计划走偏。这种场景下，如果基线计划是正确的索引，那么开启 SPM 可以避免因为计划走偏导致的分页查询性能下降

SPM 功能的使用限制：由于演进的结果是暂存在 OBServer 节点的 Local Cache 中，然后定期同步到 Inner Table 中，所以任何 OBServer 节点的演进结果对于其它的 OBServer 节点来讲是无法立即感知的

注意：目前 OceanBase 数据库社区版暂不支持 SPM 功能

DBMS_SPM 包

OceanBase 提供了类似 Oracle 数据库的 DBMS_SPM 包来管理 SPM 和 Baseline 计划，在 Oracle 和 MySQL 租户均可以使用。

子程序名	类型	描述
LOAD_PLANS_FROM_CURSOR_CACHE	Function	从 Cursor Cache 中读取一条或多条 SQL 计划作为计划基线
ALTER_SQL_PLAN_BASELINE	Function	修改基线中一个或一组计划的属性
DROP_SQL_PLAN_BASELINE	Function	丢弃一个或多个计划基线
ACCEPT_SQL_PLAN_BASELINE	Function	结束一个正在演进的任务，并将好的计划转为基线
CANCEL_EVOLVE_TASK DROP_EVOLVE_TASK	Function	取消一个正在演进的任务
CONFIGURE	Function	设置 SPM 的一些参数

Function：返回值表示被修改的基线数量。可以直接使用 SELECT 语句调用执行

SELECT DBMS_SPM.DROP_SQL_PLAN_BASELINE('testdb','521413248E4F1EC3F0FEDD9FF9E161C5','4392083053520233360');

Procedure：不返回值，可以使用 CALL 语句调用执行

CALL DBMS_SPM.ACCEPT_SQL_PLAN_BASELINE('testdb','521413248E4F1EC3F0FEDD9FF9E161C5','4392083053520233360');

添加 Baseline 计划

SPM 管理机制中，同一个 SQL 可以同时有多个 Baseline 计划，SQL 执行时优先选择最优的 Baseline 计划。

LOAD_PLANS_FROM_CURSOR_CACHE：从 Plan Cache 中读取一条或多条 SQL 的计划作为基线计划。这个过程不会影响已经存在的基线计划，只会增加新的基线计划。

```
###oracle 模式
DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE (
sql_id IN VARCHAR,
plan_hash_value IN NUMBER := NULL,
fixed IN VARCHAR := 'NO',
enabled IN VARCHAR := 'YES'
)
RETURN PLS_INTEGER;
```

```
###MySQL 模式
DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE (
database_name VARCHAR(65535),
sql_id VARCHAR(65535),
plan_hash_value DECIMAL DEFAULT NULL,
is_fixed VARCHAR(65535) DEFAULT 'NO',
enabled VARCHAR(65535) DEFAULT 'YES'
) RETURN DECIMAL;
```

参数名称	说明
database_name	数据库名称，仅 MySQL 模式下需要指定该参数
sql_id	要加载的 SQL_ID
plan_hash_value	要加载的计划的 Plan Hash Value。如果值为 NULL，加载 Plan Cache 中指定 SQL 的所有计划作为计划基线

固定 Baseline 计划

可以修改 Baseline 计划的属性，比如 Fix 该计划，让 SQL 执行总是优先使用 Fixed 基线计划

ALTER_SQL_PLAN_BASELINE：修改基线计划的属性

```
###ORACLE 模式
DBMS_SPM.ALTER_SQL_PLAN_BASELINE (
sql_handle IN VARCHAR,
plan_name IN VARCHAR := NULL,
attribute_name IN VARCHAR,
attribute_value IN VARCHAR
)
RETURN PLS_INTEGER;

###MySQL 模式
DBMS_SPM.ALTER_SQL_PLAN_BASELINE (
database_name VARCHAR(65535),
sql_handle VARCHAR(65535),
plan_name VARCHAR(65535) DEFAULT NULL,
attribute_name VARCHAR(65535),
attribute_value VARCHAR(65535)
) RETURN DECIMAL;
```

attribute_name	attribute_value	说明
enabled	'YES' 或 'NO'	'YES' 表示计划是有效的

fixed	'YES' 或 'NO'	'YES' 表示优先使用当前基线计划，并且不进行自动演进
autopurge	'YES' 或 'NO'	'YES' 表示计划一段时候（默认为 1 年）没有使用后会被自动淘汰；'NO' 表示永不淘汰

典型使用场景：SQL 因为数据量变化和统计信息收集导致执行计划不稳，可以将能够提供稳定性能的基线计划固定住，让 SQL 执行尽量使用该固定计划

查看 Baseline 计划

系统视图 DBA_SQL_PLAN_BASELINES 记录了 SQL 的基线计划的状态和执行统计信息

DBA_SQL_PLAN_BASELINES

```
SELECT SQL_HANDLE, SQL_TEXT, PLAN_NAME, ENABLED, FIXED,
REPRODUCED, AUTOPURGE, EXECUTIONS, ELAPSED_TIME,
CPU_TIME FROM oceanbase.DBA_SQL_PLAN_BASELINES\G;
*********************** 1. row ***********************
SQL_HANDLE: 521413248E4F1EC3F0FEDD9FF9E161C5
SQL_TEXT: select * from t1 where c2='abc'
PLAN_NAME: 9518032312681390441
ENABLED: YES
FIXED: NO
REPRODUCED: YES
AUTOPURGE: YES
EXECUTIONS: 1
ELAPSED_TIME: 14962
CPU_TIME: 6638
*********************** 2. row ***********************
SQL_HANDLE: 521413248E4F1EC3F0FEDD9FF9E161C5
SQL_TEXT: select * from t1 where c2='abc'
PLAN_NAME: 4392083053520233360
ENABLED: YES
FIXED: NO
REPRODUCED: YES
AUTOPURGE: YES
EXECUTIONS: 1
ELAPSED_TIME: 1719
CPU_TIME: 391
```

重要字段	说明
SQL_HANDLE	SQL 的标识符，也就是 SQL_ID
SQL_TEXT	SQL 文本
PLAN_NAME	Plan Hash Value

ENABLED	是否启用该 Baseline 执行计划
FIXED	是否固定该 Baseline 执行计划
REPRODUCED	该 Baseline 计划是否依然有效
AUTOPURGE	是否允许自动清理该 Baseline 执行计划
EXECUTIONS	Baseline 计划创建时计划的执行次数
ELAPSED_TIME	Baseline 计划创建时计划的总耗时
CPU_TIME	Baseline 计划创建时计划的总 CPU 时间

查看执行计划演进

系统视图 GV$OB_PLAN_CACHE_PLAN_STAT 记录 Plan Cache 中计划的详细信息及其执行统计信息，也包含演进中的执行计划的信息。

GV$OB_PLAN_CACHE_PLAN_STAT

```
SELECT SQL_ID, STATEMENT, PLAN_HASH, AVG_EXE_USEC,
EXECUTIONS, EVOLUTION, EVO_EXECUTIO, EVO_CPU_TIME
FROM oceanbase.GV$OB_PLAN_CACHE_PLAN_STAT
WHERE SQL_ID='521413248E4F1EC3F0FEDD9FF9E161C5'\G
********************** 1. row **********************
SQL_ID: 521413248E4F1EC3F0FEDD9FF9E161C5
STATEMENT: select * from t1 where c2=?
PLAN_HASH: 4392083053520233360
AVG_EXE_USEC: 5484
EXECUTIONS: 12
EVOLUTION: 1
EVO_EXECUTIONS: 12
EVO_CPU_TIME: 54692
********************** 2. row **********************
SQL_ID: 521413248E4F1EC3F0FEDD9FF9E161C5
STATEMENT: select * from t1 where c2=?
PLAN_HASH: 9518032312681390441
AVG_EXE_USEC: 4728
EVOLUTION: 1
EVO_EXECUTIONS: 17
EVO_CPU_TIME: 72993
```

相关字段	说明

SQL_ID	SQL 的标识符
STATEMENT	SQL 参数化后的文本
PLAN_HASH	Plan 的全局唯一识别号
AVG_EXE_USEC	该计划的平均执行时间
EXECUTIONS	该计划总的执行次数
EVOLUTION	该执行计划是否在演进中
EVO_EXECUTIONS	演进过程中的执行次数
EVO_CPU_TIME	演进执行的总 CPU 耗时

开启 SPM 并管理基线计划

本示例展示如何开启 SPM，以及如何干预演进任务、管理 Baseline 计划

开启会话级别的 SPM 功能

```
SET optimizer_capture_sql_plan_baselines=on; //* 开启基线计划捕捉
SET optimizer_use_sql_plan_baselines=on; //* 开启执行计划演进
```

执行 SQL，检查 SPM 捕捉基线计划的结果

```
SELECT * FROM oceanbase.DBA_SQL_PLAN_BASELINES WHERE SQL_TEXT LIKE ...; //* 确认当前有 0 个基线计划
SELECT * FROM t1 WHERE C2='ABC'； //* 首次运行，生成计划，被捕捉为基线
SELECT * FROM oceanbase.DBA_SQL_PLAN_BASELINES WHERE SQL_HANDLE= ...; //* 确认当前有 1 个基线计划
```

创建索引，并将演进计划接受为基线计划

```
CREATE INDEX t1_i2 ON t1(c2); //* 确认没有基线计划
SELECT * FROM T1 WHERE C2='ABC'； //* 重复多次执行该 SQL
SELECT * FROM oceanbase.GV$OB_PLAN_CACHE_PLAN_STAT WHERE SQL_ID=...; //* 确认 Plan Cache 中两个计划在演进中
CALL DBMS_SPM.ACCEPT_SQL_PLAN_BASELINE('testdb','xxxx','xxxx'); //* 将演进中的新计划接受为基线计划
SELECT * FROM oceanbase.DBA_SQL_PLAN_BASELINES WHERE SQL_ID=...; //* 确认当前有 2 个基线计划
```

注意：在 MySQL 模式中，OceanBase 系统视图属于 oceanbase 数据库；在 Oracle 模式中，OceanBase 系统视图属于 SYS 用户。

统计信息收集

统计信息概览

统计信息描述了数据库中数据的分布信息，是数据库优化器进行执行计划代价估算和执行计划选择的主要依据，对 SQL 性能优化至关重要。在 OceanBase V4 版本，数据库可以收集、使用的统计信息如下图所示：

统计信息的作用示例

创建结构相同的两张表 GTX_T1 和 GTX_T2（仅展示 GTX_T1 表的 DDL）

CREATE TABLE gtx_t1(c1 INT, c2 INT, c3 INT, PRIMARY KEY(c1));

CREATE INDEX gtx_i1 ON gtx_t1(c3,c2);

给两张表初始化相同的数据，并制造数据分布的倾斜（仅展示 GTX_T1 表的插入）

INSERT INTO gtx_t1 (c1,c2,c3) SELECT LEVEL, MOD(LEVEL,3), LEVEL FROM DUAL CONNECT BY LEVEL <= 10000;

INSERT INTO gtx_t1 (c1,c2,c3) VAlUES(9999999,9999999,9999999);

不收集统计信息，执行两张表的关联查询

```
EXPLAIN SELECT * FROM gtx_t1 l,gtx_t2 i WHERE l.c1=i.c1
and l.c2>100;

==========================================================
|ID|OPERATOR          |NAME|EST.ROWS|EST.TIME(us)|
----------------------------------------------------------
|0 |MERGE JOIN        |    |982     |1311        |
|1 |├─TABLE FULL SCAN|L   |1001    |465         |
|2 |└─TABLE FULL SCAN|I   |10002   |593         |
==========================================================
```

手动收集统计信息，再次执行两张表的关联查询

```
CALL DBMS_STATS.GATHER_TABLE_STATS('TESTDB', 'GTX_T1',
method_opt=>'FOR ALL INDEXED COLUMNS');
CALL DBMS_STATS.GATHER_TABLE_STATS('TESTDB', 'GTX_T2',
method_opt=>'FOR ALL INDEXED COLUMNS');

============================================================
|ID|OPERATOR                  |NAME|EST.ROWS|EST.TIME(us)|
------------------------------------------------------------
|0 |NESTED-LOOP JOIN          |    |2       |435         |
|1 |├─TABLE FULL SCAN         |L   |2       |417         |
|2 |└─DISTRIBUTED TABLE GET   |I   |1       |18          |
============================================================
```

思考题：在收集统计信息时，优化器是如何进行估算匹配记录数的？

统计信息的收集

在 OceanBase V4 版本中，既可以由用户手动收集特定的数据库、表、分区的统计信息，也可以由数据库依据算法智能识别需要收集统计信息的对象，并自动收集。

手动收集统计信息：DBMS_STATS

OceanBase 数据库优化器针对手动统计信息收集提供了两种方式：DBMS_STATS（推荐）、ANALYZE 命令行

DBMS_STATS 收集统计信息的程序主要有：

GATHER_SCHEMA_STATS：搜集指定 Schema（Database）下所有表和索引的统计信息，也可以收集列的直方图信息

GATHER_TABLE_STATS：搜集指定表的统计信息，也可以收集表上指定列的直方图信息

GATHER_INDEX_STATS： 搜集指定索引的统计信息

```
PROCEDURE gather_table_stats (
ownname VARCHAR2,
tabname VARCHAR2,
```

```
partname VARCHAR2 DEFAULT NULL,
estimate_percent NUMBER DEFAULT AUTO_SAMPLE_SIZE,
block_sample BOOLEAN DEFAULT FALSE,
method_opt VARCHAR2 DEFAULT DEFAULT_METHOD_OPT,
degree NUMBER DEFAULT NULL,
granularity VARCHAR2 DEFAULT DEFAULT_GRANULARITY,
cascade BOOLEAN DEFAULT NULL,
stattab VARCHAR2 DEFAULT NULL,
statid VARCHAR2 DEFAULT NULL,
statown VARCHAR2 DEFAULT NULL,
no_invalidate BOOLEAN DEFAULT FALSE,
stattype VARCHAR2 DEFAULT 'DATA',
force BOOLEAN DEFAULT FALSE
);
CALL DBMS_STATS.GATHER_SCHEMA_STATS ('testdb',
degree=>64,no_invalidate=>TRUE);
CALL DBMS_STATS.GATHER_TABLE_STATS('testdb', 'tb1',
degree=>4, force=>TRUE);
CALL DBMS_STATS.GATHER_INDEX_STATS('testdb', 'tb1_ix1',
degree=>4, tabname=>'tb1');
```

method_opt：指定搜集列的直方图信息

no_invalidate：是否 invalidate Plan Cache 中缓存的执行计划

force：是否强制搜集统计信息，即使统计信息被锁定

DBMS_STATS

除非指定了 method_opt，搜集 schema 或 table 的统计信息时，默认不搜集列的直方图信息

```
method_opt=>
FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]
| FOR COLUMNS [size clause] column [size_clause] [,column [size_clause]...]
size_clause: SIZE integer | SIZE REPEAT | SIZE AUTO| SIZE SKEWONLY
```

SIZE integer：指定收集列的直方图桶的个数，范围在 [1-2048]

SIZE REPEAT：仅仅只收集已经有收集过的直方图的列的直方图；使用之前收集直方图设置的桶个数

SIZE AUTO：由数据库优化器来决定是否收集列的直方图，取决于列的使用情况，桶个数使用默认值为 254

SIZE SKEWONLY：仅仅只收集数据分布不均匀的列的直方图；直方图桶个数使用默认值为 254

```
CALL DBMS_STATS.GATHER_TABLE_STATS(
'testdb', 'tb1', degree=>64, method_opt=>'FOR ALL COLUMNS SIZE SKEWONLY');
CALL DBMS_STATS.GATHER_TABLE_STATS(
'testdb', 'tb1', degree=>4, method_opt=>'FOR COLUMNS C2 SIZE 5');
```

ANALYZE

ANALYZE 语法（Oracle 模式）：

```
ANALYZE TABLE table_name [ PARTITION (part_name [,...])| SUBPARTITION(subpart_name [,...])]
COMPUTE STATISTICS [analyze_for_clause]
| ESTIMATE STATISTICS [analyze_for_clause] [SAMPLE INTNUM {ROWS | PERCENTAGE}]
analyze_for_clause:
FOR TABLE
| FOR ALL [INDEXED | HIDDEN] COLUMNS [SIZE integer]
| FOR COLUMNS [SIZE integer] column [SIZE integer] [,column [SIZE integer]...]
```

COMPUTE STATISTICS | ESTIMATE STATISTICS： 计算分析对象的精确统计信息，并将其存储在数据字典中

FOR TABLE：指定仅收集表和列的统计信息，不收集列的直方图信息

FOR ALL [INDEXED | HIDDEN] COLUMNS：收集表的统计信息，同时收集指定列（所有列、索引列或隐藏列）的直方图信息

SIZE：指定直方图中的最大存储桶数，默认值是 75

示例:ANALYZE TABLE tbl1 COMPUTE STATISTICS FOR ALL INDEXED COLUMNS SIZE 100;

注意：MySQL 模式的 ANALYZE 命令功能比较简单，建议使用 DBMS_STATS 来进行统计信息的手动收集

在线收集统计信息

在线统计信息收集指在批量插入或导入数据时，数据库优化器同时收集统计信息，不用手动调用系统包。

支持在线统计信息搜集的场景主要有:

CREATE TABLE AS SELECT：默认启动在线收集统计信息功能

INSERT INTO … SELECT：满足以下任一条件时，开启在线收集统计信息功能

使用 PDML（Parallel DML）

使用旁路导入（Append）

指定 Hint：GATHER_OPTIMIZER_STATISTICS

示例:

使用 PDML：INSERT /*+PARALLEL(4) ENABLE_PARALLEL_DML*/ INTO T1 SELECT * FROM FROM T2;

使用旁路导入：INSERT /*+APPEND PARALLEL(4) ENABLE_PARALLEL_DML*/ INTO T1 SELECT * FROM FROM T2;

指定 Hint：INSERT /*+GATHER_OPTIMIZER_STATISTICS*/ INTO T1 SELECT * FROM FROM T2;

自动收集统计信息

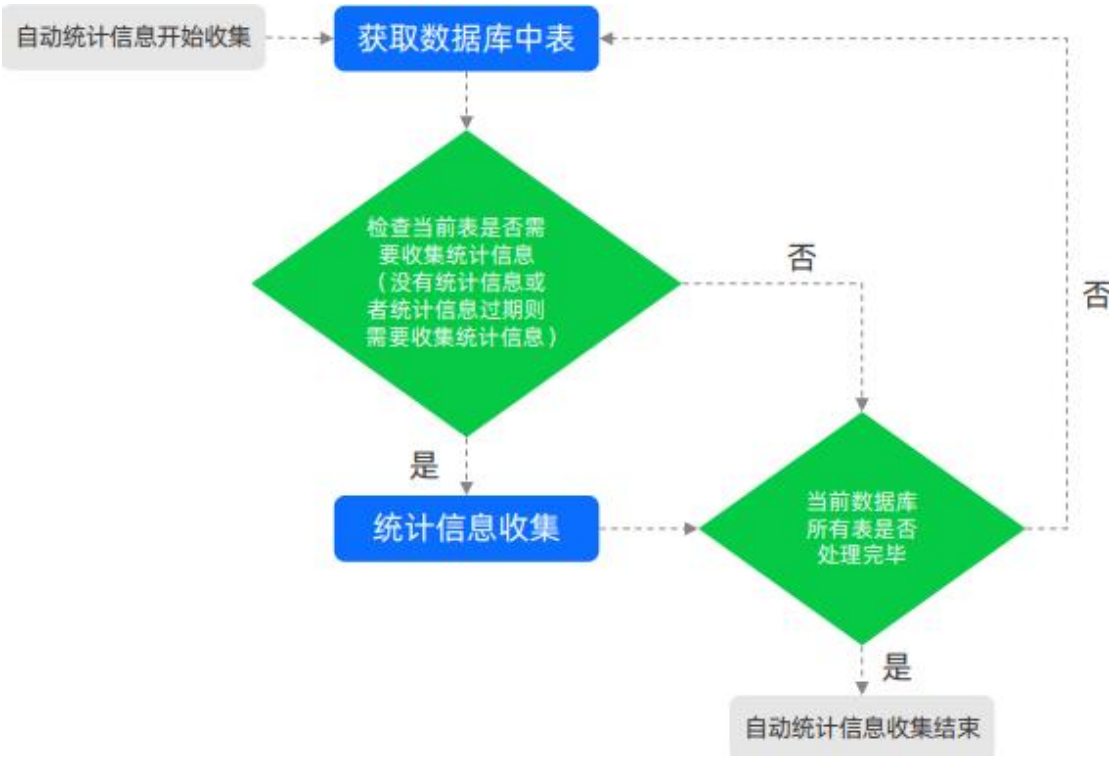
OceanBase V4 提供了自动收集统计信息的机制，数据库根据表的数据变化来判断其统计信息是否过期，并在合适的时间窗口按照统计信息收集策略自动为统计信息过期的表收集统

计信息.

过期判断：从上一次收集统计信息后，该表或分区的增/删/改的比例达到阈值，则认为该表或分区的统计信息过期

收集策略：统计信息收集的粒度、范围以及并发度等，还有统计信息的过期阈值，以及历史统计信息的保留时间等

收集窗口：数据库优化器定义了周一到周日 7 个自动统计信息收集任务，按照配置的时间窗口进行统计信息的自动收集



数据变更的监控

自动统计信息收集中，判断一个表的统计信息是否过期，主要依据上一次统计信息收集时间到本次收集期间该表的做增/删/改的比例（默认 10%），OceanBase 数据库优化器也提供了相关视图用于查询一张表的增/删/改次数

模式	视图名称	描述
Oracle	ALL_TAB_MODIFICATIONS	查询表所有的 DML stats
Oracle	DBA_TAB_MODIFICATIONS	查询表所有的 DML stats
Oracle	USER_TAB_MODIFICATIONS	查询表所有的 DML stats
MySQL	oceanbase.DBA_TAB_MODIFICATIONS	查询表所有的 DML stats

```
obclient [oceanbase]> select * from DBA_TAB_MODIFICATIONS where table_owner='gtx' ;
+-------------+------------+----------------+---------+---------+---------+------------+-----------+---------------+
| TABLE_OWNER | TABLE_NAME | PARTITION_NAME | INSERTS | UPDATES | DELETES | TIMESTAMP  | TRUNCATED | DROP_SEGMENTS |
+-------------+------------+----------------+---------+---------+---------+------------+-----------+---------------+
| gtx         | gtx_tb1    | p0             |       1 |       0 |       0 | 2023-11-04 | NULL      |          NULL |
| gtx         | gtx_tb1    | p1             |       2 |       0 |       0 | 2023-11-04 | NULL      |          NULL |
| gtx         | gtx_tb1    | p2             |       2 |       0 |       0 | 2023-11-04 | NULL      |          NULL |
+-------------+------------+----------------+---------+---------+---------+------------+-----------+---------------+
```

字段说明:

INSERTS/UPDATES/DELETES：统计自上一次自动收集统计信息以来的 DML 执行次数，转储合并后显示的是 DML 修改的记录个数

TRUNCATED：指示上一次自动收集统计信息后该表、分区是否被 Truncate 过

DROP_SEGMENTS：自上次分析以来删除的分区和子分区段数

设置默认的收集策略

OceanBase 数据库提供了类似 Oracle 的 DBMS_STATS 系统包，用来收集、管理统计信息，以及配置统计信息的默认收集策略（Prefs）。DBMS_STATS 系统包在 MySQL 租户与 Oracle 租户均可使用.

查看 Prefs：

GET_PREFS：获取 Prefs 设置的 value

设置 Prefs：

SET_GLOBAL_PREFS：设置租户的 Prefs

SET_SCHEMA_PREFS：设置 schema 的 Prefs

SET_TABLE_PREFS：设置表的 Prefs

RESET_GLOBAL_PREF_DEFAULTS：还原租户的 Prefs。

删除 Prefs：

DELETE_SCHEMA_PREFS：删除 schema 的 Prefs

DELETE_TABLE_PREFS：删除表的 Prefs

策略名称	描述
APPROXIMATE_NDV	是否使用估算方式计算 NDV，默认为 True
CASCADE	是否同时收集表和索引的统计信息，默认为 True
DEGREE	设置统计信息收集任务的并行度，默认为 1
ESTIMATE_PERCENT	设置数据采样比例，默认使用自适应的比例
GRANULARITY	设置统计信息收集时的分区粒度，默认为 AUTO
INCREMENTAL	是否采用增量收集策略，默认为 False
METHOD_OPT	设置收集直方图信息的默认选项
STATS_RETENTION	设置历史统计信息的保留时间，默认 31 天

STALE_PERCENT	设置导致统计信息过期的数据变更百分比，默认 10%

设置统计信息的过期策略

本示例展示在 Oracle 租户中使用 DBMS_STATS 系统包用来设置租户与表的统计信息过期策略

查看过期策略（STATE_PERCENT）：

```
SELECT DBMS_STATS.GET_PREFS('STALE_PERCENT') FROM DUAL;
SELECT DBMS_STATS.GET_PREFS('STALE_PERCENT', 'TEST', 'T1') FROM DUAL;
```

设置过期策略（STATE_PERCENT）：

```
CALL DBMS_STATS.SET_GLOBAL_PREFS('STALE_PERCENT', '15');
CALL DBMS_STATS.SET_TABLE_PREFS('TEST','T1','STALE_PERCENT','20');
```

删除过期策略（STATE_PERCENT）：

```
CALL DBMS_STATS.DELETE_TABLE_PREFS('TEST', 'T1', 'STALE_PERCENT');
CALL DBMS_STATS.DELETE_SCHEMA_PREFS('TEST', 'STALE_PERCENT');
```

自动收集窗口

OceanBase 数据库预定义周一到周日 7 个自动统计信息收集的任务执行窗口

MAINTENANCE WINDOW：

维护窗口名字	开始时间/频率	最大收集时长
MONDAY_WINDOW	22:00/per week	4 hours
TUESDAY_WINDOW	22:00/per week	4 hours
WEDNESDAY_WINDOW	22:00/per week	4 hours
THURSDAY_WINDOW	22:00/per week	4 hours
FRIDAY_WINDOW	22:00/per week	4 hours
SATURDAY_WINDOW	6:00/per week	20 hours
SUNDAY_WINDOW	6:00/per week	20 hours

可以使用 DBA_SCHEDULER_JOBS 来查询 MAINTENANCE WINDOW 的信息

设置自动收集窗口

OceanBase 数据库基于 DBMS_SCHEDULER 系统包来管理 MAINTENANCE WINDOW，实现每

日的自动统计信息收集。

禁止/启用指定时间窗口的自动统计信息收集任务：

DBMS_SCHEDULER.DISABLE($window_name)

DBMS_SCHEDULER.ENABLE($window_name)

设置自动统计信息收集任务在指定时间窗口的开始执行时间

DBMS_SCHEDULER.SET_ATTRIBUTE($window_name, 'NEXT_DATE', $next_time);

设置自动统计信息收集任务在指定时间窗口的执行时长：

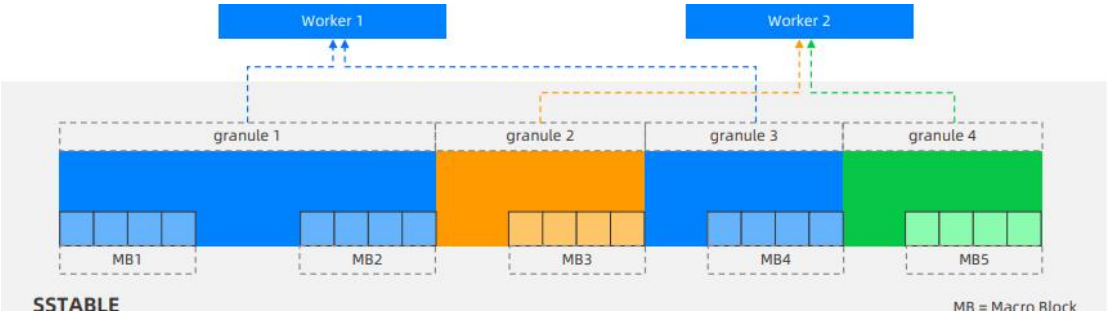
DBMS_SCHEDULER.SET_ATTRIBUTE($window_name, 'JOB_ACTION', $job($duration));

```
/* 禁用/开启周一自动收集统计信息 */
CALL DBMS_SCHEDULER.DISABLE('MONDAY_WINDOW');
CALL DBMS_SCHEDULER.ENABLE('MONDAY_WINDOW');
/* 设置周一自动收集统计信息开始的时间在晚上 8 点 */
CALL DBMS_SCHEDULER.SET_ATTRIBUTE('MONDAY_WINDOW', 'NEXT_DATE', '2022-09-12 20:00:00');
/* 设置周一自动收集统计信息的持续时长为 6 小时(21600000000 us) */
CALL DBMS_SCHEDULER.SET_ATTRIBUTE('MONDAY_WINDOW','JOB_ACTION','DBMS_STATS.GATHER_DATABASE_STATS_JOB_PROC(21600000000)');
```

开启并行计算

并行执行

并行执行是指将一个 SQL 查询任务分成多个子任务，并允许这些子任务在多个处理器上同时运行，以提高整个查询任务的执行效率。在现代计算机系统中，多核处理器、多线程和高速网络连接广泛应用，这使得并行执行成为了一种可行的高效率查询技术。



并行执行通过充分利用多个 CPU 和 IO 资源，达到降低 SQL 执行时间的目的

并行度（DOP）

并行度 DOP（Degree Of Parallelism）指定了并行执行的线程数量

OceanBase 多种方式对查询开启并行执行。在查询支持并行执行的前提下，多种并行开启方式优先级由高到低的顺序如下：

并行度的分配策略:

OceanBase 会将 DOP 按照分区个数的比例分配到多个 OBServer 节点上。例如，DOP 为 6，要访问 120 个分区，其中 server1 上有 60 个分区， server2 上有 40 个分区，server3 上有 20 个分区，那么会分 3 个线程给 server1，分 2 个线程给 server2，分 1 个线程给 server3，达到平均每个线程可以处理 20 个分区的效果

并行度的设置方法

在 OceanBase V4 版本， SQL 查询的并行度由两种方式来控制：手动并行和自动并行（Auto DOP）

手动 DOP 设置方法

在 OceanBase V4 版本中，我们可以通过以下方式来手动设置 SQL 语句或表扫描的并行度：

通过 PARALLEL Hint 指定 SQL 语句的并行度

```
SELECT /*+ parallel(3) */ * FROM t1, t2 WHERE t1.c2 = t2.c1; //设置 Query 的全局并行度
SELECT /*+ parallel(t1 3) */ * FROM t1, t2 WHERE t1.c2 = t2.c1; //设置 Query 的表级并行度
```

通过表属性 parallel 设置表扫描的默认并行度

```
ALTER TABLE tb1 PARALLEL 3;
ALTER INDEX tb1_ix1 PARALLEL 3; //设置索引扫描的并行度（Oracle 模式）
ALTER TABLE tb1 ALTER INDEX tb1_ix1 PARALLEL 3; //设置索引扫描的并行度（MySQL 模式）
```

通过会话的并行属性，设置会话中查询的默认并行度

```
#MySQL 模式
set _force_parallel_query_dop = n;
#Oracle 模式
ALTER SESSION FORCE PARALLEL QUERY PARALLEL n;
ALTER SESSION DISABLE PARALLEL QUERY;
```

注意：n>1 开启并行，n=1 关闭并行。

AUTO DOP 策略

Auto DOP 策略下，优化器如何判断是否使用并行执行，如何确定并行度以及每个查询的最大并行读，这些都需要用户进行相应的策略配置。

是否使用并行执行？

在执行计划选择时，OceanBase 数据库优化器会比较顺序执行与并行执行的代价，如果并行执行的耗时更低，才可能选择并行执行

系统变量 parallel_min_scan_time_threshold 控制基表扫描开启并行的最低执行时间

当优化器估算的基表扫描执行时间大于设定值时，会对基表扫描开启并行，并利用这个设定值计算一个合适的并行度

变量默认值设置为 1000 毫秒，即 1 秒

变量默认值设置为 1000 毫秒，即 1 秒

首先确定基表算子的 DOP。通过逐步增大 DOP 来评估并行执行的耗时，并考虑并行执行框架的开销，确定一个最合适的 DOP

其他算子，比如 Join 等，继承与之关联的算子的 DOP

如何限制最大的并行度？

系统变量 parallel_degree_limit 控制 Auto DOP 策略下所有查询语句的最大并行度

默认值为 0， 表示对自动获取的 DOP 不进行限制

AUTO DOP 的示例

本示例展示 Auto DOP 策略的设置和并行度的计算与选择.

设置会话级别的 Auto DOP 策略

```
SET parallel_degree_limit = 32; //* 设置 Auto DOP 策略的最大并行度
SET parallel_min_scan_time_threshold = 500; //* 设置 Auto DOP 策略选择并行执行的阈值
```

通过 EXPLAIN 查看查询的并行度

A、B 两表的关联查询不指定 Parallel Hint，Auto DOP 的结果如下：

A 表（tp1）的分区数为 4，基表扫描的 DOP 为 4

B 表（tp2）的分区数为 8，基表扫描的 DOP 为 8

Hash Join 的 DOP 继承自子算子中较大的并行度（DOP = 8）

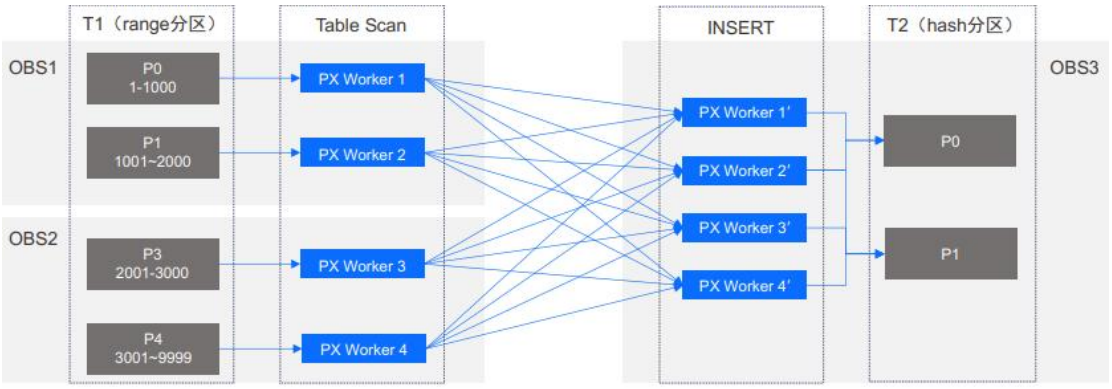
```
CREATE TABLE tp1(c1 INT, c2 INT, c3 INT) PARTITION BY HASH(c1) PARTITIONS 4;
CREATE TABLE tp2(c1 INT, c2 INT, c3 INT) PARTITION BY HASH(c1) PARTITIONS 8;

EXPLAIN SELECT * FROM tp1 a, tp2 b WHERE a.c1 = b.c1;
=======================================
|ID|OPERATOR                     |NAME    |
---------------------------------------
|0 |PX COORDINATOR               |        |
|1 | EXCHANGE OUT DISTR          |:EX10002|
|2 |  HASH JOIN                  |        |
|3 |   EXCHANGE IN DISTR         |        |
|4 |    EXCHANGE OUT DISTR (HASH)|:EX10000|
|5 |     PX BLOCK ITERATOR       |        |
|6 |      TABLE SCAN             |a       |
|7 |   EXCHANGE IN DISTR         |        |
|8 |    EXCHANGE OUT DISTR (HASH)|:EX10001|
|9 |     PX BLOCK ITERATOR       |        |
|10|      TABLE SCAN             |b       |
=======================================
Outputs & filters:
-------------------------------------
  1 - output(...), filter(nil), rowset=256, dop=8
  4 - output(...), filter(nil), rowset=256, dop=4
  8 - output(...), filter(nil), rowset=256, dop=8
```

并行 DML

并行 DML 通过使用并行执行机制来提高对大型数据库的表和索引执行插入、更新、删除等操作，以提高执行效率

OceanBase 数据库 V4 版本支持在 INSERT INTO SELECT、UPDATE、DELETE 等语句使用并行执行能力

开启并行 DML 时，查询部分总是自动开启并行，DML 的并行度和查询部分的并行度一致。读出的数据会根据待更新表的分区位置做重分布，然后由多个线程并行 DML，每个线程负责若干个分区



INSERT INTO T2 SELECT FROM T1

开启并行 DML

OceanBase 数据库支持在 SQL 语句或会话中显式启用并行 DML

通过 Hint 在 SQL 语句中启用或关闭并行 DML

```
INSERT /*+ ENABLE_PARALLEL_DML PARALLEL(4) */ INTO t2 SELECT * FROM T1;
INSERT /*+ DISABLE_PARALLEL_DML */ INTO t2 SELECT * FROM T1;
```

通过会话的并行属性，设置会话中的 DML 是否使用并行

```
#MySQL 模式
SET _force_parallel_dml_dop = n;
SET _force_parallel_dml_dop = 1;

#Oracle 模式
ALTER SESSION ENABLE PARALLEL DML;
ALTER SESSION FORCE PARALLEL DML PARALLEL n;
ALTER SESSION DISABLE PARALLEL DML;
```

注意：

Parallel DML 执行只有在 DOP>1 且 ENABLE PARALLEL DML 时才会生效

使用 Auto DOP 时自动开启 Parallel DML

并行 DML 的示例

如下示例为同时使用 ENABLE_PARALLEL_DML Hint 和 PARALLEL(n) 参数指定并行度 n，并且 n > 1，并行度 dop=2。

```
CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 INT) NOPARALLEL;
CREATE TABLE t2 (c1 INT PRIMARY KEY, c2 INT) PARALLEL 11 PARTITION BY HASH(c1) PARTITIONS 3;
CREATE TABLE t3 (c1 INT PRIMARY KEY, c2 INT) PARALLEL 10 PARTITION BY HASH(c1) PARTITIONS 4;
obclient> EXPLAIN INSERT /*+ ENABLE_PARALLEL_DML PARALLEL(2) */  INTO t1 SELECT * FROM T3;
+------------------------------------------------------------------------------------------------
| Query Plan
+------------------------------------------------------------------------------------------------
| =======================================================================
| |ID|OPERATOR                     |NAME           |EST.ROWS|EST.TIME(us)|
| -----------------------------------------------------------------------
| |0 |OPTIMIZER STATS MERGE        |               |1       |18          |
| |1 | PX COORDINATOR              |               |1       |18          |
| |2 |  EXCHANGE OUT DISTR         |:EX10001       |1       |18          |
| |3 |   INSERT                    |               |1       |17          |
| |4 |    EXCHANGE IN DISTR        |               |1       |4           |
| |5 |     EXCHANGE OUT DISTR (HASH)|:EX10000      |1       |4           |
| |6 |      OPTIMIZER STATS GATHER |               |1       |4           |
| |7 |       SUBPLAN SCAN          |ANONYMOUS_VIEW1|1       |4           |
| |8 |        PX BLOCK ITERATOR    |               |1       |4           |
| |9 |         TABLE SCAN          |t3             |1       |4           |
| =======================================================================
| Outputs & filters:
| -------------------------------------
|   0 - output(nil), filter(nil), rowset=256
|   1 - output([column_conv(INT,PS:(11,0),NOT NULL,ANONYMOUS_VIEW1.c1)], [column_conv(INT,PS:(11,0),NULL,ANONYMOUS_VIEW1.c2)]),
|   2 - output([column_conv(INT,PS:(11,0),NOT NULL,ANONYMOUS_VIEW1.c1)], [column_conv(INT,PS:(11,0),NULL,ANONYMOUS_VIEW1.c2)]),
|       dop=2
```

数据迁移集成技术

数据迁移集成概述

数据迁移集成的场景

数据迁移：将数据从一个系统或位置转移到另一个系统或位置

迁移后的数据在新的系统中是静态的，即数据的位置和结构已经改变，且数据本身不再与老系统进行同步

应用场景：适用于系统升级、更换、数据标准化、数据存储结构变化等情况。如系统升级时，新系统的表结构已经与老系统不同，但为了保障业务的连续性，需要将未完成生

命周期的数据一次性迁移到新系统

数据集成：将来自不同源的数据合并在一起，以便提供统一的数据视图和支持更复杂的业务流程

集成后的数据是动态的，可以在不同系统之间共享和更新，支持业务流程的自动化和数据的实时分析

应用场景：适用于需要跨系统或跨部门共享数据的情况，如多渠道数据整合、业务流程自动化、实时数据分析等。如实时数据分析场景下，需要将不同系统的数据实时流转到统一的数据湖，进行实时的业务分析，形成业务反馈闭环，加速业务的决策和改变

数据迁移集成的常见方式

通过数据迁移和集成技术，实现将数据从一个环境转移到另一个环境，或将来自不同来源的数据整合在一起

数据迁移按照迁移的不同需求可分为全量迁移、增量迁移、文件传输、自定义脚本等

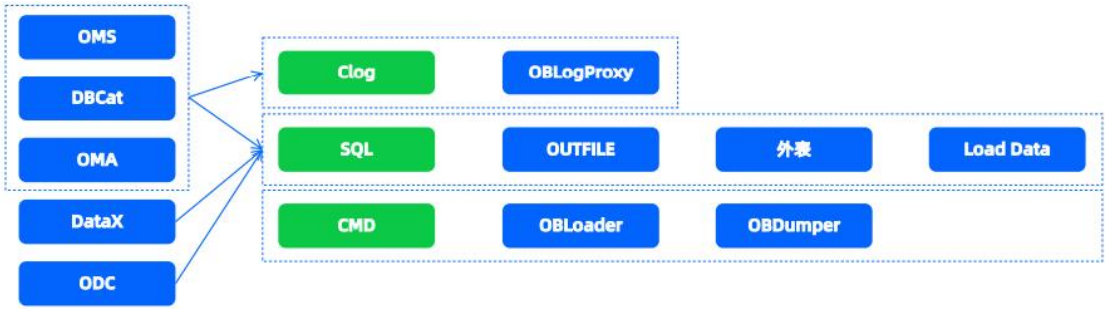
数据集成可分为：ETL、ELT、CDC、API 等

ETL (Extract, Transform, Load) 是一种数据集成过程，通常用于将数据从一个或多个源系统抽取出来，经过清洗、转换等处理后，加载到目标数据存储中

ELT (Extract, Load, Transform) 则是先将数据从源系统抽取出来，直接加载到目标系统中，然后再进行必要的转换操作

CDC (Change Data Capture)是一种用于监控和记录数据库表中数据变化的技术。它能够捕获对数据库表执行的 INSERT、UPDATE 和 DELETE 操作，并将这些变化以近乎实时的方式传输到其它系统

数据迁移集成的产品功能和工具使用



为支撑数据迁移和集成的需求，OB 目前提供了多种工具，包括：

SQL 和文件交互的工具

clog 改变数据捕获工具

命令行和文件交互的工具

DataX 数据集成工具

OMS 综合迁移集成工具

内核导入导出数据功能

外表

外表概述

外表的概念:通过 SQL 定义数据库外的文本文件的解析格式，实现 SQL 直接查询、统计分析文本文件的能力

使用场景

针对外表，可以查询、排序、聚合等读取操作

支持与其它表关联查询，包括外表和普通表

不支持针对外表创建索引或约束

不支持通过 SQL 修改外表的数据

可通过外表实现跨系统平台和异构数据源进行数据移动

通过外部表导入数据到实体表

创建外表

通过本地目录按照默认格式创建外表

前提条件: 为租户配置安全目录，安全目录用于控制导入或导出到文件时可以访问的路径

```
#在需要设置租户对应的 OBServer 节点上，通过 unix socket 建立 obclient 连接
obclient -S ObServer 安装目录下的 run/sql.sock -u 用户名@租户名 -p
```

```
#配置安全目录
obclient> SET GLOBAL secure_file_priv = "/home/test";
```

将/home/test/extDir/目录内的 CSV 文件创建外表 ext_t1。本地 location 需要配置安全目录，且 location 目录为安全目录的子目录,格式为 LOCATION = '[file://] local_file_path'。

```
create external table ext_t1(c1 int, c2 char(10) ) location ='file:///home/test/extDir/' format =(type ='CSV' );
```

location 也可设置为远程目录，支持 OSS、COS 两种，采用远程目录时，不需要配置安全目录

```
#远程文件格式为
{oss|cos}://$ACCESS_ID:$ACCESS_KEY@$HOST/remote_file_path
```

自定义列映射

col_name col_type [AS (metadata$filecol{N})] ： 用于定义列。其中， AS (metadata$filecol{N}) 用于手动定义列映射

将/home/test/extDir/目录内的 b.CSV 文件创建外表 ext_t1，只导入字段 2 和字段 3

```
cat /home/test/extDir/b.csv
1 5 apple
2 8 pear
3 10 banana
CREATE EXTERNAL TABLE ext_t1(
cnt int AS (metadata$filecol2), #映射第二列到 cnt 字段
name char(10) AS (metadata$filecol3) #映射第三列到 name 字段
)
LOCATION = 'file:///home/test/extDir/'
FORMAT = ( TYPE= 'CSV' )
PATTERN = 'b.csv'; #指定只映射 location 下的 b.csv 文件
```

自定义行列分割符，及包裹字符值符号

将/home/test/extDir/目录内的 a.CSV 文件创建外表 ext_t5，只导入字段 2 和字段 3

LINE_DELIMITER：指定文件的行分隔符。如果不指定，默认值为 LINE_DELIMITER='\n'

FIELD_DELIMITER：指定文件的列分隔符。如果不指定，默认值为 FIELD_DELIMITER='\t'

FIELD_OPTIONALLY_ENCLOSED_BY：指定文件中包裹字段值的符号，默认值为空

```
#cat /home/test/extDir/a.csv
1,5,"apple",
2,8,"pear",
3,10,"banana",
CREATE EXTERNAL TABLE ext_t5(
cnt int AS (metadata$filecol2),
name char(10) AS (metadata$filecol3)
) LOCATION = 'file:///home/test/extDir/'
FORMAT = (
TYPE = 'CSV'
LINE_DELIMITER = '\n' #行分割符
FIELD_DELIMITER = ',' #列分割符
FIELD_OPTIONALLY_ENCLOSED_BY ='"' #包裹字段值的符号
) PATTERN = 'a.csv';
```

自定义字符编码

将/home/test/extDir/目录内的 a.CSV 文件创建外表 ext_t6，只导入字段 2 和字段 3

ENCODING：指定文件的字符集编码格式，默认值为 UTF8MB4

```
cat /home/test/extDir/a.csv
1,5,“张三”,
2,8,“李四”,
3,10,“王五”,
CREATE EXTERNAL TABLE ext_t6(
cnt int AS (metadata$filecol2),
name char(10) AS (metadata$filecol3)
) LOCATION = 'file:///home/test/extDir/'
FORMAT = ( TYPE = 'CSV'
LINE_DELIMITER = '\n'
FIELD_DELIMITER = ','
FIELD_OPTIONALLY_ENCLOSED_BY ='"'
ENCODING = 'gb18030' #中文字符集
) PATTERN = 'a.csv';
```

创建报错示例

将/home/test/extDir/目录内的 a.CSV 文件创建外表 ext_t7，因有文件头未跳过，数据类型不一致导致无法查询

```
cat /home/test/extDir/a.csv
序号|数量|名称
1|5|张三
2|8|李四
3|10|王五
CREATE EXTERNAL TABLE ext_t7(
cnt int AS (metadata$filecol2),
name char(10) AS (metadata$filecol3)
) LOCATION = 'file:///home/test/extDir/'
FORMAT = (
TYPE = 'CSV'
FIELD_DELIMITER = '|'
ENCODING = 'gb18030'
) PATTERN = 'a.csv';

select * FROM ext_t7
ErrorCode = 1366, SQLState = HY000, Details =
Incorrect integer value
CREATE EXTERNAL TABLE ext_t7(
cnt int AS (metadata$filecol2),
name char(10) AS (metadata$filecol3)
) LOCATION = 'file:///home/test/extDir/'
FORMAT = (
TYPE = 'CSV'
```

```
FIELD_DELIMITER = '|'
ENCODING = 'gb18030'
SKIP_HEADER =1 #跳过第一行
) PATTERN = 'a.csv';
```

外表操作

用户租户可以通过 ALL_OB_EXTERNAL_TABLE_FILES 视图和 DBA_OB_EXTERNAL_TABLE_FILES 视图来查看外表信息

TABLE_NAME：外表的表名

TABLE_SCHEMA：外表所在的数据库名

PARTITION_NAME：外表的分区名

FILE_URL：外表所访问的文件的 URL

FILE_SIZE：外表所访问的文件的大小

刷新外表数据，SQL 语句：ALTER EXTERNAL TABLE table_name REFRESH

OUTFILE 导出数据

OUTFILE 导出数据

SELECT INTO OUTFILE 可通过 SQL 将查询结果集导出为文本文件

INTO OUTFILE file_route_option，导出文件路径（file_route_option）支持本地和远程（如：阿里云对象存储 OSS）

本地文件需要设置安全目标,且 ObServer 所在服务器的 admin 用户要有操作安全目录的权限

```
# 导出指定学号（12003）的选课情况（学号、姓名、课程号、成绩）
SELECT s.s_id,s.s_name,e.c_id,e.score
INTO OUTFILE '/home/test/extDir/12003.csv' //导出数据到安全目录/home/test/下的文件extDir/12003.csv
FROM students s , enrollment e
WHERE s.s_id=e.s_id AND s.s_id=12003;
# cat /home/test/extDir/12003.csv，导出文件的格式如下：
12003 张三 101 91.50
12003 张三 102 92.00
12003 张三 103 92.00
12003 张三 104 88.00
```

自定义导出文件格式，包括分隔符、换行符、字段包裹符

FIELDS|COLUMNS TERMINATED BY 指定列分割符

FIELDS|COLUMNS [OPTIONALLY] ENCLOSED BY 指定字段包裹符，如果使用了 OPTIONALLY 关键字，则仅对字符串类型的值使用指定字符包裹

LINES TERMINATED BY 指定换行符

```
# 导出语句
```

```
SELECT s.s_id,s.s_name,e.c_id,e.score
INTO OUTFILE '/home/test/extDir/12003.csv'
FIELDS TERMINATED BY '|' //以“|”作为列分割符号
OPTIONALLY ENCLOSED BY '"' //以“”作为包裹符符号
LINES TERMINATED BY '\n' //以“\n”作为换行符号
FROM students s , enrollment e
WHERE s.s_id=e.s_id AND s.s_id=12003;

# cat /home/test/extDir/12003.csv
12003|"张三"|101|91.50
12003|"张三"|102|92.00
12003|"张三"|103|92.00
12003|"张三"|104|88.00
```

自定义导出文件字符集

CHARSET | CHARACTER SET charset_name ， 默认为 UTF-8

```
# 导出语句
SELECT s.s_id,s.s_name,e.c_id,e.score
INTO OUTFILE '/home/test/extDir/12003.csv'
CHARSET gbk #设置字符集
FIELDS TERMINATED BY '|'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM students s , enrollment e
WHERE s.s_id=e.s_id AND s.s_id=12003;
```

外表与普通表关联查询结果集，也可通过 OUTFILE 导出数据

```
# 导出语句，将上一章节的关联查询结果集导出
SELECT a.name, a.cnt*b.price as total from ext_t3 a , t9 b
INTO OUTFILE '/home/test/extDir/ets.csv'
CHARSET gbk
FIELDS TERMINATED BY '|'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM ext_t3 a , t9 b
WHERE a.name=b.name;
```

LOAD DATA 导入数据

服务器文件导入

load data 可将外表文件加载到数据库表中

采用服务器端文件导入需要设置安全目录，并对导入用户设置文件操作权限

文件格式：INFILE 'file_name' ， 导入文件在 OBServer：/$PATH/$FILENAME

```
# 安全目录沿用之前的/home/test
```

```
# 对用户授权
GRANT FILE ON school_db.* TO user_name; //为 user_name 授权 school_db 库的所有表针对文件操作权限
# cat /home/test/courses.csv
101,数学,张老师
202,音乐,王老师
303,语文,杜老师
. . . . . .
# 导入数据
LOAD DATA
INFILE '/home/test/courses.csv' # 指定导入文件，需要 course 表所在 OBServer 进程运行用户 admin 有文件操作
权限
INTO TABLE courses
FIELDS TERMINATED BY ',' # 指定列分割符
LINES TERMINATED BY '\n' # 指行分割符
(C_ID,C_Name,Teacher); # 指映射列
```

本地&&远程文件导入

从客户端本地文件或者远程文件导入数据，[REMOTE_OSS | LOCAL] INFILE 'file_name'

导入文件在 OSS 上：oss://$PATH/$FILENAME/?host=$HOST&access_id=$ACCESS_ID&access_key=$ACCESSKEY

导入文件为本地文件：在 INFILE 前面加 Local

```
# cat /tmp/courses.csv
CS101,数学,张老师
CS202,音乐,王老师
CS303,语文,杜老师
. . . . . .
# 客户端本地文件加载，obclient 带--local-infile 启动，数据库版本为 4.2.2 及以上，且 oblient 版本为 V2.2.4 及以上。
LOAD DATA LOCAL INFILE '/tmp/courses.csv' INTO TABLE courses
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
(C_ID,C_Name,Teacher);
# 远程文件支持 OSS
LOAD DATA REMOTE_OSS INFILE 'OSS 地址' INTO TABLE courses
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
(C_ID,C_Name,Teacher);
```

并行导入数据

并行导入数据提升导入效率，/*+ parallel(N) */，N 默认为 4

```
# 默认并行度 4 导入
LOAD DATA /*+ PARALLEL */ #并行 4
INFILE '/home/test/courses.csv' INTO TABLE courses
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
(C_ID,C_Name,Teacher);
# OBServer 日志
[2024-09-15 13:19:49.751680] INFO [SQL.ENG] execute (ob_load_data_impl.cpp:1960)
[19261][T1002_L0_G0][T1002][YB42AC1C0FD2-000620F5868F6484-0-0] [lt=5] LOAD DATA start
report(file_path=/home/test/courses.csv, table_name=`school`.`courses`, batch_size=100,
parallel=4, load_mode=0, transaction_timeout=600000000)
# 并行度 10 导入
LOAD DATA /*+ PARALLEL(10) */ #并行 10
INFILE '/home/test/courses.csv' INTO TABLE courses
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
(C_ID,C_Name,Teacher);
# OBServer 日志
[2024-09-15 13:23:07.414730] INFO [SQL.ENG] execute (ob_load_data_impl.cpp:1960)
[19261][T1002_L0_G0][T1002][YB42AC1C0FD2-000620F5868F6490-0-0] [lt=8] LOAD DATA start
report(file_path=/home/test/courses.csv, table_name=`school`.`courses`, batch_size=100,
parallel=10, load_mode=0, transaction_timeout=600000000)
```

错误处理

load data 语句会在 OBServer 服务器下的 log 目录中生成 obloaddata.log.的日志文件，详细记录报错的情况

```
# cat /home/test/courses.csv
课程号,课程名,讲师
101,数学,张老师
202,音乐,王老师
303,语文,杜老师
# 导入数据
LOAD DATA INFILE '/home/test/courses.csv'
INTO TABLE courses
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
(C_ID,C_Name,Teacher);
# 错误信息
obclient [school]> LOAD DATA /*+ PARALLEL(10) */
INFILE '/home/test/courses.csv' INTO TABLE
courses FIELDS TERMINATED BY ',' LINES
```

```
TERMINATED BY '\n' (C_ID,C_Name,Teacher);
ERROR 1366 (HY000): Incorrect integer value

#obloaddata.log 日志
[admin log]$ cat obloaddata.log.yqKyA2
Tenant name: T1
File name: /home/test/courses.csv
Into table: `school`.`courses`
Parallel: 10
Batch size: 100
SQL trace: YB42AC1C0FD2-000620F5868EED8C-0-
0
Start time: 2024-09-03 10:57:48.024480
Load query:
.........
BatchId LineNum Type ErrCode ErrMsg
1 1 ERROR -4226 Incorrect
integer value
# 修正导入，增加跳过标题行，覆盖重复行
LOAD DATA INFILE '/home/test/courses.csv'
REPLACE
INTO TABLE courses
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 LINES #跳过标题行
(C_ID,C_Name,Teacher);
```

旁路导入

旁路导入概述

前面介绍的 load data 功能，在 SQL 执行时，需要将数据加载到内存，在转储或合并时将数据写入磁盘。在大规模数据加载到数据库时，这种机制过于低效，为此 OB 提供了旁路导入功能，绕过 SQL 层的接口，直接在 data 文件中直接分配空间并插入数据，从而提高数据导入的效率

旁路导入适用于大批量数据导入的场景，如数据迁移和同步、ETL 数据装载、文本文件导入数据等

支持的语法：

LOAD DATA /*+ direct(need_sort,max_error)|APPEND parallel(N) */ INFILE 'file_name' ...

APPEND 等同于使用的 direct(true, 0)，同时实现在线收集统计信息(GATHER_OPTIMIZER_STATISTICS Hint)的功能

INSERT /*+ append enable_parallel_dml parallel(N) */ INTO table_name select_sentence

OBLoader（后续章节介绍）

注意事项：

旁路导入会把所有的已有的数据都写一遍。如果原表的数据比较大，导入的数据比较少，可能不适合使用旁路导入

对于数据量较小的导入任务，不建议使用旁路导入

不能两个语句同时写一个表，因为导入的过程中会先加表锁

LOAD DATA 操作中采用了并行设计，每个子任务都作为一个独立的事务进行处理，并且执行顺序是随机的

旁路导入属于 DDL 语句，无法在多行事务（包含多个操作的事务）中执行,不能在 Begin 中执行,会自动提交（Autocommit 必须设置为 1）

旁路导入示例

```
#LOAD DATA 旁路导入
LOAD DATA /*+ PARALLEL(10) APPEND */
INFILE '/home/test/courses.csv'
INTO TABLE courses
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
(C_ID,C_Name,Teacher);
#Insert into 旁路导入
INSERT /*+ append enable_parallel_dml parallel(10)
*/ INTO courses
SELECT * FROM ext_courses
#旁路导入执行计划查看
EXPLAIN EXTENDED_NOADDR
INSERT /*+ append enable_parallel_dml parallel(10)
*/ INTO courses
SELECT * FROM ext_courses
```

使用 OMS 实现数据迁移和集成

DBCAT 评估和转换数据对象

DBCAT 概述

DBCAT 是一款轻量级的命令行工具，可用于提供数据库之间 DDL 转换（convert）和 Schema 比对（compare）等功能

DBCAT 需要安装 JDK 1.8 及以上版本，安装只需要解压 dbcat-{版本}-SNAPSHOT.tar.gz 包

Windows 可执行文件为 bin/dbcat.bat,Linux 可执行文件为 bin/dbcat

```
# dbcat 目录结构
bin：可执行文件目录。
conf： 日志文件配置目录。
lib：运行时期依赖的包。
output SQL 文件与报告文件，运行时生成。
# dbcat 命令查看帮助（如右图）
dbcat --help
# dbcat 命令查看版本
dbcat --version
```

```
# 查看转换和比对的帮助
dbcat help convert/compare
```

convert（结构转换）

dbcat convert 可将 TiDB、PG、SYBASE、ORACLE、DB2、MySQL 等与 OBOracle 或 OBMysql 之间进行对象转换，也可将 OB 对象转换到 MySQL 或 ORACLE

转换后的 SQL 文件输出到 dbcat 安装目录下的 output 目录

```
#mysql 到 obmysql 模式转换
./dbcat convert -H192.168.111.54 -P3306 -uroot -p'******' -D test --from mysql80 --to obmysql40 --all
#mysql 到 oboracle 模式转换
./dbcat convert -H192.168.111.54 -P3306 -uroot -p'******' -D test --from mysql80 --to oboracle40 --all
```

compare（结构对比）

dbcat 表结构比对，支持 MYSQL、ORACLE、SYBASE 与 OBMysql、OBOracle 进行表结构对比

dbcat 表结构比对需要设置配置文件，DBcat 安装目录下 conf/dbcat.properties

```
# 用法
./dbcat compare --config-file ../conf/dbcat.properties
# 修改 dbcat.properties 配置文件，设置源和目标的 JDBC 连接、用户、密码、版本等。
#比较结果在 output 下生成*verify.html
```

OMA 迁移评估

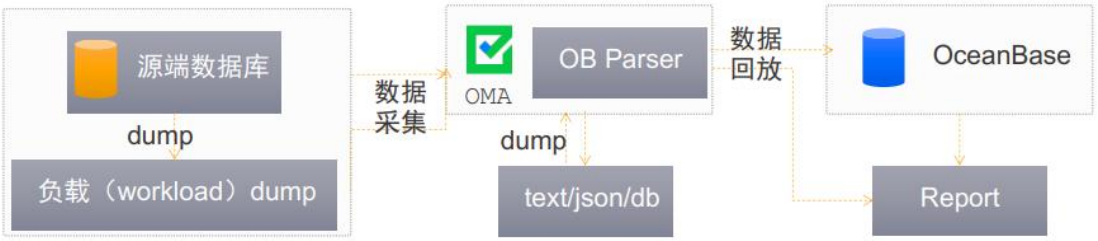
OMA 迁移评估概述

OceanBase 迁移评估（OceanBase Migration Assessment，OMA）是 OceanBase 提供的数据库迁移评估产品，支持对常见的关系型数据库 Oracle、MySQL、PostgreSQL、DB2 等迁移到 OceanBase 时，涉及的数据库对象和 SQL 语句提供兼容性评估、性能评估，并提供数据库画像分析和对象自动转换方案

OMA 是连接待评估的数据库和目标库（通常为 OceanBase 数据库）的中间工具。待评估数据库或负载信息采集对应的对象详情和 SQL 详情，内置的 OB Parser 对采集的对象和 SQL 进行兼容性分析，根据应用场景的不同，生成评估报告

OMA 4.×.×版本需要 Java1.8 版本的环境，运行的基本配置需要 8C16G 起，至少 2GB 的磁盘空间

OMA 运行只需要解压相应版本的 oma-x.x.x.tar.gz，执行其中的 start.sh 或 start.bat 脚本即可, --help 获取参数帮助信息



SQL 语句兼容性评估示例

SQL 语句兼容性评估支持数据来源为 Text、MyBatis、iBatis 格式

通过 TEXT 文件评估 SQL 语句的兼容性，需要使用 $$ 分隔各个 SQL

报告查看方式：默认为命令运行完成终端直接显示结果，也可通过 store-in-db 参数配置存储到 SQLite 库中，运行完成使用 OMA 根目录下的 reportTool 工具查看

```
#评估 SQL 命令
sh ./start.sh
--name test # 任务名称
--mode ANALYZE # 分析模式
--from-type TEXT # 数据来源 Text 文件
--evaluate-mode SOURCE_TARGET # 源和目标对比
--source-file "/home/test/a.sql" # SQL 路径
--source-db-type PostgreSQL # 源库类型
--source-db-version 10 # 源库版本
--target-db-type OBMYSQL # 目标 mysql 模式
--target-db-version 4.2.1.0 # 版本 4.2.1
--process-thread-count 5 # 并发度
```

MySQL 对象兼容性评估示例

OMA4.×.×支持支持 MySQL 5.6、5.7 和 8.0 版本

与 SQL 评估不一样，需要指定源数据库的 IP、端口、用户、密码、数据库

报告查看方式与 SQL 语句兼容性评估一致

```
# 评估对象命令
sh bin/start.sh
--name test
--mode ANALYZE
--from-type DB # 设置评估类型为 DB
--evaluate-mode SOURCE_TARGET
--source-db-type MySQL
--source-db-version 8.0
--source-db-host 192.168.111.54
--source-db-port 3306
--source-db-user "root"
--source-db-password "******"
```

```
--schemas "test"
--target-db-type OBMYSQL
--target-db-version 4.2.1.0
```

评估业务代码示例

OMA 评估是否支持业务代码中使用的 OCI 接口和 Pro*C 接口，同时支持评估 Java 代码

报告查看方式与 SQL 语句兼容性评估一致

```
# 评估命令
sh bin/start.sh
--name test
--mode ANALYZE
--from-type CODE # 数据来源代码
--evaluate-mode APPLICATION_CODE # 评估方式，应用代码
--source-file "/root/OBCP/com" # 代码所在目录
```

```
[INFO ] 13:57:59.160 [ScheduleTask] c.a.o.oma.scheduler.ReportUtil - 评估完成开始创建basic.html报告。
[INFO ] 13:57:59.162 [ScheduleTask] c.a.o.oma.reporter.html.HtmlReporter - create html report at /root/OBCP_V4_LAB_MIRROR/oma-3.4
0/report//test_20240904_135743-basic.html, current dir /root/OBCP_V4_LAB_MIRROR/oma-3.4.0
[INFO ] 13:57:59.199 [ScheduleTask] c.a.o.oma.scheduler.ReportUtil - basic.html报告: /root/OBCP_V4_LAB_MIRROR/oma-3.4.0/report//t
st_20240904_135743-basic.html
[INFO ] 13:57:59.200 [ScheduleTask] c.a.oceanbase.oma.scheduler.Analyze - all schema finished ...start Time : [ 2024.09.04 13:57
4 ] end Time : [ 2024.09.04 13:57:59 ] cost time :  15 Sec

评估程序OMA运行完成,评估报告简报:
任务 : test_20240904_135743 开始时间 : 2024.09.04 13:57:44 结束时间 : 2024.09.04 13:57:59

SCHEMA : DEFAULT 评估耗时 : 15021 毫秒
+-----------------------------------------------------------------------------+
| schema: schema      | source: sourceDB          | target: targetDB          |
+-----------------+---------+---------+---------+---------+---------+---------+
|  Object Type    |  pass   | convert | failure | skipped |  total  | percent |
+-----------------+---------+---------+---------+---------+---------+---------+
| JAVA            | 12      | 0       | 0       | 0       | 12      | 100.0%  |
+-----------------+---------+---------+---------+---------+---------+---------+
```

整库评估示例

为了满足用户一次性对一个数据库实例进行全面评估的需求，OMA 支持整库评估模式。可通过参数选定评估任务中需要的分析类型，其实现原理和对象评估、SQL 评估等一致

命令运行完成后，会在 report 目录下生成 PDF 格式的评估报告

```
# 评估命令,与对象评估类似
sh bin/start.sh
--name test
--mode ANALYZE_TOTAL #整库评估
--analyze-types OBJECT,SQL # 评估对象和 SQL
--from-type DB
--evaluate-mode SOURCE_TARGET
--source-db-type MySQL
--source-db-version 8.0
--source-db-host 192.168.111.54
--source-db-port 3306
--source-db-user "root"
```

```
--source-db-password "******"
--schemas "test"
--target-db-type OBMYSQL
--target-db-version 4.2.1.0
```

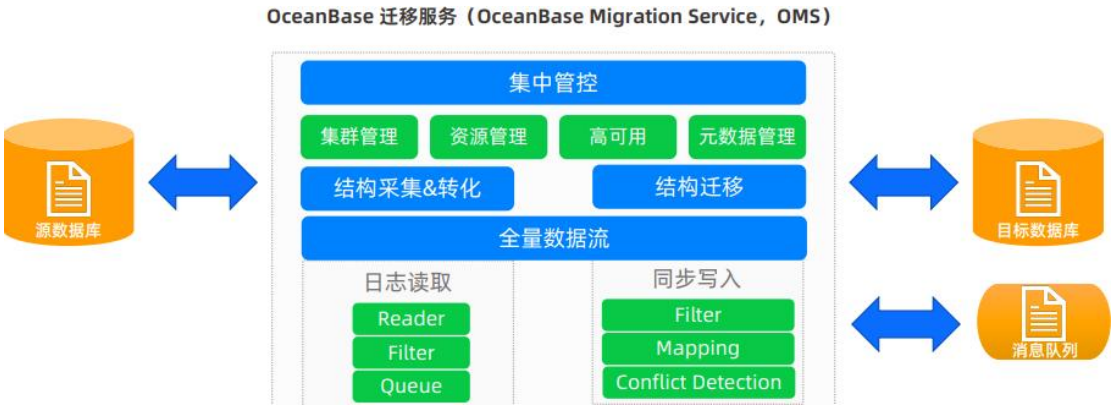
OMS 迁移和同步数据

OMS 概述

OceanBase 迁移服务（OceanBase Migration Service，OMS）是 OceanBase 数据库提供的一种支持同构或异构数据源与 OceanBase 数据库之间进行数据交互的服务，具备在线迁移存量数据和实时同步增量数据的能力

数据迁移适用于数据库升级、跨实例数据迁移、数据库拆分、扩容等业务场景

数据同步适用于数据异地多活、数据异地灾备、数据聚合和实时数据仓库等多种业务场景



OMS 组件功能

OMS 提供了白屏化的管理界面，从功能分层来看，在服务接入层包括传输项目的管理、各种类型数据源的管理、OMS 各个组件模块的运维和监控，以及告警设置等功能；在流程编排层包括实现上层表结构迁移、启动全量数据迁移/同步、增量数据迁移/同步、数据校检和数据订正，以及链路切换等任务的编排功能

OMS 的底层组件包括了负责结构迁移的组件 DBCat、增量拉取组件 Store、全量导入组件 Full-Import、增量同步组件 Incr-Sync、全量校验组件 Full-Verification、组件监控管理 Supervisor

OMS 关键概念

全量数据复制

将全量源端的存量数据全部迁移或同步到目标端

全量数据读取采用 JDBC 读取，根据源端的元信息，包括库、表、主键或非空唯一键、列等，选择合适的数据切片方法进行数据分片，分别读取数据

全量数据写入也是通过 JDBC 进行，支持按照最小切片点进行断点续传

数据校验

Full-Verification 组件负责从源端和目标端读取数据，并根据映射关系（依赖索引信息）对两端的数据进行全字段对比，对比结果包括差异数据文件和订正 SQL 文件

增量数据复制

通过 store 组件读取源端 CDC 数据同步到目标端， Store 组件是基于日志的 CDC，Store 由 Reader 和本地储存构成，不同的数据源对应不同的 Reader 组件，包括 Oracle Reader(基于 Oracle LogMiner 来获取增量变更日志)、MySQL Reader（基于 Binlog ）、OB Reader（基于 Clog）、DB2 Reader（基于 db2ReadLog API 读取日志记录）等

数据读取时，解析 store 组件传递的消息，将消息转换成框架适配的 RecordBatch

数据写入时，以主键/唯一键索引作为关键字，将 Incr-Sync 程序内存中的消息形成一张链式列表，保持原有事务的序

增量数据复制支持断点续传，依赖 Incr-Sync 维护框架内最小写入消息的 Checkpoint 并定期存储实现

OMS 部署结构

OMS 部署支持单节点部署、单地域多节点部署、多地域多节点部署三种方式。同时可将单节点扩展为多节点

OMS 部署涉及三个部件，OMS 安装包、MetaDB 集群(可以是 OBMysql 租户)、InfluxDB 时序数据库(可选，存储 OMS 历史监控数据)

部署方式支持通过 OAT 部署或者命令行部署

OMS 数据迁移同步限制

OMS 数据迁移支持实时迁移其它数据源的数据至 OceanBase 数据库，以及迁移 OceanBase 数据库的数据至其它数据源

对表是否有主键有一些限制，其中部分数据如 MySQL、Oracle 为源端时，支持无主键表，OMS 会在迁移过程中自动创建隐藏列及唯一索引；而对于 TiDB、PostgreSQL 、DB2 LUW 作为源端时，不支持无主键表

在少数场景下不支持增量 DDL，如 Oracle-> OB_MySQL 、 TiDB -> OB_MySQL 、PostgreSQL -> OB_MySQL、OB_Oracle -> MySQL 等

并不是所有类型的 DDL 都支持，支持的 DDL 范围，可在选择同步类型页面查看

OMS 数据同步支持 OceanBase 数据库的两种租户和 Kafka、RocketMQ、DataHub 之间进行实时数据同步，以及同步其它类型数据库的数据至 RocketMQ，并支持同步 ODP/IDB 逻辑表至 OceanBase 数据库 MySQL 租户的物理表和 DataHub

均不支持无主键表

目标端为 DataHub（Blob）、Kafka、OB_MySQL 时，大多场景均支持增量 DDL

并不是所有类型的 DDL 都支持同步，支持同步的 DDL 范围，可在选择同步类型页面查看

在 MySQL/Oracle 同步到 DataHub 时，不支持全量同步

OMS 迁移步骤

数据迁移项目需要以下 6 个步骤，包括建立源和目标端数据库的用户、建立数据源、建立数据迁移项目、预检查启动项目、查看运行情况、停止并释放项目

建立数据迁移项目，设置迁移类型时（如图）,可以选择全量迁移、增量同步、反向增量等，其中反向增量指源到目标迁移和同步完成后，可以实时回流业务切换后在目标端数据库产生的变更数据至源端数据库

以下示例为从 mysql 到 OB 的迁移项目，迁移内容包括：结构迁移、全量迁移、增量同步、全量校验、反向增量

预检查完成了连通性、时钟同步、数据库版本、增量日志等多个检查项目

结构迁移完成了从 mysql 到 OB 的对象语法的转换和创建

正向切换前，需要进行预检查、确认源端停写等多个步骤后，启动反向增量

OMS 同步步骤

数据同步项目需要以下 6 个步骤，包括建立源和目标端的用户、建立数据源、建立数据同步项目、预检查启动项目、查看运行情况、停止并释放项目

建立数据同步项目时，可以选择需要同步是的对象范围、设置同步的参数、全量和增量等。其中同步参数设置需要根据不同的数据源进行设置

以下示例为从 OB 到 Kafka 的同步项目，同步内容包括：结构迁移、全量迁移、增量同步

共涉及四个步骤，分别是结构同步、同步增量日志拉取、全量同步、增量同步

在全量同步页面，可以查看同步的状态、启动时间、结束时间、总计耗时和全量同步的进度，以及预估总行数、已完成同步行数、RPS 和实时流量等信息

在增量同步页面，可以查看同步的状态、延迟时间、当前位点时间、同步性能等信息

在同步项目页面点击 查看组件监控，可查看组件 Store、Incr-Sync、Full-Import 组件的详细信息

OMS 运维管理

权限管理

ROOT 角色的 root 用户由系统默认生成，不允许被删除。该用户等同于系统管理员，对整个系统具备读写操作的权限

ROOT-VIEWER 角色的用户由 root 用户创建，具备系统管理员级别的只读权限。

ADMIN 角色的 admin 用户由系统默认生成，不允许被删除

ADMIN-VIEWER 角色的用户具备部门管理员的只读权限

USER 角色的用户由 root 用户或 ADMIN 角色的用户创建，仅支持查看本 USER 创建的项目、数据源和运维任务

组件监控

OMS 运维监控模块中的概览页面，展示了机器的 CPU、内存、磁盘和总数，以及 Store、 Incr-Sync、Full-Import 和 FullVerification 组件的运行状态和数量等信息

在运维监控->组件->Store/Incr-Sync/Full-Import/Full-Verification，可查看对应组件的运行详情

运行日志查看

OMS 日志的默认目录为/home/admin/logs，包括了 Console 组件日志（ghana 目录）、CM 组件日志(cm 目录)，需要进入容器查看(docker exec -it <oms_container_id> bash)

/home/ds/目录下包括了 Store 、Full-Import、Incr-Sync、Full-Verification、Supervisor 组件日志

除了登录容器查看日志，也可从管理控制台 运维监控 模块查看相应组件日志

使用工具导入导出数据

OBLoader/OBDumper

OBLoader / OBDumper 概述

OBLoader 和 OBDumper 是 Java 语言编写的应用程序，用于通过命令行实现灵活的数据导入和导出功能，支持导入和导出 DDL、CSV、SQL 等多种文本文件的导入和导出

OBLoader 功能概述

支持的存储介质包括：本地磁盘、HDFS、对象存储（阿里云 OSS、亚马逊 S3、腾讯云 COS、华为云 OBS）

支持的导入数据文件格式包括：CSV、DDL、ORC、Cut、Parquet、Avro 等包括普通文本、SQL 以及大数据等格式文件

支持多线程、数据逻辑切分、直接路径等方式提升导入速率

支持自定义导入对象，包括：表、视图、库等

支持对导入数据的轻量级自定义加工

支持多种错误数据的处理机制

OBDumpe 功能概述

支持的存储介质包括：本地磁盘、对象存储（阿里云 OSS、亚马逊 S3、腾讯云 COS、华为云 OBS）

支持的导出数据文件格式包括：CSV、SQL、ORC、Cut、Parquet、Avro 等

支持自定义导出对象，包括：表、视图、库等

支持对导出数据预处理、过滤处理、自定义 SQL 等操作

支持从备副本、快照版本导出数据

OBLoader / OBDumper 运行启动

■ 运行环境要求

- 操作系统支持Linux/Macos/Windows7以上版本
- JDK需要1.8.0.3×××及以上

■ 工具下载地址：https://www.oceanbase.com/softwarecenter

- V4.0.0开始支持OceanBaseV4.0.0版本
- V4.2.1开始支持OceanBaseV4.1.0版本
- V4.2.6开始支持OceanBaseV4.2.1版本
- V4.3.0开始支持OceanBaseV4.3.0版本
- V4.3.1开支支持OceanBaseV4.2.1版本及以上版本的并行DDL

■ 工具的目录结构

```
#   Windows
 D:\ob-loader-dumper-4.3.1-RELEASE的目录

2024/08/12  10:11    <DIR>          .
2024/08/12  10:11    <DIR>          ..
2024/08/12  10:11    <DIR>          bin
2024/08/12  10:11    <DIR>          conf
2024/08/12  10:11    <DIR>          docs
2024/08/12  10:11    <DIR>          ext
2024/08/12  10:11    <DIR>          lib
2024/08/12  10:11             6,943 LICENSE
2024/08/12  10:11             4,712 NOTICE
2024/08/12  10:11    <DIR>          tools
```

■ 命令执行示例

```
#   Windows
~/bin/windows/obloader.bat -H
~/bin/windows/obdumper.bat -H

#  Linux
~/bin/obloader -H
~/bin/obdumper -H
```

OBLoader / OBDumper 参数格式

选项风格：

参数设计采用 Unix 和 GNU 两种风格方式。

Unix 风格的参数前加单破折线，选项为单字符，例如：ps -e；该风格的选项和参数之间可以不加空格，例如：-p******

GNU 风格的长参数前加双破折线，选项为单字符或者字符串，例如：ps --version；要求选项和参数之间有空格，例如：--table 'test'

选项分类：

命令行选项分为基础选项和高级选项。

基础选项：常用选项，包括连接选项（连接数据库模式）、功能选项（文件格式、数据库对象类型、存储路径）和其它选项

高级选项：包括功能选项（时间戳格式、表/列黑白名单筛选、错误处理）、性能选项和其它选项

在基础选项中有些选项是不可少的，称为必选选项，如连接选项、格式选项、数据库对象类型选项、存储路径选项

选项示例

连接选项

-h host_name, --host= host_name

用于连接 OceanBase 物理节点的主机地址

-P port_num, --port= port_num

用于连接 OceanBase 物理节点的主机端口

-u,--user

用于连接目标数据库的用户名、租户名和集群名。格式：

<user>@<tenant>#<cluster>，示例：-u

```
user@tenant#cluster
-D database_name, --database= database_name
该选项表示从指定的数据库中导出数据库对象定义和表数据
-p 'password', --password='password'
用于连接 OceanBase 数据库的账号密码
# 格式选项
--csv
用于标识导出 CSV 格式的数据文件
# 对象选项
--all
用于标识导出所有的数据库对象定义和表数据
-f 'file_path', --file-path= 'file_path'
用于指定数据文件存放在本地磁盘的绝对路径
```

OBLoader 简单示例

导入 DDL

```
# DDL mysql 模式, 4.0 以上可不指定--sys-user 和--sys-password, 导入目录/extDir 下的*-schema.sql 命名格式文件的 ddl。
ls /extDir/*
/extDir/school-schema.sql
./obloader -h172.28.15.207 -P2883 -uroot@T1#CPTest -p'******' --sys-user 'root' --sys-password '******' -D school --
ddl --all -f '/extDir'
```

导入 CSV

```
# CSV mysql 模式 导入学生和课程表
ls /extDir/*.csv
/extDir/courses.csv /extDir/students.csv
./obloader -h172.28.15.207 -P2883 -uroot@T1#CPTest -p'******' -D school --csv --table 'courses,students' -f
'/extDir'
```

导入 SQL

```
# SQL mysql 模式 导入 students 表 SQL 文件数据
ls /extDir/*.sql
/extDir/school-schema.sql /extDir/students.sql
./obloader -h172.28.15.207 -P2883 -uroot@T1#CPTest -p'******' -D school --sql --table '*' -f '/extDir'
```

OBLoader 存储路径

OBLOADER 支持从本地磁盘、HDFS、对象存储（阿里云 OSS、亚马逊 S3、腾讯云 COS、华为云 OBS）导入数据库对象定义和表数据。

语法格式：-f 'file_path', –file-path= 'file_path'

本地磁盘示例：指定从存储空间导入数据的资源路径，路径必须为绝对路径

```
# 再次导入 students 表 sql 格式数据，由于主键冲突，需要先删除之前导入数据（--truncate-table）
ls ~/../extDir
courses.csv logs school-schema.sql students.csv students.sql
./obloader -h172.28.15.207 -P2883 -uroot@T1#CPTest -p'******' -D school --sql --table 'students' --truncate-table
-f '~/../extDir'
# 因为相对路径报错如下
2024-08-29 21:30:50 [ERROR] Load failed! Error: Cannot create local tmp path for buffer data and meta files.
Reason: Cannot create directory '~/../extDir'.
```

OBLoader 自定义 CSV 格式

OBLoader 针对 CSV 格式数据文件的导入提供多种选项用于灵活配置文件格式，包括：

是否跳过文件头（--skip-header）
自定义字符串定界符（--column-delimiter 'column_delimiter_char’）
自定义行分割符（--line-separator 'line_separator_string’）
自定义列分割符（--column-separator 'column_separator_char’）
设置数据库连接串字符集（见下）
自定义需要替换为 NULL 的字符（--null-string 'null_replacer_string’）
自定义需要替换为空字符的字符（--empty-string'empty_replacer_string’）
自定义数据文件的后缀名（--file-suffix 'suffix_name’）
设置 CSV 文件的字符编码（见下）等
character-set && file-encoding 的区别
--character-set 用于指定数据库连接串字符集，与数据库编码一致
--file-encoding 用于指定 CSV 文件的字符编码，用于文件读取解析

```
# cat /home/test/courses.csv
课程号，课程名，讲师
CS101^“数学”^“张老师”
CS202^“音乐”^“王老师”
CS303^“语文”^“杜老师”
# csv 数据导入
./obloader -h172.28.15.207 -P2883 -
uroot@T1#CPTest -p'******' -D school --table
'courses' --csv --truncate-table -f
'/extDir'
--skip-header #跳过文件头
--column-delimiter '“' #字符串定界符
--line-separator '\n' #行分割符
--column-separator '^' #列分割符
```

```
--file-encoding 'gb2312’ #文件编码
```

OBLoader 自定义 SQL 格式

OBLoader 针对 SQL 格式数据文件的导入也提供了一些配置选项，包括：自定义行分割符（- - line - separator'line_separator_string'）、设置数据库连接串字符集(--character-set 'character_set_string')、自定义需要替换为 NULL 的字符(--null-string 'null_replacer_string')、自定义数据文件的后缀名(--file-suffix 'suffix_name')、设置 SQL 文件的字符编码(--fileencoding 'encode_name')等。

自定义 SQL 格式示例

```
# cat /extDir/courses.ss
insert into courses values('202','音乐','王老师');
insert into courses values('201','音乐','王老师');
insert into courses values('303','语文','杜老师');
# sql 数据导入
./obloader -h172.28.15.207 -P2883 -uroot@T1#CPTest -p'******' -D school --table 'courses' --sql
--truncate-table
-f '/extDir'
--line-separator '\n’ #行分割符
--file-suffix 'ss' #文件后缀
--file-encoding 'gb2312’ #文件字符编码
```

OBLoader 导入对象

OBLoader 针对 DDL 格式数据文件的配置选项包括：设置数据库连接串字符集(--character-set 'character_set_string')、自定义数据文件的后缀名(--file-suffix'suffix_name')、是否忽略对十六进制字符串进行解码(--ignore-unhex)、设置 CSV 文件的字符编码(--file-encoding 'encode_name’)等。

OBLoader 可通过参数设置指定要导入的数据库对象类型，包括：表(--table)、表组(--table-group)、视图(--view)、触发器(--trigger)、序列(--sequence)、同义词(--synonym)、自定义类型（--type）、包(--package)、包体(--package-body)、函数(--function)、存储过程(--procedure)等。

对象类型参数需和 ddl、csv、cut 等数据格式一起使用，实现对象定义和数据的导入

--all 参数指所有已支持的数据库对象，和其它对象选项互斥且 all 优先级更高

对象类型格式：--对象标识 '对象名 [, 对象名...]' | --对象标识 '*'

```
# cat /extDir/courses.ddl
drop table courses;
CREATE TABLE courses (
C_ID BIGINT NOT NULL,
C_Name VARCHAR(50) NOT NULL,
Teacher VARCHAR(50) NOT NULL,
PRIMARY KEY (C_ID)
);
```

```
# DDL 对象导入
./obloader -h172.28.15.207 -P2883 -
uroot@T1#CPTest -p'******' -D school --table
'courses' --ddl
--file-suffix 'ddl' #自定义后缀为 ddl
-f '/extDir'
```

OBLoader 旁路导入

旁路导入可以绕过 SQL 层的接口，直接在 data 文件中直接分配空间并插入数据，从而提高数据导入的效率

OBLoader V4.2.6 及之后的版本支持旁路导入数据

数据库版本要求 V4.2.0 及以上，如果通过 ODP 导入，需要数据库的版本为 V4.2.1 及以上，同时 ODP 也需要采用高版本

旁路导入需要使用 RPC 端口（OB 默认 2882、ODP 默认 2885）传输数据，并非 SQL 协议端口

基于表粒度进行整体提交，不再是会话级或事务级

参数

--direct 用于指定旁路导入模式

--parallel= parallel_num 用于旁路导入时加载数据的并行度

--rpc-port= rpc_port_num 用于连接 OBServer RPC 端口

```
# csv 数据导入课程表，旁路导入且 10 个并发
./obloader -h172.28.15.207 -P2883 -uroot@T1#CPTest -p'******' -D school --table 'courses'
--rpc-port 2885 #RPC 端口
--direct #旁路导入
--parallel 10 #并行导入
--csv --skip-header --column-delimiter '"' --line-separator '\n' --column-separator '^' --
fileencoding 'gb2312' --truncate-table -f '/extDir'
```

OBLoader 数据加工

OBLoader 支持针对数据文件进行自定义加工，可使用条件表达式和函数对要导入的数据进行简单的加工

对数据预处理需要定义控制文件，控制文件命名格式为<表名>.ctrl

ctl-path 控制文件参数用于指定控制文件的目录

操作步骤：

编辑控制文，映射和加工字段值

--ctl-path 'control_path' 指定控制文件存放在本地磁盘的绝对路径

```
# cat /extDir/pwdtest.csv
c1 ,c2,c3
key2 ,2,pwd2
key3 ,1,pwd678
key4 ,3,pwd234
```

```
# cat /extDir/pwdtest.ctrl
lang=java
(
c1 "trim(c1)" map(1), #去除空格
c2,
c3 "case when c2=1 then sm3_digest(c3,'BASE64') else sm3_digest(c3,'hex') end" #条件表达式
);
# csv 数据导入
./obloader -h172.28.15.207 -P2883 -uroot@T1#CPTest -p'******' -D test --table 'pwdtest' --csv -f
'/extDir' --ctlpath '/extDir/pwdtest.ctrl' --skip-header
```

OBLoader 错误处理

日志文件

log-path 指定运行日志的输出目录，默认情况下日志输出为-f 指定的导入文件所在的目录

.bad 文件存储错误数据，包括数据类型不匹配、非法的 NULL 值、数据类型溢出、预处理数据失败、列数不匹配等错误

.discard 文件存储重复数据，包括主键或者唯一键重复错误

错误处理参数设置

max-discards 参数指定每一张表的重复数据的最大值，默认值为-1，表示忽略重复数据继续导入

max-errors 参数指定每一张表导入时遇到错误数的最大值，任意一张表导入的错误数超过最大值，则停止导入该表的数据，默认值 1000

replace-data 参数标识替换表中重复的数据

```
# 重复导入 courses 数据，因主键冲突报错
./obloader -h172.28.15.207 -P2883 -uroot@T1#CPTest -p'******' -D school --table 'courses' --csv
--skip-header --
column-delimiter '"' --line-separator '\n' --column-separator '^' -f '/extDir'
# cat /extDir/logs/*error
2024-08-29 23:37:11 [ERROR] Error: Failed to load 3 records from "/extDir/courses.csv" into
table `school`.`courses`.
Check "ob-loader-dumper.bad" and "ob-loader-dumper.discard" for details
2024-08-29 23:37:11 [ERROR] System exit 1
# cat /extDir/logs/*warn
2024-08-29 23:37:10 [WARN] Retry Table: "courses", Partition: 0. Records: 3. Error: Duplicate
entry '101' for key
'PRIMARY'. Retry Mode: SERIAL.
# cat /extDir/logs/*discard
INSERT INTO `courses` (`C_ID`,`C_Name`,`Teacher`) VALUES ('101','"数学"','"张老师"');
```

OBLoader 性能优化

JVM 参数设置，JVM 内存配置默认为 4G，可根据实际进行配置

可直接编辑 OBLoader 安装目录下 bin/obloader 或 obloader.bat 文件

VM_OPTS="-Xms4G -Xmx4G" //默认配置 4G

JAVA_OPTS="$JAVA_OPTS -server $VM_OPTS -XX:MetaspaceSize=128M -XX:MaxMetaspaceSize=128M -Xss512K"

数据库内核参数设置

SQL 执行过程中的内存占用百分比，set global ob_sql_work_area_percentage=20; 默认值为 5，可设置范围[0,100]

服务端可接收的最大的网络数据包大小,set global max_allowed_packet=1073741824; 默认值 130023424

转储和合并相关参数，合并比较消耗性能，尽量不要触发。可开启内存的转储，并将转储的次数设置为 100 以上

OBLoader 参数

thread 线程数，用于标识并发线程数，默认值 CPU×2。需要关注 JVM 内存的设置，与线程数同步调整

rw 文件解析线程与总线程数的比例，默认值 1，文件解析线程数等于 thread 值×rw 值

batch 批量写入的事务大小，默认值 200。需要与行数据的大小成反比进行调整

block-size 文件切分的块大小，默认值 64MB

OBDumper 过滤对象

OBDumper 在格式选项、对象选项、路径选项、连接选项等基本与 OBLoader 一致，因此针对以上内容可参考 OBLoader

OBDumper 数据导出时经常需要过滤相关表、分区、列、行的数据

table 指定要导出的表

exclude-table 可指定要排除的表，表名支持模糊匹配

exclude-column-names 可指定要排除的列，不支持模糊匹配

include-column-names 可指定导出的列及顺序

partition 可指定要导出的分区

where 指定满足条件的数据

```
# 导出 school 库中 students 表中的 P3 分区，并设置查询条件。
./obdumper -h172.28.15.207 -P2883 -uroot@T1#CPTest -p'******' -D school --csv --table 'students' --exclude-columnnames 'sex' --partition 'p3' --where "sex = 'F'" -f /extDir
# tree /extDir
/extDir
├── data
│ └── school
│ └── TABLE
│ └── students.csv
```

OBDumper 自定义 SQL

尽管通过过滤处理可以灵活设置导出的结果数据，但有时不如直接采用 SQL 更加直观方便， 也不可实现表关联操作

OBDumper 可通过 query-sql 参数指定自定义 SQL 导出结果集数据

格式：--query-sql 'select_statement' 标识导出自定义查询语句的结果数据。该选项不与 --partition, --where 选项搭配使用

skip-check-dir 跳过检查目录是否为空，但是导出的文件可能会覆盖目录中同名文件的数据

```
# 指定 SQL 导出数据
./obdumper -h172.28.15.207 -P2883 -uroot@T1#CPTest -p'******' -D school --csv
--query-sql "select * from students where sex='F' " # 自定义 SQL
-f '/extDir/qq.csv' #指定导出文件名称
--skip-check-dir # 跳过目录不为空检查
```

OBDumper 数据加工

OBDumper 可通过预定义的控制文件对导出的数据进行预处理，可使用条件表达式和函数对要导出的数据进行简单的加工

对数据预处理需要定义控制文件，控制文件命名格式为<表名>.ctrl

ctl-path 控制文件参数用于指定控制文件的目录

操作步骤：

编辑控制文，映射和加工字段值

--ctl-path 'control_path' 指定控制文件存放在本地磁盘的绝对路径

```
# cat /extDir/courses.ctrl
lang=java
(
c_name "lower(c_name)",#转换小写
);
# 通过控制文件加工数据
./obdumper -h172.28.15.207 -P2883 -uroot@T1#CPTest -p'******' -D test --csv
--query-sql "select c_name,teacher from courses "
-f '/extDir/q1.csv'
--ctl-path '/extDir/courses.ctrl'
--skip-check-dir
```

OBDumper 快照版本读

OBDumper 支持设置 snapshot 参数导出快照版本数据。快照版本数据为最新合并后的数据

--snapshot 用于标识导出历史版本数据

```
# 查询 T1 租户合并状态
```

```
obclient [oceanbase]> select * from DBA_OB_MAJOR_COMPACTION \G
*************************** 1. row ***************************
FROZEN_SCN: 1724993279138960591
FROZEN_TIME: 2024-08-30 12:47:59.138961
GLOBAL_BROADCAST_SCN: 1724993279138960591
LAST_SCN: 1724993279138960591
LAST_FINISH_TIME: 2024-08-30 12:51:31.030541
START_TIME: 2024-08-30 12:47:59.207540
STATUS: IDLE
IS_ERROR: NO
IS_SUSPENDED: NO
# 插入新数据
insert into students values('12005','tom','F','22');
# 导出数据
./obdumper -h172.28.15.207 -P2883 -uroot@T1#CPTest -p'******' -D test --csv --table ‘students’ -f '/extDir' --
snapshot
# 查看导出数据，无新增的 12005 记录
cat /extDir/data/test/TABLE/students.csv
'S_ID','S_Name','Sex','Age'
12004,'larry','M',22
12003,'mary','F',21
```

OBDumper 副本弱读

OBDumper 支持设置 weak-read 参数从备副本导出数据，格式为--weak-read

OBDUMPER V3.1.0 之前版本仅与 ODP V3.1.2 及之后版本搭配使用

OBDUMPER V3.1.0 及之后版本需要修改 conf/session.config.json 文件中的参数 ob_route_policy=4{UNMERGE_FOLLOWER_FIRST}

```
# grep -i 'route' conf/session.config.json
"set session ob_route_policy=4"
# 弱读示例
./obdumper -h172.28.15.208 -P2883 -uroot@T1#CPTest -p'******' -D school --csv --table 'students' --partition='p0' -f
'/extDir' --weak-read
# 查看主备分布，数据读取的 OBServer 节点
SELECT object_name,tablet_id,svr_ip,subobject_name FROM DBA_OBJECTS obj, DBA_OB_TABLET_TO_LS ls,DBA_OB_LS_LOCATIONS
ll WHERE OBJECT_NAME like 'students%' AND ls.tablet_id=obj.data_object_id and ls.ls_id=ll.ls_id AND ll.role='leader'
and owner='school' and subobject_name='p0';
+-------------+-----------+---------------+----------------+
| object_name | tablet_id | svr_ip | subobject_name |
```

```
+-------------+-----------+---------------+----------------+
| students | 200089 | 172.28.15.207 | p0 |
+-------------+-----------+---------------+----------------+
SELECT svr_ip,client_ip,query_sql,sql_id,USER_CLIENT_IP FROM GV$OB_SQL_AUDIT where query_sql like
'%students%partition%' \G
svr_ip: 172.28.15.209
client_ip: 172.28.15.208
query_sql: SELECT `S_ID`,`S_Name`,`Sex`,`Age` FROM `students` partition(`p0`)
sql_id: DB8ED4678D47C70A20A4E407CE3FD60C
USER_CLIENT_IP: 172.28.15.205
```

OBDumper 性能优化

JVM 参数设置，JVM 内存配置默认为 4G，可根据实际进行配置

可直接编辑 OBDumper 安装目录下 bin/obdumper 或 obdumper.bat 文件

VM_OPTS="-Xms4G -Xmx4G" //默认配置 4G

JAVA_OPTS="$JAVA_OPTS -server $VM_OPTS -XX:MetaspaceSize=128M -XX:MaxMetaspaceSize=128M -Xss512K"

数据库内核调优

要求导出一致性数据时，建议在导出数据前，手动触发一次合并，在合并成功后再重新导出数据

OBDumper 参数

thread 线程数，用于标识并发线程数，默认值 CPU×2。需要关注 JVM 内存的设置，与线程数同步调整

page-size 任务分片的大小，默认值 1000000

block-size 标识文件块的切分阈值，默认值 1024MB

parallel-macro 一个线程可以处理的宏块数，默认值 8

fetch-size 只在 Oracle 模式下起作用，用于设置每次从数据库游标中读取的行数，默认值 1000

逻辑备份恢复

OBLoader&&OBDumper 可对数据库或者表进行逻辑备份和恢复

示例将 school 数据库的数据恢复到 test 库中

```
# 备份数据库
./obdumper -h172.28.15.207 -P2883 -uroot@T1#CPTest -
p'******' -D school --csv --ddl --all -f '/extDir'
#导入前
obclient [test]> show tables;
+----------------+
| Tables_in_test |
+----------------+
```

```
| pwdtest |
+----------------+
#导入数据库
./obloader -h172.28.15.207 -P2883 -uroot@T1#CPTest -
p'******' -D test --ddl --csv --all -f '/extDir'
#导入后
obclient [test]> show tables;
+----------------+
| Tables_in_test |
+----------------+
| classes |
| courses |
| pwdtest |
| students |
+----------------+



# 备份指定表数据库
./obdumper -h172.28.15.207 -P2883 -uroot@T1#CPTest -
p'******' -D school --csv --ddl --table 'classes'
-f '/extDir'
#导入前
obclient [test]> show tables;
+----------------+
| Tables_in_test |
+----------------+
| pwdtest |
+----------------+
#导入指定表
./obloader -h172.28.15.207 -P2883 -uroot@T1#CPTest -
p'******' -D test --ddl --csv --table 'classes'
-f '/extDir'
#导入后
obclient [test]> show tables;
+----------------+
| Tables_in_test |
+----------------+
| classes |
| pwdtest |
+----------------+
```

OBLoader / OBDumper 性能监测

JVM 内存监测
jstat -gcutil $pid $interval 重点观察 GC 的耗时、频率、回收效果
jmap -histo:live $pid 重点查看是否由大幅占用内存的对象
线程状态监测
top -Hp $pid 重点查看整体资源消耗最多的线程
jstack $pid 重点查看线程状态及运行线程调用堆栈耗时最多的方法
操作系统资源监测
pidstat -p $pid $interval 重点查看进程的资源消耗
sar -n DEV $interval 重点查看网络传输的效率
运行状态监控（运行日志）
Enqueue && Dequeue 监控 查看队列效率
内核监控 评估数据库资源消耗
进度监控 评估任务运行速度

DataX 移动数据

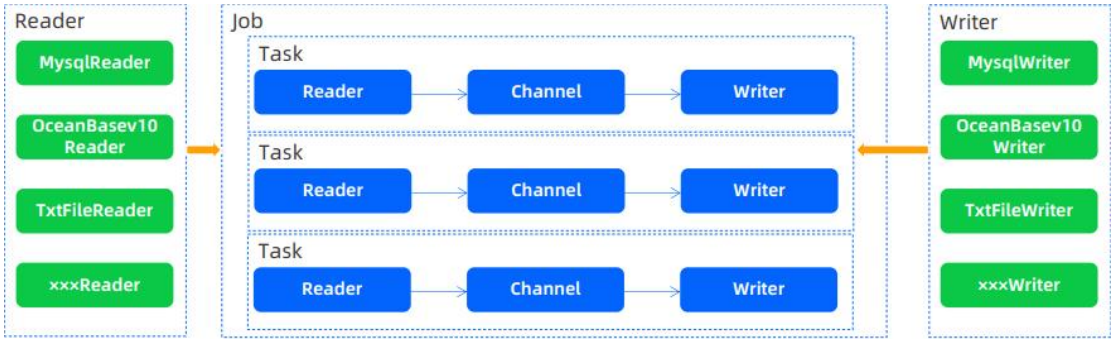
DataX 概述

DataX 是一个异构数据源离线同步工具，实现包括关系型数据库（MySQL、OceanBase、SQLServer 等）、NoSQL 数据存储（hive、MongoDB、HBase 等）、非结构化数据存储（TxtFile、Ftp、HDFS 等）等多种构数据源之间稳定高效的数据同步功能，具备细粒度的重试能力、详细的过程数据管理和展示、灵活的数据转换等功能

DataX 通过按照 Job 启动一个数据同步进程，按照数据源端切分策略拆分为多个 Task，Task 线程按照 Reader—>Channel—>Writer 过程完成所负责的数据处理。

Reader 和 Writer 以插件的方式加入 DataX 框架，不同的数据源插件开发完成后可和现有 DataX 生态中的任何数据源互通

Channel 用于对接 reader 和 writer，作为两者的数据传输通道，并处理缓冲、流控、并发、丰富数据转换等功能



DataX 运行启动

DataX 运行启动

DataX3.0 软件下载（https://datax-opensource.oss-cn-hangzhou.aliyuncs.com/202309/datax.tar.gz）

目录结构（如下图）

bin 目录下 datax.py 为 DataX 新任务的启动脚本

conf 目录包含核心参数的配置文件，如 taskGroup 的数量、JVM 参数等

job 目录放置了一个测试 job 的配置文件

log 目录用于存放 DataX 的运行日志

plugin 目录放置各种 reader 和 writer 的插件，图中为 OceanBase 数据库相关插件

```
${DATAX_HOME}
├── bin
│   ├── datax.py
│   ├── dxprof.py
│   └── perftrace.py
├── conf
│   ├── core.json
│   └── logback.xml
├── job
│   └── job.json
├── lib
│   ├── datax-common-0.0.1-SNAPSHOT.jar
│   ├── datax-core-0.0.1-SNAPSHOT.jar
├── log
├── plugin
│   ├── reader
│   │   ├── oceanbasev10reader
│   │   │   ├── libs
│   │   │   ├── oceanbasev10reader-0.0.1-
SNAPSHOT.jar
│   │   │   ├── plugin_job_template.json
│   │   │   └── plugin.json
│   └── writer
│       ├── oceanbasev10writer
│       │   ├── libs
│       │   ├── oceanbasev10writer-0.0.1-
SNAPSHOT.jar
│       │   ├── plugin_job_template.json
│       │   └── plugin.json
├── script
└── tmp
```

DataX 示例

DataX 的 job 配置文件格式为 JSON 格式

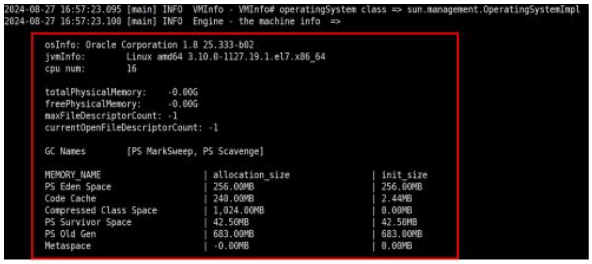
setting 部分为配置参数设置，如下图：speed.byte 用于设置 channel 的速度， 也可设置为 record 记录数或者 channel 并发数。errorLimit 参数为报错记录数的容忍度，可以是记录数 record 或者百分比 percentage

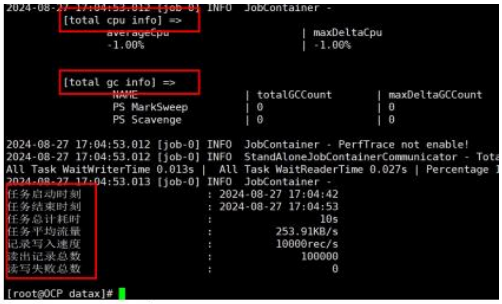
content 部分设置 reader 和 writer 插件的相关参数

```
{
    "job": {
        "setting": {
            "speed": {
                "byte":10485760
            },
            "errorLimit": {
                "record": 0,
                "percentage": 0.02
            }
        },
        "content": [
            {
                "reader": {
                    "name": "streamreader",
                    "parameter": {
                        "column" : [
                            {
                                "value": "DataX",
                                "type": "string"
                            },
                            * * * * * *
                        ],
                        "sliceRecordCount": 100000
                    }
                },
                "writer": {
                    "name": "streamwriter",
                    "parameter": {
                        "print": false,
                        "encoding": "UTF-8"
                    }
                }
            }
        ]
    }
}
```

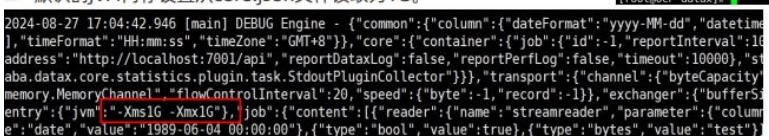

运行示例 Job：python ./bin/datax.py ./job/job.json --loglevel=debug

默认在控制台和日志 log 目录有 CPU、JVM、GC、任务统计等相关信息输出





■ 默认的JVM内存设置从core.json文件读取为1G。



DataX OceanBase 插件

OceanBasev10Reader

OceanBasev10Reader 通过 Java 客户端（底层 MySQL JDBC 或 OceanBase Client）以 obproxy 远程连接 OceanBase 数据库，并根据用户配置的信息生成查询语句，然后发送到远程 OceanBase 数据库，远程的 OceanBase 数据库将该 SQL 执行的返回结果使用 DataX 自定义的数据类型拼装为抽象的数据集，再传递给下游 Writer 处理

OceanBasev10writer

oceanbasev10writer 插件实现了将数据写入到 OceanBase 数据库的目标表中的功能。在底层实现上，oceanbasev10writer 通过 Java 客户端（底层 MySQL JDBC 或 OceanBase Client）以 obproxy 远程连接 OceanBase 数据库，并执行相应的 insert sql 语句来将数据写入到 OceanBase 数据库， OceanBase 数据库内部会分批次提交入库

实现原理方面，oceanbasev10writer 通过 DataX 框架获取 Reader 生成的协议数据，生成 insert 语句。在写入数据时，如果出现主键或唯一键冲突，OceanBase 数据库的 MySQL 租户可以通过 replace 模式来更新表中的所有字段；OceanBase 数据库的 Oracle 租户当前只能使用 Insert 方式。出于性能考虑，写入采用 batch 方式批量写，当行数累计到预定阈值时，才发起写入请求

示例OBMysql到OBOracle（一）

```
{
    "job": {
        "setting": {
            "speed": {
                "channel": 4      #并发通道数
            },
            "errorLimit": {
                "record": 0,
                "percentage": 0.1
            }
        },
        "content": [
            {
                "reader": {
                    "name": "oceanbasev10reader",
                    "parameter": {
                        "splitPk":  "p_id",    #并发读取分割主键，最好是数字型
                        "username": "root@T1#CPTest",
                        "password": "******",
                        "column": ["Plocal","y2024m7","
.....
```

```
            "writer": {
                "name": "oceanbasev10writer",
                "parameter": {
                    "batchsize":"200",        #批量提交数
                    "writerThreadCount":"20",     #并发写线程数
                    "obWriteMode": "insert",
                    "column": ["*"],
                    "preSql": ["truncate table ucpi2"],
                    "connection": [
                        {
                            "jdbcUrl":
"jdbc:oceanbase://172.28.15.208:2883/sys",
                            "table": ["ucpi2"]
                        }
                    ],
                    "username": "sys@T2#CPTest2",
                    "password":"*******",
                    "memstoreThreshold": "0.9"     #memstore阈值
                }
            }
        }
    ]
  }
}
```

ODC 导入导出数据

ODC 导入导出数据概述

功能：包括导入结构和数据、仅导入数据、仅导入结构

V4.2.3 及之后版本支持导入 /导出 MySQL 类型数据库结构和数据

V4.2.4 及之后版本支持导入/导出 Oracle 类型的数据库结构和数据

ODC 导入工具支持导入 ZIP 压缩文件（批量导入和单表导入）、SQL 文件（批量导入）和 CSV 文件（单表导入）

注意事项:

Web 版 ODC 对文件大小有限制，最大支持 2GB（压缩后）的文件

最大支持 5 个任务并行运行，后续任务在队列中等待运行

任务涉及的文件默认保留 14 天

在 ODC V4.1.0 之后的版本，针对 OceanBase 数据源，配置 sys 租户账号可以提升导入速度

操作流程

第一步：创建导入/导出任务（工单->导入/导出->新建导入/导出）

第二步：查看和操作导入/导出任务（工单->导入/导出->选择对应任务），可查看任务信息、任务流程、任务日志

ODC 造数示例

创建模拟数据任务（工单->模拟数据->新建模拟数据）

查看和操作任务（工单->模拟数据->选择对应任务）

总结

工具	结构迁移	全量复制	增量复制	文件导出/导入	支持的数据格式
OUTFILE语句	-	支持	不支持	导出	数据格式：CSV，只针对OceanBase
LOAD DATA	-	支持	不支持	导入	数据格式：CSV，只针对OceanBase
外部表	-	支持	不支持	导入	数据格式：CSV，只针对OceanBase
DBCAT	支持	不支持	不支持	支持结构转换后导出	数据源：TiDB、PG、SYBASE、ORACLE、DB2、MySQL等 输出文件为：sql文本文件
OMS	支持	支持	支持	不支持	迁移：支持OceanBase、MySQL、Oracle、PostgreSQL DB2_LUW等 同步：支持OceanBase到Kafka、RocketMQ、Datahub等的同步
OBLoader/OBDumper	支持	支持	不支持	支持	只针对OceanBase
DataX	不支持	支持	不支持	支持（可输出为文本）	关系型数据库（MySQL、OceanBase、SQLServer等）、NoSQL数据存储（hive、MongoDB、HBase等）、非结构化数据存储（TxtFile、Ftp、HDFS等）等