OCEANBASE | 海量记录 笔笔算数

数据库管理与运维
OceanBase
从入门到实践

OceanBase 研发团队

联系我们

欢迎 OceanBase 爱好者、用户和客户联系我们反馈问题：

- 社区版官网论坛：https://ask.oceanbase.com/
- 社区版项目网站提 Issue
  - MiniOB GitHub：https://github.com/oceanbase/miniob/issues
  - OceanBase GitHub：https://github.com/oceanbase/oceanbase/issues
  - OceanBase Gitee：https://gitee.com/oceanbase/oceanbase/issues
- 技术交流及关注动态：群号 33254054

钉钉技术交流群
（群号：33254054）

微信 OB 小助手

目 录

第 1 章：OceanBase 数据库概述

OceanBase 数据库是一个原生的分布式关系数据库，它是完全由阿里巴巴和蚂蚁集团自主研发的项目。OceanBase 数据库构建在通用服务器集群上，基于 Paxos 协议和分布式架构，提供金融级高可用和线性伸缩能力，不依赖特定硬件架构，具备高可用、线性扩展、高性能、低成本等核心技术优势。

OceanBase 数据库具有如下特点：

- 高可用

  单服务器故障能够自愈，支持跨城多机房容灾，数据零丢失，可满足金融行业 6 级容灾标准（RPO=0，RTO<=30 秒）。

- 线性扩展

  透明扩展，自动负载均衡，应用透明的水平扩展，集群规模可超过 1500 节点，数据量可达 PB 级，单表记录万亿行。

- MySQL/Oracle 高度兼容

  社区版兼容 MySQL 协议、语法和使用习惯，MySQL 客户端工具可以直接访问 OceanBase 数据库。

  企业版兼容 MySQL、Oracle 协议，需要使用 OceanBase 自己的驱动才可以访问 OceanBase 数据库的 Oracle 租户。

  > **说明**
  > MySQL 从 5.6 开始兼容，Oracle 从 Oracle 11g 开始兼容。

- 高性能

  准内存级数据变更操作、独创的编码压缩技术，结合线性水平扩展，TPC-C 测试达到 7.07 亿 tpmC。

- 低成本

  使用 PC 服务器和低端 SSD，高存储压缩率降低存储成本，高性能降低计算成本，多租户充分利用系统资源。

- 多租户

  原生支持多租户架构，同一套数据库集群可以为多个独立业务提供服务，租户间数据隔离，降低部署和运维成本。

OceanBase 数据库支持支付宝的全部核心业务，以及银行、保险、证券、运营商等多个行业的数百个客户的核心业务系统。

OceanBase 发展历史

在使用 OceanBase 之前，我们先对 OceanBase 的历史做一个简单的了解。

- 诞生

  2010 年，OceanBase 创始人阳振坤博士带领初创团队启动了 OceanBase 项目，第一个应用是淘宝的收藏夹业务。如今收藏夹依然是 OceanBase 的客户，收藏夹单表数据量非常大，OceanBase 用独创的方法解决了其高并发的大表连接小表的需求。

- 关系数据库

  早期的版本中，应用通过定制的 API 库访问 OceanBase。2012 年，OceanBase 发布了支持 SQL 的版本，初步成为一个功能完整的通用关系数据库。

- 初试金融业务

  OceanBase 进入支付宝（后来的蚂蚁集团），开始应用于金融级的业务场景。2014 年“双11”大促活动，OceanBase 开始承担交易库部分流量。此后，新成立的网商银行把所有核心交易库都运行在 OceanBase 上。

- 金融级核心库

  2016 年，OceanBase 发布了架构重新设计后的 1.0 版本，支持分布式事务，提升了高并发写业务中的扩展，同时实现多租户架构，这个整体架构延续至今。同时，到 2016 年“双11”时，支付宝全部核心库的业务流量都运行在 OceanBase 上，包括交易、支付、会员和最重要的账务库。

- 走向外部市场

  2017 年，OceanBase 开始试点外部业务，成功应用于南京银行。

- 商业化加速

  2018 年，OceanBase 发布 2.0 版本，开始支持 Oracle 兼容模式。这一特性降低应用改造适配成本，在外部客户中快速推广开来。

- 登峰造极

  2019 年，OceanBase 2.2 版本参加代表 OLTP 数据库最权威的 TPC-C 评测，以 6000 万 tpmC 的成绩登顶

世界第一。随后，在 2020 年，又以 7 亿 tpmC 刷新纪录，截止目前依然稳居第一，这充分证明了 OceanBase 优秀的扩展性和稳定性。OceanBase 是第一个也是截止目前唯一一个上榜 TPC-C 的中国数据库产品。

- HTAP 混合负载

  2021 年，OceanBase 3.0 基于全新的向量化执行引擎，在 TPC-H 30000GB 的评测中以 1526 万 QphH 的成绩刷新了评测榜单。这标志着 OceanBase 一套引擎处理 AP 和 TP 混合负载的能力取得了基础性的突破。

- 开源开放

  2021 年 6 月 1 日，OceanBase 宣布开源，开放合作，共建生态。OceanBase 在 2021 年 6 月正式推出社区版并开放源码，版本从 3.1.0 开始，源码托管地址：https://github.com/oceanbase/oceanbase。同时代码也同步发布到开源中国网站：https://gitee.com/oceanbase/oceanbase。开源的内容包括：

  - 数据库内核 OceanBase
  - 反向访问代理 `obproxy`
  - 数据库客户端命令行工具 `obclient`
  - 自动化部署工具 `OBD`
  - C 语言驱动 `obconnector-c`
  - CDC 组件代理 `oblogproxy` 和 `canal` 插件
  - OceanBase 监控客户端组件 `obagent`
  - spark 插件 `obspark`（待开源）

OceanBase 业务案例

与其他开源数据库不同， OceanBase 先有企业版后有社区版、先有大企业商业版案例再有社区版案例。但社区版和企业版的核心能力一样。

典型客户如下:

- 自用：蚂蚁集团（包括支付宝、网商银行）。
- 银行：中国工商银行、南京银行、东莞银行、天津银行、苏州银行、常熟农商银行。
- 保险：中国人民保险、中华保险。
- 证券：招商证券、上投摩根。
- 非金融行业：浙江移动、山东移动、数字江西、中国石化。

更多客户案例，请参考：https://www.oceanbase.com/customer/home 。

OceanBase 是单进程软件，独立部署，跟硬件、云平台没有绑定关系。可以部署在各个云厂商的云服务器上。OceanBase 在阿里云也有公有云数据库服务（https://www.aliyun.com/product/oceanbase）。

OceanBase 在公有云上（包括在 ECS 上）独立部署的客户案例有:

- 中华联合财险
- 菲律宾版支付 GCash
- 印度尼西亚电子钱包 DANA

OceanBase 社区版简介

OceanBase 数据库社区版使用 MulanPubL - 2.0 许可证，您可以免费复制及使用源代码。当您修改或分发源代码时，请遵守木兰协议。 OceanBase 社区版官方网站地址是： https://open.oceanbase.com。

下载方法

- 官网下载:

  https://open.oceanbase.com/softwareCenter/community

- GitHub 下载:

  https://github.com/oceanbase/oceanbase/releases/

- 阿里云 Yum 源:

  https://mirrors.aliyun.com/oceanbase/OceanBase.repo

支持的操作系统

OceanBase 社区版支持的操作系统包括:

- CentOS：推荐 7.2 以后版本。
- Debian：推荐 9.8、10.9 版本。
- openSUSE：推荐 15.2 版本。
- OpenAnolis：推荐 8.2 版本。
- SUSE： 推荐 15.2 版本。
- Ubuntu：推荐 16.04、18.04、20.04 等版本。

与 MySQL 数据库的不同

OceanBase 社区版兼容 MySQL 语法功能（主要是 5.6 的绝大部分语法，部分 8.0 的新特性等），底层原理跟 MySQL 完全没有关系，不依赖开源 MySQL 组件，没有 InnoDB 引擎等。 OceanBase 自身的存储引擎与 MySQL 的存储引擎相比，空间压缩效果更明显，社区版的压缩效果可以做到 MySQL 空间的四分之一。

OceanBase 是分布式数据库集群产品，生产环境默认三副本，并且三副本之间的同步协议不是异步同步或半同

步，而是使用 Paxos 协议同步事务日志。OceanBase 集群可以跨机房跨城市部署，机器或者机房故障时，集群内部多副本自动切换，不丢数据。因此 OceanBase 天然适合两地三中心异地容灾和多活建设。

OceanBase 集群支持多租户（也叫多实例），所有的租户按需分配，弹性伸缩，具备高可用能力，类似于云数据库服务。运维人员只需要维护少数几套集群，就可以提供很多实例给业务使用，易用性非常好。

OceanBase 支持水平拆分技术，具体体现为分区表，不需要分库分表，SQL 和事务对业务完全透明，功能上没有限制。分区表线性扩展性较好，目前已知案例最大单租户节点规模是 1500 台。

OceanBase 的 SQL 引擎能力远比 MySQL 功能强大，支持 SQL 解析和执行计划缓存，支持复杂的 SQL 运算，支持大纲技术干预 SQL 执行计划等。同时一套 SQL 引擎一个数据源，同时支持 OLTP 和 OLAP 类型的混合场景需求，即通常说的 HTAP 能力。

社区版核心功能

OceanBase 社区版包含 OceanBase 企业版的所有核心功能，如下:

- 多副本高可用、强同步能力。
- 多租户能力。
- 在线弹性伸缩能力。
- 异地容灾/多活能力（包括两地三中心、三地五中心等）。
- 分区表、复制表等分布式能力。
- HTAP 能力。
- MySQL 兼容性。
- 备份恢复能力。
- CDC 能力。

OceanBase 社区版跟企业版的差异在于企业版会包含更多高级功能，如商业特性兼容、图形化管理工具、操作审计、安全加密、高可用扩展等。有关企业版信息请查看企业版官方网站（https://www.oceanbase.com/）。

适合社区版的业务场景

- MySQL 5.6/5.7 实例规模很大的场景。

  MySQL 实例规模大，需要自动化运维平台。自动化运维平台在处理 MySQL 异常宕机切换和主备不一致问题时很可能需要 DBA 介入。高可用和强一致问题是 MySQL 最大的风险， OceanBase 的多租户、高可用和强一致能力可以彻底解决这个痛点。

- MySQL 5.6/5.7 数据量非常大、存储成本高的场景。

  MySQL 业务数据量增长到几 T 以上时，查询和读写性能可能会下降，大表 DDL 时间变长，风险增加。单机

磁盘容量可能到达扩容瓶颈。

OceanBase MySQL 租户的在线 DDL，数据存储高压缩比可以解决这些痛点。

- 业务访问压力大或者变化大的场景。

  业务访问压力大，基于 MySQL 改造的分布式数据库中间件产品能分担一定程度的业务压力和存储空间压力，但是缺乏跨节点的强一致性查询，以及需要分布式事务中间件协调事务，扩容的时候可能需要数据逻辑拆分（俗称拆库拆表），运维成本高，风险高。

  OceanBase MySQL 租户提供分区表的水平拆分方案，提供原生的 SQL 和事务能力，对业务透明。并且支持在线扩容和缩容，内部数据迁移异步进行，具备高可用能力，不怕扩容和缩容过程中出现故障，可以解决上面这些痛点。

- 交易数据库上的复杂查询场景。

  交易数据库上有少量复杂的查询场景，涉及到的数据量很大，传统解决方案是通过数据同步到数据仓库进行查询。

  OceanBase 数据库的 SQL 引擎同时满足 OLTP 和 OLAP 场景，采用经过 Oracle 复杂业务场景检验的先进的 SQL 优化器技术，能支持复杂的 SQL 优化和高效执行。因此可以在交易数据库上直接做复杂查询，减少不必要的数据同步。此外，OceanBase 还提供不同程度的读写分离技术来控制复杂查询对交易场景的影响。

其他更多场景待实践总结，敬请关注。

联系我们

欢迎 OceanBase 爱好者、用户和客户联系我们反馈问题:

- 社区版官网论坛：https://open.oceanbase.com/answer
- 社区版项目网站提 `Issue`：https://github.com/oceanbase/oceanbase/issues
- 钉钉群：群号 `33254054`

第 2 章：如何部署 OceanBase 社区版

本章介绍如何手动和自动部署 OceanBase 社区版集群，包括单副本和三副本集群。

本章目录

2.1 部署准备

OceanBase 数据库是一个分布式集群产品，在生产环境中至少要求三台机器，学习环境可以部署单机版本。OceanBase 数据库的部署跟传统数据库的部署相比，存在很多共同的地方，对操作系统硬件、软件设置、文件系统等会有一些最佳实践建议，这些是 OceanBase 数据库发挥高性能稳定运行的基础。社区版提供了一些工具保证 OceanBase 数据库可以实现一定程度的自动化。

软件介绍

OceanBase 数据库本质上是一个单进程的软件，可执行文件名为 `observer`。OceanBase 数据库可以通过 RPM 包安装，也可以通过源码编译安装。本教程介绍如何通过 RPM 包安装 OceanBase 数据库。

软件包下载地址如下:

- 官网下载：https://open.oceanbase.com/softwareCenter/community
- GitHub 下载：https://github.com/oceanbase/oceanbase/releases/
- 阿里云 Yum 源：https://mirrors.aliyun.com/oceanbase/OceanBase.repo

软件包名	进程名	软件用途
`oceanbase-ce`-x.x.x-`*.rpm`	`observer`	`OceanBase` 数据库进程，常驻后台运行。
`oceanbase-ce-libs`-x.x.x-`*.rpm`	--	提供软件运行的 `library`，不运行。
`obproxy`-x.x.x-`*.rpm`	`obproxy`	`OceanBase` 访问反向代理，单进程，常驻后台运行。
`ob-deploy`-x.x.x-`*.rpm`	obd	`OceanBase` 自动化部署软件，提供部署命令行，不常驻后台运行。
`libobclient`-x.x.x-`*.rpm`	--	`obclient` 依赖包。
`obclient`-x.x.x-`*.rpm`	`obclient`	`OceanBase` 官方命令行客户端。

注意

- 版本号后期可能会发生变化，请您以实际为准，建议使用最新版本。
- 请根据自己使用的操作系统版本下载对应的软件包。

如果机器可以连接公网，您可将阿里云 YUM 源添加到本地仓库，使用 yum 命令安装。

```
yum install -y yum-utils
```

```
yum-config-manager --add-repo https://mirrors.aliyun.com/oceanbase/OceanBase.repo
yum -y install ob-deploy oceanbase-ce obclient obproxy
```

部署资源要求

OceanBase 数据库运行时会对主机资源有一些要求，主要是 CPU、内存和磁盘空间。安装 OceanBase 数据库的目的不同，对资源的要求也不同。

目的	CPU(核数)	可用内存	磁盘	备注
功能学习	2	10G	10G	不初始化数据。
性能测试	24	128G	SSD 500G 以上	数据盘和日志盘要分开。
生产环境	32	256G	SSD 2T 以上	数据盘和日志盘要分开。日志盘大小是内存的 3-4 倍。如果数据量增长，数据盘大小也要增加。

注意

这里性能测试环境和生产环境的资源要求只是建议。在社区版后续版本，会进一步降低对内存的要求。

OceanBase 数据库对操作系统也有一些要求，目前仅支持以下系统:

- Redhat / CentOS 7.x/8.x
- SUSE / OpenSUSE 15.x
- Anlios 7.x/8.x
- Debian 9.x
- Ubuntu 20.x

部署过程

自动化部署过程分为以下几步:

- 初始化 OceanBase 数据库各个节点环境，包括参数配置、文件系统目录设置等。
- 初始化中控机到 OceanBase 数据库各个节点的 SSH 免密登录。
- 准备 OBD 自动化部署配置文件。
- 使用 OBD 部署集群节点目录。
- 使用 OBD 启动并初始化集群。

说明

随后章节会详细介绍 单节点集群 和 三节点集群 的部署方法，以及 手动部署 的步骤。

2.2 如何快速体验 OceanBase 数据库

在部署 OceanBase 社区版之前，您可通过 Docker 环境快速体验部署好的 OceanBase 社区版环境。我们提供了一个 OceanBase 社区版 Docker 镜像，您可在您的电脑上使用 Docker 技术快速部署并启动 OceanBase 社区版的 Docker 容器。

机器资源要求

OceanBase Docker 容器对资源的要求如下:

- 机器可用内存不少于 10G。

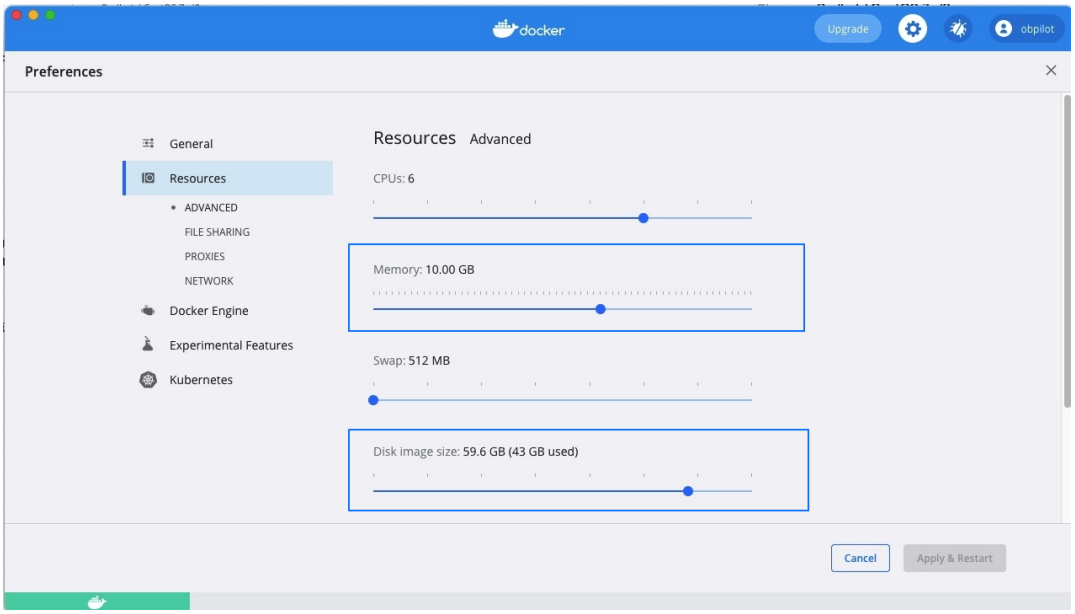
  > **注意**
  >
  > 此处的可用内存指剩余可用内存。

- CPU 建议至少有 2 个逻辑 CPU。

安装 Docker

Docker 是一个免费软件，在 Windows、Linux、Mac 系统里均可安装运行。下载和安装地址如下: https://docs.docker.com/get-docker/。

Docker 安装后，对默认的容器资源有限制，需要手动调整。下面以 Mac 电脑上的 Docker 设置为例进行说明。



常用 Docker 命令参考

```
# 查看docker版本
docker version
# 显示docker系统的信息
docker info
# 查看当前正在运行的容器
docker ps
# 故障检查
service docker status
# 启动关闭docker
service docker start | stop
# 查看容器日志
docker logs -f <容器名 or ID>

# 清理命令，危险！！！
# 清理不用的容器
docker container prune
# 清理不用的镜像
docker image prune
# 清理不用的卷
docker volume prune
```

下载镜像并启动

OceanBase Docker 镜像地址：https://hub.docker.com/r/oceanbase/oceanbase-ce

镜像在 Github 上的源码地址：https://github.com/oceanbase/oceanbase/tree/master/tools/docker/standalone

若您有兴趣可点击链接查看详情。

- （可选）拉取 OceanBase 数据库相关镜像

```
docker search oceanbase    # 搜索 oceanbase 相关镜像

docker pull oceanbase/oceanbase-ce
```

> **说明**
>
> 上述命令默认拉取最新版本，可根据实际需求在 Docker 镜像 中选择版本。

- 启动 OceanBase Docker 容器

```
# 根据当前容器部署最大规格的实例
docker run -p 2881:2881 --name obstandalone -d oceanbase/oceanbase-ce

## 部署 mini 的独立实例
docker run -p 2881:2881 --name obstandalone -e MINI_MODE=1 -d oceanbase/oceanbase-ce
```

输出：

```
→  ~ docker run -p 2881:2881 --name obstandalone -d oceanbase/oceanbase-ce
9e1607d1a0d933622e0a927a576069fa2e43154cdacf215d3e0f2c412c65c102

→  ~ docker ps
CONTAINER ID   IMAGE                     COMMAND              CREATED          STATUS          PO
RTS                     NAMES
9e1607d1a0d9   oceanbase/oceanbase-ce    "/bin/sh -c _boot"   40 seconds ago   Up 39 seconds   0.
0.0.0:2881->2881/tcp    obstandalone
```

- 查看容器启动日志

  刚启动的 OceanBase 数据库需要几分钟初始化集群。您可运行如下命令查看容器启动日志。

```
docker logs obstandalone
```

  输出:

```
→  ~ docker logs obstandalone
generate boot.yaml ...
create boot dirs and deploy ob cluster ...
Package oceanbase-ce-3.1.3 is available.
install oceanbase-ce-3.1.3 for local ok
Cluster param config check ok
Open ssh connection ok
Generate observer configuration ok
oceanbase-ce-3.1.3 already installed.
+-----------------------------------------------------------------------------------------------+
|                                          Packages                                             |
+--------------+---------+----------------------+-------------------------------------------+
| Repository   | Version | Release              | Md5                                       |
+--------------+---------+----------------------+-------------------------------------------+
| oceanbase-ce | 3.1.3   | 10000292022032916.el7 | eab08e5d473bd4884fdf2ac4d7dff6a329b68abe |
+--------------+---------+----------------------+-------------------------------------------+
Repository integrity check ok
Parameter check ok
Open ssh connection ok
Remote oceanbase-ce-3.1.3-eab08e5d473bd4884fdf2ac4d7dff6a329b68abe repository install ok
Remote oceanbase-ce-3.1.3-eab08e5d473bd4884fdf2ac4d7dff6a329b68abe repository lib check !!
[WARN] 127.0.0.1 oceanbase-ce-3.1.3-eab08e5d473bd4884fdf2ac4d7dff6a329b68abe require: libmariadb
.so.3

Try to get lib-repository
Package oceanbase-ce-libs-3.1.3 is available.
install oceanbase-ce-libs-3.1.3 for local ok
Use oceanbase-ce-libs-3.1.3-c68c3aca8a1329a360fe5d65e1c3d4fa0f93f2d5 for oceanbase-ce-3.1.3-eab0
8e5d473bd4884fdf2ac4d7dff6a329b68abe
Remote oceanbase-ce-libs-3.1.3-c68c3aca8a1329a360fe5d65e1c3d4fa0f93f2d5 repository install ok
Remote oceanbase-ce-3.1.3-eab08e5d473bd4884fdf2ac4d7dff6a329b68abe repository lib check ok
Cluster status check ok
Initializes observer work home ok
obcluster deployed
Get local repositories and plugins ok
Open ssh connection ok
Load cluster param plugin ok
Check before start observer ok
```

```
[WARN] (127.0.0.1) clog and data use the same disk (/)

Start observer ok
observer program health check ok
Connect to observer ok
Initialize cluster
Cluster bootstrap ok
Wait for observer init ok
+-----------------------------------------------+
|                    observer                   |
+-----------+---------+------+-------+---------+
| ip        | version | port | zone  | status  |
+-----------+---------+------+-------+---------+
| 127.0.0.1 | 3.1.3   | 2881 | zone1 | active  |
+-----------+---------+------+-------+---------+

obcluster running
Get local repositories and plugins ok
Open ssh connection ok
Connect to observer ok
Create tenant test ok
start ob cluster ...
Get local repositories and plugins ok
Open ssh connection ok
Load cluster param plugin ok
Cluster status check ok
Deploy "obcluster" is running
boot success!
```

分析上面日志可以看出几点信息：

1. OceanBase 会安装两个软件包：`oceanbase-ce-lib` 和 `oceanbase-ce-3.1.3`。
2. 启动 OceanBase 数据库会先初始化集群目录。
3. 之后初始化集群（`bootstrap`）。
4. 最后初始化业务租户（`tenant`）。

连接 OceanBase 数据库实例

oceanbase-standalone 镜像包含 obclient（OceanBase 数据库客户端）和默认连接脚本 ob-mysql。

```
docker exec -it obstandalone ob-mysql sys # 连接 sys 租户
docker exec -it obstandalone ob-mysql root # 连接用户租户的 root 账户
docker exec -it obstandalone ob-mysql test # 连接用户租户的 test 账户
```

或者，您可以运行以下命令以使用本地 obclient 或 MySQL 客户端连接到 OceanBase 实例。

```
$mysql -uroot -h127.1 -P2881
```

连接 OceanBase 实例成功后，终端返回如下信息：

```
  →  ~ docker exec -it obstandalone ob-mysql sys
  login as root@sys
  Command is: obclient -h127.1 -uroot@sys -A -Doceanbase -P2881
  Welcome to the OceanBase.  Commands end with ; or \g.
  Your OceanBase connection id is 3221488602
  Server version: 5.7.25 OceanBase 3.1.3 (r10000292022032916-3d79cacb37012cf61b7cb8faf00d9a6bb152bcd1
) (Built Mar 29 2022 08:20:39)

  Copyright (c) 2000, 2022, OceanBase and/or its affiliates. All rights reserved.

  Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

  obclient [oceanbase]>
```

OceanBase 数据库进程特点

- 运行以下命令进入容器

```
docker exec -it obstandalone bash
```

- 查看 OceanBase 社区版的 YUM 仓库

```
[root@9e1607d1a0d9 /]# cat /etc/yum.repos.d/OceanBase.repo

输出:
# OceanBase.repo

[oceanbase.community.stable]
name=OceanBase-community-stable-el$releasever
baseurl=http://mirrors.aliyun.com/oceanbase/community/stable/el/$releasever/$basearch/
enabled=1
gpgcheck=1
gpgkey=http://mirrors.aliyun.com/oceanbase/RPM-GPG-KEY-OceanBase

[oceanbase.development-kit]
name=OceanBase-development-kit-el$releasever
baseurl=http://mirrors.aliyun.com/oceanbase/development-kit/el/$releasever/$basearch/
enabled=1
gpgcheck=1
gpgkey=http://mirrors.aliyun.com/oceanbase/RPM-GPG-KEY-OceanBase
```

- 查看 observer 进程特点

  分析 OceanBase 集群节点进程，首先通过下面命令确定其启动位置、启动文件和启动参数等。

```
yum -y install sysvinit-tools
```

```
[root@9e1607d1a0d9 ~]# ps -ef|grep observer
root       100     1 99 08:55 ?        00:50:19 /root/ob/bin/observer -r 127.0.0.1:2882:2881 -o
__min_full_resource_pool_memory=268435456,enable_syslog_recycle=True,enable_syslog_wf=True,max_s
yslog_file_count=4,memory_limit=8G,system_memory=4G,cpu_count=16,datafile_size=27G,clog_disk_uti
```

```
lization_threshold=92,clog_disk_usage_limit_percentage=98 -z zone1 -p 2881 -P 2882 -n obcluster
-c 1 -d /root/ob/store -i lo -l INFO
root       809   744  0 09:28 pts/0    00:00:00 grep --color=auto observer

[root@9e1607d1a0d9 ~]# ll /proc/`pidof observer`/{cwd,exe,cmdline}
-r--r--r-- 1 root root 0 Apr 12 08:55 /proc/100/cmdline
lrwxrwxrwx 1 root root 0 Apr 12 08:55 /proc/100/cwd -> /root/ob
lrwxrwxrwx 1 root root 0 Apr 12 08:55 /proc/100/exe -> /root/.obd/repository/oceanbase-ce/3.1.3/
eab08e5d473bd4884fdf2ac4d7dff6a329b68abe/bin/observer

[root@9e1607d1a0d9 ~]# cat /proc/`pidof observer`/cmdline
/root/ob/bin/observer -r 127.0.0.1:2882:2881 -o __min_full_resource_pool_memory=268435456,enable
_syslog_recycle=True,enable_syslog_wf=True,max_syslog_file_count=4,memory_limit=8G,system_memory
=4G,cpu_count=16,datafile_size=27G,clog_disk_utilization_threshold=92,clog_disk_usage_limit_perc
entage=98 -z zone1 -p 2881 -P 2882 -n obcluster -c 1 -d /root/ob/store -i lo -l INFO
```

从上面可以看出 `observer` 进程几点信息:

- 进程启动目录在 `/root/ob` 下。
- 进程可执行文件目录在 `/root/.obd/repository/oceanbase-ce/3.1.3/eab08e5d473bd4884fdf2ac4d7dff6a329b68abe/bin/` 下。

  说明

  这个目录是 OBD 安装 OceanBase 软件的目录，目录中带了具体的版本号。目录较长，OBD 后面版本已将这个目录映射到 `/root/ob/bin/` 下。
- 进程的启动参数很长，部分参数含义后文将进行详细介绍。

- 查看进程监听端口。

```
yum install -y net-tools

netstat -ntlp

输出:
[root@9e1607d1a0d9 ~]# netstat -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:2881            0.0.0.0:*               LISTEN      100/observer
tcp        0      0 0.0.0.0:2882            0.0.0.0:*               LISTEN      100/observer
```

`observer` 进程会监听 2 个端口

- 连接端口 2881。
- RPC 通信端口 2882。

- 查看 OceanBase 数据库工作目录结构

```
yum -y install tree
```

```
[root@9e1607d1a0d9 ~]# tree /root/ob/
/root/ob/
|-- admin
|-- bin
|   `-- observer -> /root/.obd/repository/oceanbase-ce/3.1.3/eab08e5d473bd4884fdf2ac4d7dff6a329b
68abe/bin/observer
|-- etc
|   |-- observer.config.bin
|   `-- observer.config.bin.history
|-- etc2
|   |-- observer.conf.bin
|   `-- observer.conf.bin.history
|-- etc3
|   |-- observer.conf.bin
|   `-- observer.conf.bin.history
|-- lib
|   |-- libaio.so -> /root/.obd/repository/oceanbase-ce/3.1.3/eab08e5d473bd4884fdf2ac4d7dff6a329
b68abe/lib/libaio.so
|   |-- libaio.so.1 -> /root/.obd/repository/oceanbase-ce/3.1.3/eab08e5d473bd4884fdf2ac4d7dff6a3
29b68abe/lib/libaio.so.1
|   |-- libaio.so.1.0.1 -> /root/.obd/repository/oceanbase-ce/3.1.3/eab08e5d473bd4884fdf2ac4d7df
f6a329b68abe/lib/libaio.so.1.0.1
|   |-- libmariadb.so -> /root/.obd/repository/oceanbase-ce/3.1.3/eab08e5d473bd4884fdf2ac4d7dff6
a329b68abe/lib/libmariadb.so
|   `-- libmariadb.so.3 -> /root/.obd/repository/oceanbase-ce/3.1.3/eab08e5d473bd4884fdf2ac4d7df
f6a329b68abe/lib/libmariadb.so.3
|-- log
|   |-- election.log
|   |-- election.log.wf
|   |-- observer.log
|   |-- observer.log.20220412092815
|   |-- observer.log.wf
|   |-- observer.log.wf.20220412092815
|   |-- rootservice.log
|   `-- rootservice.log.wf
|-- run
|   |-- mysql.sock
|   `-- observer.pid
`-- store
    |-- clog
    |   `-- 1
    |-- ilog
    |   `-- 1
    |-- slog
    |   `-- 1
    `-- sstable
        `-- block_file
```

若您是手动部署 OceanBase 数据库节点，该工作目录下的子目录结构需手动维护。否则，`observer` 可能会启动失败。

若您使用自动化部署软件 OBD 部署 OceanBase 数据库节点，OBD 会自动创建相应目录。

目录路径（相对于工作目录）	备注
etc、etc2、etc3	配置文件所在目录。

log	运行日志目录。
run	运行输出目录，输出 pid 文件。
store	数据（包括日志）所在总目录。
`store/clog`	`commit log` 所在目录。
`store/ilog`	ilog 所在目录。
`store/slog`	slog 所在目录。
`store/sstable`	数据文件 block file 所在目录。

注意

该 Docker 示例把 OceanBase 数据库安装在 `root` 用户目录下，并以 `root` 用户运行，这里只是用作学习。生产环境中请勿以 `root` 用户部署和运行 OceanBase 数据库。

2.3 如何规划 OceanBase 集群部署

集群架构规划

OceanBase 数据库以集群形态运行，生产环境中最小规模为 3 台服务器（节点）。即集群中业务数据有三份，所以也叫三副本。 您在学习测试的时候，可以部署单副本单节点 OceanBase 集群。

> **注意**
>
> 单副本跟单节点并不完全对等。单副本单节点是最小集群规模，单副本也可以扩容为多个节点，整个集群里数据依然是一份，所以叫单副本。

生产环境中，每台机器上启动一个 `observer` 进程，所以一台机器就对应一个节点。

学习环境中，一台机器可以启动多个 `observer` 进程，模拟多个节点。每个节点的监听端口（默认是 2881 和 2882）、数据总目录是独立的，互不冲突。每个节点进程启动的最小内存是 10G。

只有一台服务器时，启动的进程数可以分为以下几种情况:

- 机器可用内存不足 10G， 不能启动 `observer` 进程。
- 机器可用内存在 10G ~ 20G 之间，只可以启动一个 `observer` 进程。
- 机器可用内存在 20G ~ 30G 之间，可以启动 2 个 `observer` 进程。
- 机器可用内存超过 30G ，可以启动 3 个 `observer` 进程。

内存充足时，也可以调大每个 `observer` 进程能获取的内存。内存越大，节点的资源能力就越大。当然，如果有三台机器，则无需在一台机器上模拟多个节点。

您在部署 `observer` 进程后还需要部署 `obproxy`。`obproxy` 也是单进程软件，是访问 OceanBase 数据库的反向代理。虽然 `observer` 节点可以直接访问，但是生产环境中还是建议通过 `obproxy` 访问 OceanBase 集群。

`obproxy` 进程对于部署位置没有要求。您可以部署在应用服务器上，也可以部署在独立的机器上，或者部署在 OceanBase 机器上。

`obproxy` 可以部署多个，生产环境中建议至少部署两个。

用户规划

OceanBase 数据库本质上是一个软件，可以运行在任意用户下。OceanBase 数据库软件包默认解压目录在 `/home/admin/` 下，生产环境默认也是安装在用户 `admin` 下，社区版的软件 RPM 包也是如此。OceanBase 支持部署在任意用户的任意目录下。

> **说明**

为了安全起见，不建议在 `root` 用户下直接部署。随后章节均以部署在 `admin` 用户下为前提。

在部署 OceanBase 之前需初始化环境，此时可能需要修改操作系统的配置，或者设置目录的权限等，这些操作需要 `root` 用户权限。不同客户内部主机登录规范不一样，您可以通过 `su` 切换到 `root` 用户，或者给 `admin` 用户增加 `sudo` 权限。

目录规划

和 `observer` 相关的软件安装目录有如下几个:

- 如果您使用 RPM 包安装 OceanBase 数据库，则需要提前创建好 `admin` 用户，数据库的 RPM 包将被自动安装在目录 `/home/admin/oceanbase` 下。

```
[root@obce00 ~]# useradd admin
[root@obce00 ~]# rpm -ivh rpm/*
准备中...                          ################################# [100%]
正在升级/安装...
   1:oceanbase-ce-libs-x.x.x.el7    ################################# [ 33%]
   2:oceanbase-ce-x.x.x.el7         ################################# [ 67%]
   3:obproxy-x.x.x.el7              ################################# [100%]

[root@obce00 ~]# rpm -ql oceanbase-ce-x.x.x.el7.x86_64
/home/admin/oceanbase
/home/admin/oceanbase/bin
/home/admin/oceanbase/bin/import_time_zone_info.py
/home/admin/oceanbase/bin/observer
/home/admin/oceanbase/etc
/home/admin/oceanbase/etc/timezone_V1.log

[root@obce00 ~]# rpm -ql obproxy-x.x.x.el7
/home/admin/obproxy-x.x.x/bin
/home/admin/obproxy-x.x.x/bin/obproxy
/home/admin/obproxy-x.x.x/bin/obproxyd.sh
```

- 如果您通过 OBD 软件自动化安装 OceanBase 数据库，则会将 RPM 包解压到 HOME 目录的隐藏文件夹 `.obd` 下，如：`/.obd/repository/oceanbase-ce/3.1.0/afd11d52f83eef4b456d77969fde620c4bfba85e` 。这种方式可以同时部署多个版本。

说明

随后章节讲解部署方法时会首先介绍 OBD 软件自动化部署 方法。手动部署 方法留在最后，供感兴趣的用户参考。

2.4 如何初始化服务器环境

OceanBase 数据库是单进程软件，使用时需要访问网络、打开多个文件以及开启很多 TCP 连接，所以需要修改内核参数和用户会话设置。

注意

您如果在独立服务器上部署 OBProxy，也需按本文要求初始化服务器环境。

内核参数修改

您可参考以下命令修改配置文件：

```
vim /etc/sysctl.conf

net.core.somaxconn = 2048
net.core.netdev_max_backlog = 10000
net.core.rmem_default = 16777216
net.core.wmem_default = 16777216
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216

net.ipv4.ip_local_port_range = 3500 65535
net.ipv4.ip_forward = 0
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.tcp_syncookies = 0
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.tcp_fin_timeout = 15
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_slow_start_after_idle=0

vm.swappiness = 0
vm.min_free_kbytes = 2097152
vm.max_map_count=655360
fs.aio-max-nr=1048576
```

运行以下命令可让配置生效：

```
sysctl -p
```

修改会话变量设置

您可以通过配置 `limits.conf` 修改会话限制。OceanBase 数据库的进程涉及的限制包括：线程最大栈空间大小（Stack）、最大文件句柄数（Open Files）和 core 文件大小（Core File Size）。

您可以使用以下两种方法修改资源限制：

- 通过启动时在会话级别修改。如：`ulimit -c unlimited`，通过此方法修改只影响当前会话。如果会话断开之后重连，则又会是默认配置。
- 通过配置文件 `/etc/security/limits.conf` 在全局级别修改。

> **注意**
>
> 修改后，已经登录的会话需要退出后重新登录才生效。

更改配置文件

您可将会话级别的最大栈空间大小设置为 `unlimited`，最大文件句柄数设置为 `655350`，Core 文件大小设置为 `unlimited`。 如果已有设置值低于这个设置值，则按照以下命令修改 `/etc/security/limits.conf` 配置文件。

```
vi /etc/security/limits.conf

* soft nofile 655360
* hard nofile 655360
* soft nproc 655360
* hard nproc 655360
* soft core unlimited
* hard core unlimited
* soft stack unlimited
* hard stack unlimited
```

查看配置

您可退出当前会话，重新登录。执行以下命令，查看配置是否生效：

```
ulimit -a
```

关闭防火墙和 SELinux

不同操作系统的防火墙设置可能会有不同，下文以 CentOS 系统为例进行讲解。

关闭防火墙

查看防火墙状态。

```
systemctl status firewalld
```

如果当前防火墙状态为 `inactive`，则不需要关注。若当前防火墙状态为 `active`，则需要永久关闭。

```
systemctl disable firewalld
systemctl stop firewalld
systemctl status firewalld
```

关闭 SELinux

修改 SELinux 配置文件中的 `SELINUX` 选项。

```
vi /etc/selinux/config

# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=disabled
```

注意

`SELINUX` 选项必须使用注释中的三个值之一。如果填写错误，机器重启后操作系统会报错，之后就只能通过进入单用户模式修改。

配置文件修改后需等到重启主机后才可生效，您可使用下面命令使其立即生效。

```
setenforce 0
```

配置时间同步服务

OceanBase 数据库是分布式数据库产品，是一个集群软件，对各个节点之间的时间同步性有要求。技术上要求所有节点之间的时间误差需控制在 50ms 以内。实际生产环境中为了稳定性和性能考虑，建议时间误差控制在 10ms 以内。

通常，只要节点配置时间同步服务器跟公网时间保持同步即可。实际上在企业机房里，企业会有统一的时间服务器跟机房提供的时间服务器或者直接跟公网时间服务器同步，OceanBase 节点只需要跟机房统一的时间服务器进行同步即可。

CentOS 或 RedHat 7.x 版本推荐使用 `chrony` 服务做时间源。`Chrony` 是 NTP（即 `Network Time Protocol`，网络时间协议，服务器时间同步的一种协议）的另一种实现。与 `ntpd` 不同的是，`Chrony` 可以更快且更准确地同步系统时钟，最大程度的减少时间和频率误差。

判断是否使用 ntpd 同步时间

```
systemctl status ntpd
Unit ntpd.service could not be found.
```

如果出现上述提示信息，表示没有使用 `ntpd`，您可继续进行后续步骤。

如果提示有 `ntpd` 服务，则需卸载 `ntpd` 软件。

安装 chrony 服务

这里采用 YUM 安装方法。您也可以下载相应的 RPM 包安装。

```
yum -y install chrony
```

chrony 配置说明

`chrony` 服务守护进程名为 `chronyd`，`chronyc` 是用来监控 `chronyd` 性能和配置参数的命令行工具。`chrony` 的主配置文件为 `/etc/chrony.conf`。配置方法如下：

```
vi /etc/chrony.conf

# server 后面跟时间同步服务器
# 使用 pool.ntp.org 项目中的公共服务器。按 server 配置，理论上您想添加多少时间服务器都可以。
# 或者使用 阿里云的 ntp 服务器
# Please consider joining the pool (http://www.pool.ntp.org/join.html).
server ntp.cloud.aliyuncs.com minpoll 4 maxpoll 10 iburst
server ntp.aliyun.com minpoll 4 maxpoll 10 iburst
server ntp1.aliyun.com minpoll 4 maxpoll 10 iburst
server ntp1.cloud.aliyuncs.com minpoll 4 maxpoll 10 iburst
server ntp10.cloud.aliyuncs.com minpoll 4 maxpoll 10 iburst

# 如果是测试环境，没有时间同步服务器，那就选取一台配置为时间同步服务器。
# 如果选中的是本机，则取消下面 server 注释
#server 127.127.1.0

# 根据实际时间计算出服务器增减时间的比率，然后记录到一个文件中，在系统重启后为系统做出最佳时间补偿调整。
driftfile /var/lib/chrony/drift

# chronyd 根据需求减慢或加速时间调整，
# 在某些情况下系统时钟可能漂移过快，导致时间调整用时过长。
# 该指令强制 chronyd 调整时期，大于某个阀值时步进调整系统时钟。
# 只有在因 chronyd 启动时间超过指定的限制时（可使用负值来禁用限制）没有更多时钟更新时才生效。
makestep 1.0 3

# 将启用一个内核模式，在该模式中，系统时间每 11 分钟会拷贝到实时时钟（RTC）。
rtcsync

# Enable hardware timestamping on all interfaces that support it.
# 通过使用 hwtimestamp 指令启用硬件时间戳
#hwtimestamp eth0
#hwtimestamp eth1
#hwtimestamp *
```

```
# Increase the minimum number of selectable sources required to adjust
# the system clock.
#minsources 2

# 指定一台主机、子网，或者网络以允许或拒绝 NTP 连接到扮演时钟服务器的机器
#allow 192.168.0.0/16
#deny 192.168/16

# 即使没有同步到时间源，也要服务时间
local stratum 10

# 指定包含 NTP 验证密钥的文件。
#keyfile /etc/chrony.keys

# 指定日志文件的目录。
logdir /var/log/chrony

# Select which information is logged.
#log measurements statistics tracking
```

最简单的配置文件如下:

```
server 127.127.1.0
allow 172.20.0.0/16
local stratum 10
```

常用命令

使用 `chrony` 时间服务是为了保证 OceanBase 集群各个节点时间尽可能同步，下面这些命令供参考。具体使用请查看 `chrony` 官方使用说明：Chronyc Frequently Asked Questions。

```
# 查看时间同步活动
chronyc activity

# 查看时间服务器
chronyc sources

# 查看同步状态
chronyc sources -v

# 校准时间服务器:
chronyc tracking
```

说明

使用 `clockdiff` 命令可以检查本机跟目标机器的时间同步误差，以该命令结果为准。

(可选）时区设置

如果时间显示跟当前实际时间差异很大，您可参考如下命令查看确认当前系统时区。

```
timedatectl

输出:
[root@obce00 ~]# timedatectl

               Local time: 六 2021-09-11 07:37:22 CST
           Universal time: 五 2021-09-10 23:37:22 UTC
                 RTC time: 六 2021-09-11 07:37:22
                Time zone: Asia/Shanghai (CST, +0800)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: yes

Warning: The system is configured to read the RTC time in the local time zone.
         This mode cannot be fully supported. It will create various problems
         with time zone changes and daylight saving time adjustments. The RTC
         time is never updated, it relies on external facilities to maintain it.
         If at all possible, use RTC in UTC by calling
         'timedatectl set-local-rtc 0'.
```

您也可使用如下命令查看所有可用时区。

```
timedatectl list-timezones
```

设置当前系统时区方法如下。

```
timedatectl set-timezone Asia/Shanghai

chronyc -a makestep

输出:
[root@obce00 ~]# chronyc -a makestep
200 OK
```

设置完时区后，您可强制同步系统时钟。

配置安装用户

前文介绍过，建议安装部署在普通用户下，后文均以用户 `admin` 为例。

注意

为 `admin` 用户赋与 `sudo` 权限不是必须的，只是为了某些时候方便操作。您可以结合企业安全规范决定是否执行。

下面是创建用户 `admin` 并授予 `sudo` 权限的方法，仅供参考。

```
# 新增普通用户 admin
```

```
useradd admin

# 修改用户密码
passwd admin

# 或运行下面命令指定密码，密码修改为自己的。

echo 'admin:adminPWD123' | chpasswd
```

在 CentOS 上为 `admin` 用户增加 `sudo` 权限有以下两个方法:

- 把用户加到 用户组 `wheel` 里。
- 把用户加到 `/etc/sudoers` 文件里。

```
# 如果 sudo 不存在，就安装 sudo
yum install -y sudo

# 方法一: admin 加到用户组 wheel 里。
[root@obce00 ~]# usermod admin -G wheel
[root@obce00 ~]# id admin
uid=1000(admin) gid=1000(admin) groups=1000(admin),10(wheel)


# 方法二: admin 添加到  /etc/sudoers 文件中
[root@obce00 ~]# cat /etc/sudoers |grep wheel
## Allows people in group wheel to run all commands
%wheel  ALL=(ALL)       ALL
# %wheel        ALL=(ALL)       NOPASSWD: ALL

vim /etc/sudoers
## Allow root to run any commands anywhere
admin    ALL=(ALL)       ALL
```

验证方法是否生效，切换到 `admin` 用户下，执行命令：`sudo date`。输入密码后查看返回结果。

配置 SSH 免密登录

如果您是完全手动部署 OceanBase 集群，登录到相应节点上安装相关软件包，并启动 `observer` 或 `obproxy` 进程，则不需要配置 SSH 免密登录。

如果您是使用自动化技术部署 OceanBase 集群，所有的命令通过中控机向 OceanBase 集群节点发出。则需要配置中控机中 OBD 运行的用户到 OceanBase 集群节点中 OBServer 安装的用户的 SSH 免密登录。

本文示例是中控机的用户 `admin` 到 OBServer 节点的用户 `admin` 的免密登录。

配置 SSH 免密登录方法有很多，这里选择将中控机的 RSA 或 DSA 公钥复制到目标节点的 SSH 配置文件中。

- 在中控机生成 RSA 或 DSA 公钥和私钥

  ```
  ssh-keygen -t rsa
  ```

```
输出:
[admin@obce00 ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/admin/.ssh/id_rsa):
Created directory '/home/admin/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/admin/.ssh/id_rsa.
Your public key has been saved in /home/admin/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:7yCIks5NT8j7L1XIq+gRL3qm04cvHTSQmlaNr4gdHqc admin@obce00
The key's randomart image is:
+---[RSA 3072]----+
|    +            |
|   = .           |
|  + o . .        |
| +o .+ o .       |
|oo.*o . S        |
|.oEo+o o .       |
|o o*=o= . .      |
|oo+B*= . o       |
| =*+=+o.  .      |
+----[SHA256]-----+
[admin@obce00 ~]$

[admin@obce00 ~]$ ls -al .ssh/
total 8
drwx------ 2 admin admin   38 Sep 11 14:43 .
drwx------ 4 admin admin  115 Sep 11 14:43 ..
-rw------- 1 admin admin 2602 Sep 11 14:43 id_rsa
-rw-r--r-- 1 admin admin  569 Sep 11 14:43 id_rsa.pub
```

上面命令会在用户的 HOME 目录中生成文件夹 `.ssh`。注意不要改变文件夹以及里面文件的访问权限。

- 打通到本机的 SSH 免密登录

  复制 RSA 或 DSA 公钥到目标节点，推荐使用命令 `ssh-copy-id`。

```
[admin@obce00 ~]$ ssh-copy-id `hostname -i`
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/admin/.ssh/id_rsa.pub"
The authenticity of host '172.20.249.50 (172.20.249.50)' can't be established.
ECDSA key fingerprint is SHA256:Zyyq5dY+05pkdqGCm6K43s97l8DUGv0LjY5t+zrdVkE.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that ar
e already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to
install the new keys
admin@172.20.249.50's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh '172.20.249.50'"
and check to make sure that only the key(s) you wanted were added.

[admin@obce00 ~]$ ls -al .ssh
total 16
```

```
drwx------ 2 admin admin   80 Sep 11 14:44 .
drwx------ 4 admin admin  115 Sep 11 14:43 ..
-rw------- 1 admin admin  569 Sep 11 14:44 authorized_keys
-rw------- 1 admin admin 2602 Sep 11 14:43 id_rsa
-rw-r--r-- 1 admin admin  569 Sep 11 14:43 id_rsa.pub
-rw-r--r-- 1 admin admin  175 Sep 11 14:44 known_hosts
[admin@obce00 ~]$
```

磁盘文件系统划分

OceanBase 数据库读写磁盘主要是三类文件:

- 运行日志

  位于启动目录下的 `log` 目录。记录进程 `observer` 的运行日志、选举服务的运行日志和 `rootservice` 的运行日志。主要读写特点是顺序写。

- 数据文件

  主要是指数据文件 `block_file`，一次性初始化大小，可以在线扩容，但是不能缩容。主要读写特点是随机读、顺序写，偶尔密集的随机写。

- 事务日志文件

  主要是指事务和 `sstable` 相关的日志，包括 `clog`、`ilog` 和 `slog` 等。主要读写特点是顺序写。

这三个文件尽可能分散在不同的磁盘上存储。如果只有一块盘，则可以使用 `fdisk` 或 `lvm` 划分为多个逻辑盘。

下面针对机器提供的裸盘（`/dev/vdb`) 演示如何分盘。

- 方法一：使用 `fdisk` 直接将 `/dev/vdb` 划分为两个逻辑盘 (`/dev/vdb1` 和 `/dev/vdb2` ）。

  注意

  这个方法存在缺陷，`/dev/vdb` 是云盘，可以扩容，但是使用 `fdisk` 分盘后，扩容比较麻烦。

- 方法二：对 `/dev/vdb` 使用 LVM 技术，划分两个 LV，一个给数据文件用，一个给日志文件。

`fdisk` 或者 `parted`，以及 LVM 技术都是磁盘划分组合的手段。这里就不详细描述方法。

无论使用哪种办法，需优先考虑事务日志文件的大小，生产环境下建议是可用内存大小的 3-4 倍。剩余的大小再留给数据文件。

学习环境下，总的盘本身就很小，可以不遵守这个规则，日志文件大小比内存大 1-2 倍即可。

注意

OBProxy 独立部署的服务器不需要做文件系统划分，OBProxy 只有运行日志目录。

2.5 如何安装 OBD 自动化部署软件

OBD 全称为 OceanBase Deployer，是 OceanBase 数据库社区版的命令行自动化部署软件。 根据中控机器能否连接公网，提供离线和在线两种安装方法，您可根据实际情况选择安装方式。

安装 OBD 软件（离线）

您需要在中控机上部署 OBD 软件。如果中控机不能上网，则需要提前下载好 OBD、OBServer 和 OBProxy 相关软件包。

下载相关软件包

请根据使用的操作系统版本下载对应的软件包。

- Redhat / CentOS 7.x 下载地址：https://mirrors.aliyun.com/oceanbase/community/stable/el/7/x86_64
- Redhat / CentOS 8.x 下载地址：https://mirrors.aliyun.com/oceanbase/community/stable/el/8/x86_64

下载 Redhat / CentOS 8.x 相关软件命令:

```
wget https://mirrors.aliyun.com/oceanbase/community/stable/el/8/x86_64/ob-deploy-x.x.x-*.rpm
wget https://mirrors.aliyun.com/oceanbase/community/stable/el/8/x86_64/oceanbase-ce-x.x.x-*.rpm
wget https://mirrors.aliyun.com/oceanbase/community/stable/el/8/x86_64/oceanbase-ce-libs-x.x.x-*.rpm
wget https://mirrors.aliyun.com/oceanbase/community/stable/el/8/x86_64/obclient-x.x.x-*.rpm
wget https://mirrors.aliyun.com/oceanbase/community/stable/el/8/x86_64/libobclient-x.x.x-*.rpm
wget https://mirrors.aliyun.com/oceanbase/community/stable/el/8/x86_64/obproxy-x.x.x-*.rpm
```

说明

请根据实际需求下载软件包，建议使用最新版本的软件。

离线安装 OBD

`ob-deploy` 软件默认安装在 `/usr/obd` 下。不同版本可能有点变化，您可通过以下命令查看安装位置。

```
rpm -ql `rpm -qa|grep ob-deploy`
```

但是 OBD 工作的文件都在当前用户 HOME 目录下：`~/.obd/`。

```
[admin@obce00 ~]$ tree ~/.obd -L 1
/home/admin/.obd
├── cluster
├── log
├── mirror
├── obd.conf
├── plugins
```

```
├── repository
└── version

5 directories, 2 files
```

您可使用 `-h` 查看 OBD 命令使用帮助。

```
obd -h

输出:
[admin@obce00 ~]$ obd -h
Usage: obd <command> [options]

Available commands:

cluster        Deploy and manage a cluster.

mirror         Manage a component repository for OBD.

repo           Manage local repository for OBD.

test           Run test for a running deploy deployment.

update         Update OBD.


Options:
  --version      show program's version number and exit
  -h, --help     Show help and exit.
  -v, --verbose  Activate verbose output.
```

将软件包添加到离线仓库

注意

下面命令需在部署运行 OBD 的操作系统用户下操作。这里是用户 `admin`。

- 您可使用如下命令禁用远程镜像仓库。

  ```
  obd mirror disable remote
  ```

- 使用下面命令将前面的软件包复制到本地仓库。

  ```
  obd mirror clone /tmp/obd/*.rpm
  ```

- 查看仓库的 RPM 列表。

  ```
  obd mirror list local
  ```

输出:

```
[admin@obce00 ~]$ obd mirror list local
+-------------------------------------------------------------------------------------------+
|                                    local Package List                                     |
+-------------------+---------+---------+--------+-----------------------------------------+
| name              | version | release | arch   | md5                                     |
+-------------------+---------+---------+--------+-----------------------------------------+
| libobclient       | 2.0.0   | 2.el8   | x86_64 | 358a90b4a47da193140c3bee023b2450126de4c6 |
| obclient          | 2.0.0   | 2.el8   | x86_64 | 71753559d82e9f6c0b8a6d949b9a5194c6c53dc6 |
| ob-deploy         | 1.1.0   | 1.el8   | x86_64 | 0c84129b699aca0b43fdfb01fb2c4439f36ff856 |
| obproxy           | 3.1.0   | 1.el8   | x86_64 | d242ea5fe45222b8f61c3135ba2aaa778c61ea22 |
| oceanbase-ce      | 3.1.0   | 3.el8   | x86_64 | 84bd2fe27f8b8243cc57d8a3f68b4c50f94aab80 |
| oceanbase-ce-libs | 3.1.0   | 3.el8   | x86_64 | 1c20be0df8929f843e9bdd509de4916f883d62f8 |
+-------------------+---------+---------+--------+-----------------------------------------+
```

安装 OBD 软件（在线）

- 在中控机上部署 OBD 软件。

  如果中控机能上网，您可直接添加 OceanBase 数据库的仓库，使用 YUM 安装。

```
yum install -y yum-utils
yum-config-manager --add-repo https://mirrors.aliyun.com/oceanbase/OceanBase.repo
yum install -y ob-deploy
```

- 查看 `OceanBase.repo` 内容。

```
cat /etc/yum.repos.d/OceanBase.repo

输出:
# OceanBase.repo

[oceanbase.community.stable]
name=OceanBase-community-stable-el$releasever
baseurl=http://mirrors.aliyun.com/oceanbase/community/stable/el/$releasever/$basearch/
enabled=1
gpgcheck=1
gpgkey=http://mirrors.aliyun.com/oceanbase/RPM-GPG-KEY-OceanBase

[oceanbase.development-kit]
name=OceanBase-development-kit-el$releasever
baseurl=http://mirrors.aliyun.com/oceanbase/development-kit/el/$releasever/$basearch/
enabled=1
gpgcheck=1
gpgkey=http://mirrors.aliyun.com/oceanbase/RPM-GPG-KEY-OceanBase
```

2.6 如何使用 OBD 自动化部署单节点集群

OBD 对 OceanBase 数据库的管理权限很高，所以 OBD 需部署在数据库服务器的中控机上，需要 DBA 有完全的控制权限。

部署规划

本文以使用一台机器部署为例进行操作。

机器信息

机器类型	云主机 ECS
IP	172.20.249.50
网卡名	eth0
OS	CentOS Linux release 8.4.2105
CPU	4C
内存	总内存 14G，可用内存 11G
磁盘1	云盘 /dev/vda 100G
磁盘2	云盘 /dev/vdb 100G

机器和角色划分

角色	机器	备注
OBD	172.20.249.50	中控机，自动化部署软件
OBServer	172.20.249.50	OceanBase 数据库
OBProxy	172.20.249.50	OceanBase 访问反向代理
OBClient	172.20.249.50	OceanBase 命令行客户端

磁盘划分

这里使用 LVM 技术对 `/dev/vdb` 进行划分。使用 LVM 技术划分 LV 大小时，请根据实际磁盘大小调整参数。

```
# lvm 分盘
pvcreate /dev/vdb
vgcreate obvg /dev/vdb
```

```
lvcreate -L 20G obvg -n lvredo
lvcreate -l 100%FREE obvg -n lvdata

# 格式化文件系统
mkfs.ext4 /dev/obvg/lvdata
mkfs.ext4 /dev/obvg/lvredo

# 修改 mount 参数文件
vim /etc/fstab
/dev/obvg/lvredo          /redo              ext4            defaults,noatime,nodiratime,nodelalloc,ba
rrier=0        0 0
/dev/obvg/lvdata             /data         ext4            defaults,noatime,nodiratime,nodelalloc,barr
ier=0        0 0

# 挂载文件系统
mkdir -p /data /redo
vim /etc/fstab
mount /data
mount /redo
chown -R admin.admin /data /redo

# 检查
df -h

输出:
文件系统                  容量  已用  可用 已用% 挂载点
/dev/mapper/obvg-lvdata   59G   53M   56G    1% /data
/dev/mapper/obvg-lvredo   20G   45M   19G    1% /redo
```

编辑 OBD 配置文件

OBD 针对不同的部署场景提供了不同的配置文件。这些配置文件示例放在 OceanBase 数据库开源项目地址中，您可访问链接查看：https://github.com/oceanbase/obdeploy/tree/master/example。

如果是部署单节点版本，您可下载其中两个配置文件：

- 部署单节点 `observer` 进程：

  https://github.com/oceanbase/obdeploy/blob/master/example/mini-single-example.yaml

- 部署单节点 `observer` 和 `obproxy` 进程：

  https://github.com/oceanbase/obdeploy/blob/master/example/mini-single-with-obproxy-example.yaml

简单起见，这里只部署单节点 `observer` 进程，所以下载第一个配置文件。

注意

后续版本的配置文件格式可能会变化，具体情况请参考 配置文件说明。

```
[admin@obce00 ~]$ cat obce-single.yaml
# Only need to configure when remote login is required
```

```
# user:
#   username: your username
#   password: your password if need
#   key_file: your ssh-key file path if need
#   port: your ssh port, default 22
#   timeout: ssh connection timeout (second), default 30
oceanbase-ce:
  servers:
    # Please don't use hostname, only IP can be supported
    - 172.20.249.50
  global:
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo
    # Please set devname as the network adaptor's name whose ip is  in the setting of severs.
    # if set severs as "127.0.0.1", please set devname as "lo"
    # if current ip is 192.168.1.10, and the ip's network adaptor's name is "eth0", please use "eth0"
    devname: eth0
    mysql_port: 2881 # External port for OceanBase Database. The default value is 2881.
    rpc_port: 2882 # Internal port for OceanBase Database. The default value is 2882.
    zone: zone1
    cluster_id: 1
    # please set memory limit to a suitable value which is matching resource.
    memory_limit: 8G # The maximum running memory for an observer
    system_memory: 3G # The reserved system memory. system_memory is reserved for general tenants. Th
e default value is 30G.
    stack_size: 512K
    cpu_count: 16
    cache_wash_threshold: 1G
    __min_full_resource_pool_memory: 268435456
    workers_per_cpu_quota: 10
    schema_history_expire_time: 1d
    # The value of net_thread_count had better be same as cpu's core number.
    net_thread_count: 4
    major_freeze_duty_time: Disable
    minor_freeze_times: 10
    enable_separate_sys_clog: 0
    enable_merge_by_turn: FALSE
    # datafile_disk_percentage: 20 # The percentage of the data_dir space to the total disk space. Thi
s value takes effect only when datafile_size is 0. The default value is 90.
    datafile_size: 50G
    syslog_level: WARN # System log level. The default value is INFO.
    enable_syslog_wf: false # Print system logs whose levels are higher than WARNING to a separate lo
g file. The default value is true.
    enable_syslog_recycle: true # Enable auto system log recycling or not. The default value is false.
    max_syslog_file_count: 10 # The maximum number of reserved log files before enabling auto recyclin
g. The default value is 0.
    root_password: b******B  # root user password, can be empty
```

该配置文件是专门针对最小内存（可用内存大于 8G）的节点配置，指定了很多 `observer` 进程的启动参数。您需注意 `yaml` 的格式，每个配置项后面冒号（`:`）跟后面的值之间必须有个空格（`' '`）。

下面对关键的几个参数进行补充说明:

配置项名	配置值	备注
servers	172.20.249.50	本示例是在中控机上部署 OBServer，所以写的是中控机 IP。可以写实际 IP，也可以写 127.0.0.1（仅学习用）。
home_path	/home/admin/oceanbase-ce	指定到普通用户（admin）的目录下，为区别于企业版，文件名叫 oceanbase-ce 。
data_dir	/data	指向独立的磁盘，这里使用前面分配的 LV（lvdata）。实际存储 OceanBase 的数据文件目录（sstable）。
redo_dir	/redo	指向独立的磁盘，这里使用前面分配的 LV（lvredo）。实际存储 OceanBase 的事务日志目录（clog、slog和ilog）。
devname	eth0	跟 servers 里指定的 IP 对应的网卡。如果前面 IP 是 127.0.0.1 ，那么这里就填lo 。通过 ip addr 命令可以查看 IP 和网卡对应关系。
mysql_port	2881	进程 observer 的连接端口，默认是 2881。后面 OceanBase 客户端直连这个端口可以访问该节点。
rpc_port	2882	进程 observer 跟其他节点进程之间的 RPC 通信端口，默认是 2882。
zone	zone1	zone 是逻辑机房的概念。单副本集群下只有一个 zone，默认取名 zone1。三副本集群会有三个 zone，名字随意，不要重复即可。
cluster_id	1	OceanBase 集群 ID 标识，不同集群不要重复即可。
memory_limit	8G	进程 observer 能从OS 获取的最大内存，最小不少于 8G 。如果机器内存丰富的话，这个参数可以大一些。
system_memory	4G	进程 observer 留给集群内部用的保留内存，这个会占用上面 memory_limit 的内存，留给业务租户的就更少。
datafile_size datafile_disk_percentage		这两个参数 2 选 1。用来指定该节点数据文件（ block_file ）大小的。可以按大小指定，或者按磁盘空间的百分比指定。 这里示例磁盘空间很小，为精确控制，指定数据文件大小。 **注意:** 当数据文件和事务日志文件共用一个磁盘时，必须指定数据文件具体大小，以避免日志文件目录空间不够的情况发生。

syslog_level	WARN 或 ERROR	运行日志的日志级别，有 INFO 、WARN、 ERROR 等几个级别。级别越低，日志量越大。进程 observer 的日志量非常大，如果磁盘空间不大的话，就调整为 WARN 或 ERROR 。
enable_syslog_recycle	TRUE	指定运行日志是否以滚动方式输出，最多保留指定数量的运行日志。
max_syslog_file_count	10	根据磁盘空间大小定数值，这里默认保留最多 10 个历史运行日志文件。
root_password	随机字符串	OceanBase 集群的超级管理员root@sys 的密码，建议设置复杂的密码。

部署成功后，OBD 会把配置文件 obce-single.yaml 复制到自己的工作目录里（~/.obd/cluster/obce-single/config.yaml），后期再对原 obce-single.yaml 文件进行修改是不生效的。

说明

如果你机器内存大于 64G，上面和内存相关的参数可以不设置。

部署集群

配置文件准备好后，就可以部署配置文件对应的集群了，部署内容主要包含:

- 复制软件到相应节点，并安装软件。
- 在相应节点创建相关目录。

部署使用命令：obd cluster deploy [集群名] -c集群配置文件。

说明

这里的集群名是配置文件在 OBD 里的唯一标识，可以跟配置文件中的集群名相同，也可以跟文件名相同。

```
obd cluster deploy obce-single -c obce-single.yaml
```

输出:

```
[admin@obce00 ~]$ obd cluster deploy obce-single -c obce-single.yaml
oceanbase-ce-3.1.0 already installed.
+-----------------------------------------------------------------------------+
|                                  Packages                                   |
+--------------+---------+---------+------------------------------------------+
| Repository   | Version | Release | Md5                                      |
+--------------+---------+---------+------------------------------------------+
| oceanbase-ce | 3.1.0   | 3.el8   | 84bd2fe27f8b8243cc57d8a3f68b4c50f94aab80 |
+--------------+---------+---------+------------------------------------------+
Repository integrity check ok
```

```
Parameter check ok
Open ssh connection ok
Remote oceanbase-ce-3.1.0-84bd2fe27f8b8243cc57d8a3f68b4c50f94aab80 repository install ok
Remote oceanbase-ce-3.1.0-84bd2fe27f8b8243cc57d8a3f68b4c50f94aab80 repository lib check ok
Cluster status check ok
Initializes cluster work home ok
obce-single deployed
[admin@obce00 ~]$
```

部署结果

1. 使用命令 `obd cluster list` 查看部署状态。

```
obd cluster list

输出:
[admin@obce00 ~]$ obd cluster list

+-----------------------------------------------------------------------+
|                             Cluster List                              |
+-------------+---------------------------------------+-----------------+
| Name        | Configuration Path                    | Status (Cached) |
+-------------+---------------------------------------+-----------------+
| obce-single | /home/admin/.obd/cluster/obce-single | deployed        |
+-------------+---------------------------------------+-----------------+
```

2. 查看目录结构。

   其中 `/store` 、`/data` 和 `/redo` 的目录关系是重点。总体结构不变，后期映射关系时可能会进行细微调整。

```
[admin@obce00 ~]$ tree /home/admin/oceanbase-ce/
/home/admin/oceanbase-ce/
├── admin
├── bin
│   └── observer -> /home/admin/.obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc57d8a3f6
8b4c50f94aab80/bin/observer
├── etc
├── lib
│   ├── libaio.so -> /home/admin/.obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc57d8a3f
68b4c50f94aab80/lib/libaio.so
│   ├── libaio.so.1 -> /home/admin/.obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc57d8a
3f68b4c50f94aab80/lib/libaio.so.1
│   ├── libaio.so.1.0.1 -> /home/admin/.obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc5
7d8a3f68b4c50f94aab80/lib/libaio.so.1.0.1
│   ├── libmariadb.so -> /home/admin/.obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc57d
8a3f68b4c50f94aab80/lib/libmariadb.so
│   └── libmariadb.so.3 -> /home/admin/.obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc5
7d8a3f68b4c50f94aab80/lib/libmariadb.so.3
├── log
└── store -> /data
[admin@obce00 ~]$ tree /data
/data
├── clog -> /redo/clog
├── ilog -> /redo/ilog
```

```
├── slog -> /redo/slog
└── sstable

4 directories, 0 files
[admin@obce00 ~]$ tree /redo
/redo
├── clog
├── ilog
└── slog

3 directories, 0 files
```

启动和初始化集群

上文中 `deploy` 操作只安装了软件和准备初始化目录，还需使用 `obd cluster start` 命令启动集群节点并初始化集群。

说明

第一次运行 `start` 命令会对集群进行初始化（`boostrap`），以后再使用 `start` 命令就只会启动集群中节点进程。

```
obd cluster start obce-single

输出:
[admin@obce00 ~]$ obd cluster start obce-single
Get local repositories and plugins ok
Open ssh connection ok
Cluster param config check ok
Check before start observer ok
Start observer ok
observer program health check ok
Connect to observer ok
Initialize cluster
Cluster bootstrap ok
Wait for observer init ok
+-------------------------------------------------+
|                    observer                     |
+---------------+---------+------+-------+--------+
| ip            | version | port | zone  | status |
+---------------+---------+------+-------+--------+
| 172.20.249.50 | 3.1.0   | 2881 | zone1 | active |
+---------------+---------+------+-------+--------+

obce-single running
```

该命令进行初始化需要几分钟。

注意

当可用内存不足 8G 或者日志目录剩余可用空间比例不足 5% 时，`bootstrap` 很可能会失败。

确认集群是否初始化成功（可选）

首次学习或生产部署时，建议执行此操作。

- 查看启动后的集群状态

```
[admin@obce00 ~]$ obd cluster list
+-------------------------------------------------------------------------+
|                             Cluster List                                |
+-------------+---------------------------------------+-----------------+
| Name        | Configuration Path                    | Status (Cached) |
+-------------+---------------------------------------+-----------------+
| obce-single | /home/admin/.obd/cluster/obce-single | running         |
+-------------+---------------------------------------+-----------------+

[admin@obce00 ~]$ obd cluster display obce-single
Get local repositories and plugins ok
Open ssh connection ok
Cluster status check ok
Connect to observer ok
Wait for observer init ok
+-------------------------------------------------+
|                    observer                     |
+---------------+---------+------+-------+--------+
| ip            | version | port | zone  | status |
+---------------+---------+------+-------+--------+
| 172.20.249.50 | 3.1.0   | 2881 | zone1 | active |
+---------------+---------+------+-------+--------+
```

- 检查数据文件大小

  进程 observer 启动后会初始化数据文件 `block_file` 的大小，可根据参数 `datafile_size` 或 `datafile_disk_percentage` 进行控制。

```
[admin@obce00 ~]$ ls -lrth /data/sstable/block_file
-rw-r--r-- 1 admin admin 50G Sep 11 17:31 /data/sstable/block_file
```

- 检查进程

  OceanBase 数据库是单进程软件，进程名为 `observer`，可以使用下面命令查看这个进程。

```
[admin@obce00 ~]$ ps -ef | grep observer | grep -v grep
admin      30616       1 68 17:30 ?        00:02:54 /home/admin/oceanbase-ce/bin/observer -r 172
.20.249.50:2882:2881 -o __min_full_resource_pool_memory=268435456,redo_dir=/redo,memory_limit=8G
,system_memory=4G,stack_size=512K,cpu_count=16,cache_wash_threshold=1G,workers_per_cpu_quota=10,
schema_history_expire_time=1d,net_thread_count=4,major_freeze_duty_time=Disable,minor_freeze_tim
es=10,enable_separate_sys_clog=0,enable_merge_by_turn=False,datafile_size=50G,enable_syslog_wf=F
alse,enable_syslog_recycle=True,max_syslog_file_count=10,root_password=b******B -z zone1 -p 288
1 -P 2882 -c 1 -d /data -i eth0 -l WARN
[admin@obce00 ~]$
```

  从进程中可以看到，可执行文件是 `/home/admin/oceanbase-ce/bin/observer`，实际上它是个软链接。

```
[admin@obce00 oceanbase-ce]$ ll /home/admin/oceanbase-ce/bin/observer
lrwxrwxrwx 1 admin admin 100 Sep 11 17:16 /home/admin/oceanbase-ce/bin/observer -> /home/admin/.
obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc57d8a3f68b4c50f94aab80/bin/observer
```

进程启动时，通过 `-o` 指定了很多参数，这些参数均已是在前述 OBD 集群部署配置文件中指定。

- 检查进程监听端口

```
[admin@obce00 ~]$ sudo netstat -ntlp |grep observer
[sudo] password for admin:
tcp        0      0 0.0.0.0:2881            0.0.0.0:*               LISTEN      30616/observer
tcp        0      0 0.0.0.0:2882            0.0.0.0:*               LISTEN      30616/observer
```

连接 OceanBase 集群的内部实例（sys）

传统的 MySQL 客户端可以连接 OceanBase 社区版，前提是 MySQL 的版本是 5.5/5.6/5.7。OceanBase 数据库也提供自己的客户端工具 `obclient`，需要安装后才可使用。和传统 MySQL 不同，OBServer 的连接端口是 `2881`，连接用户名是 `root@sys`，密码需在上述的 OBD 配置文件里指定。

```
[admin@obce00 ~]$ mysql -h 172.20.249.50 -uroot@sys -P2881 -pb******B -c -A oceanbase
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 3221488586
Server version: 5.7.25 OceanBase 3.1.0 (r3-b20901e8c84d3ea774beeaca963c67d7802e4b4e) (Built Aug 10 202
1 08:10:38)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [oceanbase]> show databases;
+--------------------+
| Database           |
+--------------------+
| oceanbase          |
| information_schema |
| mysql              |
| SYS                |
| LBACSYS            |
| ORAAUDITOR         |
| test               |
+--------------------+
7 rows in set (0.002 sec)
```

在数据库列表里看到 `oceanbase` 数据库，表示集群初始化成功。

OBClient 安装和使用示例

```
sudo rpm -ivh /tmp/obd/obclient-*.rpm /tmp/obd/libobclient-*.rpm
```

```
obclient -h 172.20.249.50 -uroot@sys -P2881 -pb******B -c -A oceanbase
```

2.7 如何使用 OBD 自动化部署多节点集群

部署规划

本文介绍 OceanBase 三节点集群的部署方法，该部署方法需要通过中控机直接远程登录到 OceanBase 节点上部署启动 `observer` 和 `obproxy` 进程。

机器信息

机器类型	云主机 ECS
IP	172.20.249.50、172.20.249.52、172.20.249.49、172.20.249.51
网卡名	eth0
OS	CentOS Linux release 8.4.2105
CPU	4C
内存	总内存 14G，可用内存 11G
磁盘 1	云盘 /dev/vda 100G
磁盘 2	云盘 /dev/vdb 100G

机器和角色划分

角色	机器	备注
OBD	172.20.249.50	中控机，自动化部署软件
OBServer	172.20.249.52	OceanBase 数据库 zone1
OBServer	172.20.249.49	OceanBase 数据库 zone2
OBServer	172.20.249.51	OceanBase 数据库 zone3
OBProxy	172.20.249.52	OceanBase 访问反向代理
OBProxy	172.20.249.49	OceanBase 访问反向代理
OBProxy	172.20.249.51	OceanBase 访问反向代理
OBClient	172.20.249.50	OceanBase 命令行客户端

磁盘划分

此处使用 LVM 技术对 `/dev/vdb` 进行划分，需要登录到每个节点上手动初始化。

```
# lvm 分盘
pvcreate /dev/vdb
vgcreate obvg /dev/vdb
lvcreate obvg -L 20G^C
lvcreate -L 20G obvg -n lvredo
lvcreate -l 100%FREE obvg -n lvdata

# 格式化文件系统
mkfs.ext4 /dev/obvg/lvdata
mkfs.ext4 /dev/obvg/lvredo

# 修改 mount 参数文件
vim /etc/fstab
/dev/obvg/lvredo          /redo              ext4            defaults,noatime,nodiratime,nodelalloc,ba
rrier=0        0 0
/dev/obvg/lvdata             /data         ext4            defaults,noatime,nodiratime,nodelalloc,barr
ier=0        0 0

# 挂载文件系统
mkdir -p /data /redo
vim /etc/fstab
mount /data
mount /redo
chown -R admin.admin /data /redo

# 检查
df -h

输出:

文件系统                  容量  已用  可用 已用% 挂载点
/dev/mapper/obvg-lvdata   59G   53M   56G    1% /data
/dev/mapper/obvg-lvredo   20G   45M   19G    1% /redo
```

编辑 OBD 配置文件

OBD 针对不同的部署场景提供不同的配置文件。这些配置文件示例放在 OceanBase 开源项目地址中，您可访问链接查看：https://github.com/oceanbase/obdeploy/tree/master/example。

如果您是部署三节点版本，只需下载如下两个配置文件：

- 部署三节点 `observer` 进程：

  https://github.com/oceanbase/obdeploy/blob/master/example/mini-distributed-example.yaml

- 部署三节点 `observer` 和 `obproxy` 进程：

  https://github.com/oceanbase/obdeploy/blob/master/example/mini-distributed-with-obproxy-example.yaml

此处仿照生产环境，选择第二种部署配置文件下载。

```
[admin@obce00 ~]$ cat obce-3zones.yaml

#  Only need to configure when remote login is required

user:
   username: admin
#    password: your password if need
   key_file: /home/admin/.ssh/id_rsa.pub
   port: your ssh port, default 22
#    timeout: ssh connection timeout (second), default 30
oceanbase-ce:
  servers:
    - name: obce01
      # Please don't use hostname, only IP can be supported
      ip: 172.20.249.52
    - name: obce02
      ip: 172.20.249.49
    - name: obce03
      ip: 172.20.249.51
  global:
    # Please set devname as the network adaptor's name whose ip is  in the setting of severs.
    # if set severs as "127.0.0.1", please set devname as "lo"
    # if current ip is 192.168.1.10, and the ip's network adaptor's name is "eth0", please use "eth0"
    devname: eth0
    cluster_id: 2
    # please set memory limit to a suitable value which is matching resource.
    memory_limit: 8G # The maximum running memory for an observer
    system_memory: 3G # The reserved system memory. system_memory is reserved for general tenants. Th
e default value is 30G.
    stack_size: 512K
    cpu_count: 16
    cache_wash_threshold: 1G
    __min_full_resource_pool_memory: 268435456
    workers_per_cpu_quota: 10
    schema_history_expire_time: 1d
    # The value of net_thread_count had better be same as cpu's core number.
    net_thread_count: 4
    major_freeze_duty_time: Disable
    minor_freeze_times: 10
    enable_separate_sys_clog: 0
    enable_merge_by_turn: FALSE
    #datafile_disk_percentage: 20 # The percentage of the data_dir space to the total disk space. Thi
s value takes effect only when datafile_size is 0. The default value is 90.
    datafile_size: 50G
    syslog_level: WARN # System log level. The default value is INFO.
    enable_syslog_wf: false # Print system logs whose levels are higher than WARNING to a separate lo
g file. The default value is true.
    enable_syslog_recycle: true # Enable auto system log recycling or not. The default value is false.
    max_syslog_file_count: 10 # The maximum number of reserved log files before enabling auto recyclin
g. The default value is 0.
    # observer cluster name, consistent with obproxy's cluster_name
    appname: obce-3zones
    root_password: 0E****8d # root user password, can be empty
    proxyro_password: uY****zx # proxyro user pasword, consistent with obproxy's observer_sys_password
, can be empty
  obce01:
    mysql_port: 2881 # External port for OceanBase Database. The default value is 2881.
    rpc_port: 2882 # Internal port for OceanBase Database. The default value is 2882.
```

```
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo
    zone: zone1
  obce02:
    mysql_port: 2881 # External port for OceanBase Database. The default value is 2881.
    rpc_port: 2882 # Internal port for OceanBase Database. The default value is 2882.
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo
    zone: zone2
  obce03:
    mysql_port: 2881 # External port for OceanBase Database. The default value is 2881.
    rpc_port: 2882 # Internal port for OceanBase Database. The default value is 2882.
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo
    zone: zone3
obproxy:
  servers:
    - 172.20.249.52
    - 172.20.249.49
    - 172.20.249.51
  # Set dependent components for the component.
  # When the associated configurations are not done, OBD will automatically get the these configuratio
ns from the dependent components.
  depends:
    - oceanbase-ce
  global:
    listen_port: 2883 # External port. The default value is 2883.
    prometheus_listen_port: 2884 # The Prometheus port. The default value is 2884.
    home_path: /home/admin/obproxy
    # oceanbase root server list
    # format: ip:mysql_port;ip:mysql_port
    rs_list: 172.20.249.52:2881;172.20.249.49:2881;172.20.249.51:2881
    enable_cluster_checkout: false
    # observer cluster name, consistent with oceanbase-ce's appname
    cluster_name: obce-3zones
    obproxy_sys_password: 0M****tm # obproxy sys user password, can be empty
    observer_sys_password: uY****zx # proxyro user pasword, consistent with oceanbase-ce's proxyro_pas
sword, can be empty
```

该配置文件是专门针对最小内存（可用内存大于 8G）的节点配置，里面指定了很多 observer 进程的启动参数。您需注意 `yaml` 的格式，每个配置项后面冒号（`:`）跟后面的值之间必须有个空格（`' '`）。 下面对关键的几个参数进

行补充说明下:

配置类	配置项名	配置值	备注
user	username	admin	中控机连接 OceanBase 节点的用户名，也是 OceanBase 要部署的用户名。
user	key_file	/home/admin/.ssh/id_rsa.pub	中控机上 SSH 用的 RSA 公钥。
user	port	22	OceanBase 集群节点的 SSH 端口，默认是 22。如果您使用的端口不是 22 则需要修改该配置值。
oceanbase-ce	servers	指定所有机器列表	每个机器是用 - name机器标识名(换行)ip:机器ip 指定。多个机器就指定多次，以后会进行优化。
oceanbase-ce	home_path	/home/admin/oceanbase-ce	指定到普通用户（admin）的目录下，为区别于企业版，文件名叫 oceanbase-ce 。
oceanbase-ce	data_dir	/data	指向独立的磁盘，这里使用前面分配的 LV（lvdata）。实际存储 OceanBase 的数据文件（block_file）。
oceanbase-ce	redo_dir	/redo	指向独立的磁盘，这里使用前面分配的 LV（lvredo）。实际存储 OceanBase 的事务日志、sstable 日志等。
oceanbase-ce	devname	eth0	和 servers 里指定的 IP 对应的网卡。如果前面 IP 是127.0.0.1，这里就填 lo。 通过 ip addr 命令可以查看 IP 和网卡对应关系。
oceanbase-ce	mysql_port	2881	进程 observer 的连接端口，默认是 2881。后面 OceanBase 客户端直连这个端口可以访问该节点。
oceanbase-ce	rpc_port	2882	进程 observer 跟其他节点进程之间的 RPC 通信端口，默认是 2882。
oceanbase-	zone	zone1 zone2 zone3	zone 是逻辑机房的概念。三副本集群下有三个 zone 。

ce			
oceanbase-ce	cluster_id	2	OceanBase 集群 ID 标识，不同集群不要重复即可。
oceanbase-ce	memory_limit	8G	进程 observer 能从 OS 获取的最大内存，最小不少于 8G 。如果机器内存丰富的话，这个参数可以大一些。
oceanbase-ce	system_memory	3G	进程 observer 留给集群内部用的保留内存，这个会占用上面 memory_limit 的内存，留给业务租户的就更少。
oceanbase-ce	datafile_size	datafile_disk_percentage	
oceanbase-ce	syslog_level	WARN 或 ERROR	运行日志的日志级别，有 INFO。
oceanbase-ce	enable_syslog_recycle	TRUE	指定运行日志是否以滚动方式输出，最多保留指定数量的运行日志。
oceanbase-ce	max_syslog_file_count	10	根据磁盘空间大小定，这里默认保留最多 10 个历史运行日志文件。
oceanbase-ce	root_password	随机字符串	OceanBase 集群的超级管理员root@sys 的密码，建议设置复杂的密码。
oceanbase-ce	proxyro_password	随机字符串	OBProxy 连接 OceanBase 集群使用的账户名（proxyro）的密码。
obproxy	servers	任意机器IP	OBProxy 可以部署在应用服务器、中控机或者 OceanBase 机器上。这里选择 OceanBase 机器。
obproxy	depends	依赖的配置节	通常指定依赖的集群配置，会自动复用集群的 proxyro 密码、集群名 cluster_name、rs_list 等等。

obproxy	listen_port	2883	OBProxy 监听端口，默认2883。
obproxy	prometheus_listen_port	2884	prometheus 监听端口，默认2884。
obproxy	home_path	/home/admin/obproxy	OBProxy 默认安装路径，建议在普通用户 admin 下。
obproxy	rs_list	172.20.249.52:2881;172.20.249.49:2881;172.20.249.51:2881	OceanBase 集群 rootservice 服务地址，由 sys 租户的三副本所在节点 IP 组成。 可以手动指定，也可以不指定，依赖前面的depends配置节自动从 OceanBase 集群配置里获取。
obproxy	enable_cluster_checkout	FALSE	
obproxy	cluster_name	obce-3zones	OceanBase 集群名字。
obproxy	obproxy_sys_password	随机字符串	OBProxy 管理员账户（proxysys）的密码。
obproxy	observer_sys_password	跟 proxyro_password 一致	OBProxy 连接 OceanBase 集群使用的账户名（proxyro）的密码。

部署成功后，OBD 会将配置文件 obce-3zones.yaml 复制到自己的工作目录里（~/.obd/cluster/obce-3zones/config.yaml），后期再对 obce-3zones.yaml 文件进行修改是不生效的。

OBD 部署集群

配置文件准备好后，就可以部署该配置文件对应的集群，部署内容主要包含:

- 复制软件到相应节点，并安装软件。
- 在相应节点创建相关目录。

您可使用命令 obd cluster deploy [集群名] -c集群配置文件 进行部署。

```
[admin@obce00 ~]$ obd cluster deploy obce-3zones -c obce-3zones.yaml
```

```
oceanbase-ce-3.1.0 already installed.
obproxy-3.1.0 already installed.
+-----------------------------------------------------------------------------+
|                                  Packages                                   |
+--------------+---------+---------+------------------------------------------+
| Repository   | Version | Release | Md5                                      |
+--------------+---------+---------+------------------------------------------+
| oceanbase-ce | 3.1.0   | 3.el8   | 84bd2fe27f8b8243cc57d8a3f68b4c50f94aab80 |
| obproxy      | 3.1.0   | 1.el8   | d242ea5fe45222b8f61c3135ba2aaa778c61ea22 |
+--------------+---------+---------+------------------------------------------+
Repository integrity check ok
Parameter check ok
Open ssh connection ok
Remote oceanbase-ce-3.1.0-84bd2fe27f8b8243cc57d8a3f68b4c50f94aab80 repository install ok
Remote oceanbase-ce-3.1.0-84bd2fe27f8b8243cc57d8a3f68b4c50f94aab80 repository lib check ok
Remote obproxy-3.1.0-d242ea5fe45222b8f61c3135ba2aaa778c61ea22 repository install ok
Remote obproxy-3.1.0-d242ea5fe45222b8f61c3135ba2aaa778c61ea22 repository lib check ok
Cluster status check ok
Initializes cluster work home ok
Initializes cluster work home ok
obce-3zones deployed
```

检查集群部署状态。

```
[admin@obce00 ~]$ obd cluster list
+-------------------------------------------------------------------------+
|                             Cluster List                                |
+-------------+---------------------------------------+-----------------+
| Name        | Configuration Path                    | Status (Cached) |
+-------------+---------------------------------------+-----------------+
| obce-3zones | /home/admin/.obd/cluster/obce-3zones | deployed        |
+-------------+---------------------------------------+-----------------+
```

OBD 启动和初始化集群

上面 `deploy` 操作只是安装了软件和准备初始化目录，您还需使用命令 `obd cluster start` 启动集群节点并初始化集群。

```
obd cluster start obce-3zones
```

输出:

```
[admin@obce00 ~]$ obd cluster start obce-3zones
Get local repositories and plugins ok
Open ssh connection ok
Cluster param config check ok
Check before start observer ok
[WARN] (172.20.249.52) The recommended value of fs.aio-max-nr is 1048576 (Current value: 65536)
[WARN] (172.20.249.52) The recommended number of open files is 655350 (Current value: 65535)
[WARN] (172.20.249.49) The recommended value of fs.aio-max-nr is 1048576 (Current value: 65536)
[WARN] (172.20.249.49) The recommended number of open files is 655350 (Current value: 65535)
[WARN] (172.20.249.51) The recommended value of fs.aio-max-nr is 1048576 (Current value: 65536)
[WARN] (172.20.249.51) The recommended number of open files is 655350 (Current value: 65535)
```

```
Check before start obproxy ok
Start observer ok
observer program health check ok
Connect to observer ok
Initialize cluster
Cluster bootstrap ok
Wait for observer init ok
+-------------------------------------------------+
|                    observer                     |
+---------------+---------+------+-------+--------+
| ip            | version | port | zone  | status |
+---------------+---------+------+-------+--------+
| 172.20.249.49 | 3.1.0   | 2881 | zone2 | active |
| 172.20.249.51 | 3.1.0   | 2881 | zone3 | active |
| 172.20.249.52 | 3.1.0   | 2881 | zone1 | active |
+---------------+---------+------+-------+--------+

Start obproxy ok
obproxy program health check ok
Connect to obproxy ok
Initialize cluster
+-------------------------------------------------+
|                    obproxy                      |
+---------------+------+-----------------+--------+
| ip            | port | prometheus_port | status |
+---------------+------+-----------------+--------+
| 172.20.249.52 | 2883 | 2884            | active |
| 172.20.249.49 | 2883 | 2884            | active |
| 172.20.249.51 | 2883 | 2884            | active |
+---------------+------+-----------------+--------+
obce-3zones running
```

如果集群节点内核参数和会话限制参数不符合要求，安装时会给出提示。

这个命令会进行几分钟的 `bootstrap`。当可用内存不足 8G 或者日志目录剩余可用空间比例不足 5% 的时候，这个 `bootstrap` 是很可能会失败的。

确认集群是否初始化成功（可选）

这个步骤是可选的。第一次学习或生产部署时，建议检查一下。

- 查看启动后的集群状态

```
[admin@obce00 ~]$ obd cluster list
+-------------------------------------------------------------------------+
|                              Cluster List                               |
+-------------+---------------------------------------+-----------------+
| Name        | Configuration Path                    | Status (Cached) |
+-------------+---------------------------------------+-----------------+
| obce-3zones | /home/admin/.obd/cluster/obce-3zones | running         |
+-------------+---------------------------------------+-----------------+
```

- 检查 OceanBase 集群各个节点进程信息

  OceanBase 数据库是单进程软件，进程名为 `observer`，可运行以下命令查看这个进程。

```
IPS="172.20.249.52 172.20.249.49 172.20.249.51"
for ob in $IPS;do echo $ob; ssh $ob "ps -ef | grep observer | grep -v grep "; done
```

  输出:

```
[admin@obce00 oceanbase-ce]$ for ob in $IPS;do echo $ob; ssh $ob "ps -ef | grep observer | grep
-v grep "; done
172.20.249.52
admin       6987       1 69 08:35 ?        01:38:26 /home/admin/oceanbase-ce/bin/observer -r 172
.20.249.52:2882:2881;172.20.249.49:2882:2881;172.20.249.51:2882:2881 -o __min_full_resource_pool
_memory=268435456,memory_limit=8G,system_memory=3G,stack_size=512K,cpu_count=16,cache_wash_thres
hold=1G,workers_per_cpu_quota=10,schema_history_expire_time=1d,net_thread_count=4,major_freeze_d
uty_time=Disable,minor_freeze_times=10,enable_separate_sys_clog=0,enable_merge_by_turn=False,dat
afile_size=50G,enable_syslog_wf=False,enable_syslog_recycle=True,max_syslog_file_count=10,root_p
assword=0E****8d,redo_dir=/redo -z zone1 -p 2881 -P 2882 -n obce-3zones -c 2 -d /data -i eth0 -
l WARN
172.20.249.49
admin       7064       1 87 08:35 ?        02:02:59 /home/admin/oceanbase-ce/bin/observer -r 172
.20.249.52:2882:2881;172.20.249.49:2882:2881;172.20.249.51:2882:2881 -o __min_full_resource_pool
_memory=268435456,memory_limit=8G,system_memory=3G,stack_size=512K,cpu_count=16,cache_wash_thres
hold=1G,workers_per_cpu_quota=10,schema_history_expire_time=1d,net_thread_count=4,major_freeze_d
uty_time=Disable,minor_freeze_times=10,enable_separate_sys_clog=0,enable_merge_by_turn=False,dat
afile_size=50G,enable_syslog_wf=False,enable_syslog_recycle=True,max_syslog_file_count=10,root_p
assword=0E****8d,redo_dir=/redo -z zone2 -p 2881 -P 2882 -n obce-3zones -c 2 -d /data -i eth0 -
l WARN
172.20.249.51
admin       6920       1 72 08:35 ?        01:42:42 /home/admin/oceanbase-ce/bin/observer -r 172
.20.249.52:2882:2881;172.20.249.49:2882:2881;172.20.249.51:2882:2881 -o __min_full_resource_pool
_memory=268435456,memory_limit=8G,system_memory=3G,stack_size=512K,cpu_count=16,cache_wash_thres
hold=1G,workers_per_cpu_quota=10,schema_history_expire_time=1d,net_thread_count=4,major_freeze_d
uty_time=Disable,minor_freeze_times=10,enable_separate_sys_clog=0,enable_merge_by_turn=False,dat
afile_size=50G,enable_syslog_wf=False,enable_syslog_recycle=True,max_syslog_file_count=10,root_p
assword=0E****8d,redo_dir=/redo -z zone3 -p 2881 -P 2882 -n obce-3zones -c 2 -d /data -i eth0 -
l WARN
```

  从进程里看，可执行文件是 `/home/admin/oceanbase-ce/bin/observer`，实际上它是个软链接。

```
[admin@obce00 oceanbase-ce]$ ll /home/admin/oceanbase-ce/bin/observer
lrwxrwxrwx 1 admin admin 100 Sep 11 17:16 /home/admin/oceanbase-ce/bin/observer -> /home/admin/.
obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc57d8a3f68b4c50f94aab80/bin/observer
```

  进程启动的时候，通过 `-o` 指定了很多参数，这些参数都是在前面 OBD 集群部署配置文件里指定的。

- 检查 OceanBase 集群各个节点监听状况

```
IPS="172.20.249.52 172.20.249.49 172.20.249.51"
for ob in $IPS;do echo $ob; ssh $ob "netstat -ntlp"; done

输出:
[admin@obce00 ~]$ for ob in $IPS;do echo $ob; ssh $ob "netstat -ntlp"; done
172.20.249.52
(Not all processes could be identified, non-owned process info
```

```
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:2881            0.0.0.0:*               LISTEN      6987/observer
tcp        0      0 0.0.0.0:2882            0.0.0.0:*               LISTEN      6987/observer
tcp        0      0 0.0.0.0:2883            0.0.0.0:*               LISTEN      7640/obproxy
tcp        0      0 0.0.0.0:2884            0.0.0.0:*               LISTEN      7640/obproxy
172.20.249.49
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:2881            0.0.0.0:*               LISTEN      7064/observer
tcp        0      0 0.0.0.0:2882            0.0.0.0:*               LISTEN      7064/observer
tcp        0      0 0.0.0.0:2883            0.0.0.0:*               LISTEN      7718/obproxy
tcp        0      0 0.0.0.0:2884            0.0.0.0:*               LISTEN      7718/obproxy
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
172.20.249.51
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:2881            0.0.0.0:*               LISTEN      6920/observer
tcp        0      0 0.0.0.0:2882            0.0.0.0:*               LISTEN      6920/observer
tcp        0      0 0.0.0.0:2883            0.0.0.0:*               LISTEN      7574/obproxy
tcp        0      0 0.0.0.0:2884            0.0.0.0:*               LISTEN      7574/obproxy
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -


<p data-line="407" class="sync-line" style="margin:0;"></p>
```

连接 OceanBase 集群的内部实例（sys)

传统的 MySQL 客户端可以连接 OceanBase 社区版，前提是 MySQL 的版本是 5.5/5.6/5.7。OceanBase 数据库也提供自己的客户端工具 `OBClient`，需要安装后才可使用。 和传统MySQL 不同的是 OBPROXY 的连接端口是 `2883`， 连接用户名为 `root@sys#集群名`，密码则在前面 OBD 配置文件里指定。

```
mysql -h 172.20.249.52 -uroot@sys#obce-3zones -P2883 -p0E****8d -c -A oceanbase

输出:
[admin@obce00 ~]$ mysql -h 172.20.249.52 -uroot@sys#obce-3zones -P2883 -p0E****8d -c -A oceanbase
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.6.25 OceanBase 3.1.0 (r3-b20901e8c84d3ea774beeaca963c67d7802e4b4e) (Built Aug 10 202
1 08:10:38)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [oceanbase]> show databases;
+--------------------+
```

```
| Database           |
+--------------------+
| oceanbase          |
| information_schema |
| mysql              |
| SYS                |
| LBACSYS            |
| ORAAUDITOR         |
| test               |
+--------------------+
7 rows in set (0.002 sec)

MySQL [oceanbase]> select a.zone,concat(a.svr_ip,':',a.svr_port) observer, cpu_total, (cpu_total-cpu_a
ssigned) cpu_free, round(mem_total/1024/1024/1024) mem_total_gb, round((mem_total-mem_assigned)/1024/1
024/1024) mem_free_gb, usec_to_time(b.last_offline_time) last_offline_time, usec_to_time(b.start_servi
ce_time) start_service_time, b.status, usec_to_time(b.stop_time) stop_time, b.build_version
from __all_virtual_server_stat a join __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_port)
order by a.zone, a.svr_ip
;

+-------+--------------------+-----------+----------+--------------+-------------+--------------------
--------+----------------------------+--------+----------------------------+-------------------------
-----------------------------------------------+
| zone  | observer           | cpu_total | cpu_free | mem_total_gb | mem_free_gb | last_offline_tim
e          | start_service_time         | status | stop_time                  | build_versio
n                                                                |
+-------+--------------------+-----------+----------+--------------+-------------+--------------------
--------+----------------------------+--------+----------------------------+-------------------------
-----------------------------------------------+
| zone1 | 172.20.249.52:2882 |        14 |     11.5 |            5 |           4 | 1970-01-01 08:00:00
.000000 | 2021-09-12 08:36:06.357140 | active | 1970-01-01 08:00:00.000000 | 3.1.0_3-b20901e8c84d3ea77
4beeaca963c67d7802e4b4e(Aug 10 2021 08:10:38) |
| zone2 | 172.20.249.49:2882 |        14 |     11.5 |            5 |           4 | 1970-01-01 08:00:00
.000000 | 2021-09-12 08:36:07.605244 | active | 1970-01-01 08:00:00.000000 | 3.1.0_3-b20901e8c84d3ea77
4beeaca963c67d7802e4b4e(Aug 10 2021 08:10:38) |
| zone3 | 172.20.249.51:2882 |        14 |     11.5 |            5 |           4 | 1970-01-01 08:00:00
.000000 | 2021-09-12 08:36:07.631981 | active | 1970-01-01 08:00:00.000000 | 3.1.0_3-b20901e8c84d3ea77
4beeaca963c67d7802e4b4e(Aug 10 2021 08:10:38) |
+-------+--------------------+-----------+----------+--------------+-------------+--------------------
--------+----------------------------+--------+----------------------------+-------------------------
-----------------------------------------------+
3 rows in set (0.004 sec)
```

当在数据库列表里看到 `oceanbase` 数据库时就表示集群初始化成功。

OBClient 安装和使用示例

```
sudo rpm -ivh /tmp/obd/obclient-*.rpm /tmp/obd/libobclient-*.rpm

obclient -h 172.20.249.52 -uroot@sys#obce-3zones -P2883 -p0E****8d -c -A oceanbase
```

2.8 如何查看和修改 OceanBase 集群参数

OceanBase 数据库以集群形态运行，提供多租户（也叫多实例）能力。集群初始化成功后，默认会有一个 `sys` 租户，用以保存集群的所有元数据、参数等。您也可通过登录 `sys` 租户管理 OceanBase 集群。

查看和修改 OceanBase 集群参数

您可通过命令 `show parameters [ like '%参数名特征%' ] ;` 或 `show parameters where name in ( '参数名1' , '参数名2' ) ;` 查看 OceanBase 集群参数。

在命令 `show parameters [ like '%参数名特征%' ] ;` 中不带 `like` 子句表示查看所有参数。

现在以查看参数 `memory_limit` 和 `memory_limit_percentage` 为例进行讲解。

首先这两个参数是指定进程 observer 启动后能获取的最大内存，如果分配不出来进程可能会启动失败或运行异常。这个内存可以指定大小，也可以指定总可用内存的比例。无论使用哪种方法，都需要确保实际可以使用的内存不少于 `8G`。

这两个参数实际只有一个生效，取两个参数中的最低值。`memory_limit` 设置为 `0` 时就表示不限制。使用哪个参数控制进程 observer 内存大小由运维人员决定。生产环境中，机器内存很大的时候，通常是通过 `memory_limit_percentage` 控制，默认值是 `80`（表示总可用内存的 80%）。

```
MySQL [oceanbase]> show parameters like 'memory_limit%';
+-------+----------+---------------+----------+-------------------------+-----------+-------+---
| zone  | svr_type | svr_ip        | svr_port | name                    | data_type | value | in
+-------+----------+---------------+----------+-------------------------+-----------+-------+---
| zone1 | observer | 172.20.249.50 |     2882 | memory_limit_percentage | NULL      | 80    | th
| zone1 | observer | 172.20.249.50 |     2882 | memory_limit            | NULL      | 8G    | th
+-------+----------+---------------+----------+-------------------------+-----------+-------+---
2 rows in set (0.002 sec)

MySQL [oceanbase]> show parameters where name in ('memory_limit','memory_limit_percentage')\G
*************************** 1. row ***************************
      zone: zone1
  svr_type: observer
    svr_ip: 172.20.249.50
  svr_port: 2882
      name: memory_limit_percentage
 data_type: NULL
     value: 80
      info: the size of the memory reserved for internal use(for testing purpose). Range: [10, 9
   section: OBSERVER
     scope: CLUSTER
    source: DEFAULT
edit_level: DYNAMIC_EFFECTIVE
*************************** 2. row ***************************
      zone: zone1
  svr_type: observer
    svr_ip: 172.20.249.50
  svr_port: 2882
```

```
      name: memory_limit
 data_type: NULL
     value: 8G
      info: the size of the memory reserved for internal use(for testing purpose), 0 means follo
   section: OBSERVER
     scope: CLUSTER
    source: DEFAULT
edit_level: DYNAMIC_EFFECTIVE
2 rows in set (0.002 sec)0
```

上述参数输出结果说明如下:

列名	列值	备注
zone	zone1	节点的 zone 名称
svr_type	observer	节点类型
svr_ip	172.20.249.50	节点 IP
svr_port	2882	节点 RPC 端口
name	memory_limit_percentage	参数名
data_type	NULL	参数类型
value	80	参数值
info	the size of the memory reserved for internal use(for testing purpose). Range [10, 90]	参数的描述。 该参数的这个描述不是很准确，这是限制进程 observer 能分配的最大内存
section	OBSERVER	参数归类
scope	CLUSTER	参数生效范围
edit_level	DYNAMIC_EFFECTIVE	参数生效时机：动态生效 / 需要重启

OceanBase 集群参数可通过命令 `alter system set参数名='参数值' [ server = '节点IP:节点RPC端口' ] ;` 进行修改。不指定 `server` 子句表示参数修改应用于所有 OceanBase 集群节点。

示例：调整参数 `syslog_level` 值为 `USER_ERROR`。

```
MySQL [oceanbase]> alter system set syslog_level = 'USER_ERR' server='172.20.249.50:2882' ;
Query OK, 0 rows affected (0.021 sec)
```

```
MySQL [oceanbase]> show parameters like 'syslog_level'\G
*************************** 1. row ***************************
      zone: zone1
  svr_type: observer
    svr_ip: 172.20.249.50
  svr_port: 2882
      name: syslog_level
 data_type: NULL
     value: USER_ERR
      info: specifies the current level of logging. There are DEBUG, TRACE, INFO, WARN, USER_ERR
   section: OBSERVER
     scope: CLUSTER
    source: DEFAULT
edit_level: DYNAMIC_EFFECTIVE
1 row in set (0.002 sec)
```

OceanBase 集群参数文件

上述参数的修改都是立即生效，并且参数修改会持久化到 OceanBase 集群节点自己的参数文件。

注意 此处的 OceanBase 集群节点自己的参数文件，不是指前面提到的 OBD 集群部署参数文件。

通常 OceanBase 集群每个节点的启动目录下都会有一个目录 `etc`，保存了该节点进程的参数文件 `observer.config.bin`。`observer.config.bin` 是一个 binary 类型的文件，不能直接用 `cat` 命令读取，您可使用 `strings` 命令读取。该文件也不建议直接修改，您可通过上面提到的命令进行修改。

```
[admin@obce00 oceanbase-ce]$ pwd
/home/admin/oceanbase-ce
[admin@obce00 oceanbase-ce]$ tree -L 2
.

├── bin
│   └── observer -> /home/admin/.obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc57d8a3f68b4c50f
94aab80/bin/observer
├── etc
│   ├── observer.config.bin
│   └── observer.config.bin.history
├── etc2
│   ├── observer.conf.bin
│   └── observer.conf.bin.history
├── etc3
│   ├── observer.conf.bin
│   └── observer.conf.bin.history

<省略掉无关内容>

9 directories, 20 files
```

从上述目录结构可得，启动目录下有三个文件夹：`etc`、`etc2`、`etc3`，每个文件夹下都有其参数文件及其历史文件

备份。

`observer` 进程默认会读取文件夹 `etc` 中的参数文件，其他两个目录是参数文件的备份，这个备份路径也是通过参数 `config_additional_dir` 指定的，默认值是同一个启动目录的 `etc2` 和 `etc3`。

生产环境一般会将 `etc` 设置到其他磁盘，这样会更加安全。当前 OBD 版本还是把它放到同一块盘，但参数文件需要放到哪里并没有具体规定，您可自行修改。

说明 `etc2` 和 `etc3` 下的参数文件名跟 `etc` 下参数文件名并不完全一致。

```
MySQL [oceanbase]> show parameters like 'config_additional_dir'\G
*************************** 1. row ***************************
      zone: zone1
  svr_type: observer
    svr_ip: 172.20.249.50
  svr_port: 2882
      name: config_additional_dir
 data_type: NULL
     value: etc2;etc3
      info: additional directories of configure file
   section: OBSERVER
     scope: CLUSTER
    source: DEFAULT
edit_level: DYNAMIC_EFFECTIVE
1 row in set (0.002 sec)

[admin@obce00 oceanbase-ce]$ strings etc/observer.config.bin | grep -n memory_limit
25:memory_limit=8G
[admin@obce00 oceanbase-ce]$ strings etc2/observer.conf.bin | grep -n memory_limit
25:memory_limit=8G
[admin@obce00 oceanbase-ce]$ strings etc3/observer.conf.bin | grep -n memory_limit
25:memory_limit=8G
```

查看实际参数文件内容，可以看出不是所有参数都在这个参数文件中。只有那些被 `alter system set` 命令修改过的参数，以及在进程 `observer` 启动时通过 `-o` 指定的参数才会记录在参数文件里。其他参数都是取默认值（写在进程 `observer` 的代码里）。

使用 OBD 修改 OceanBase 集群参数

上文中直接在 OceanBase 集群里修改参数后，会立即同步到集群节点自身的参数文件中，但是不会同步到 OBD 的集群部署配置文件中（后期 OBD 可能会改进这个功能）。

所以，在您使用 OBD 工具重启 OceanBase 集群时，默认又会带参数启动进程 observer。如果前面在 OceanBase 集群里修改的参数在 OBD 集群部署配置文件中也有，并且 OBD 集群部署配置文件的值还是未经修改的，那就意味着修改过的参数又被调整回原来的设置值了（运维需要理解这里变化的原理）。

针对这个问题，OBD 提供两个解决思路：

- 手动同步修改 OBD 集群部署配置文件中的参数值（以后工具可能会自动同步）。

- OBD 重启集群的时候不带参数启动节点进程。

您可使用命令 `obd cluster edit-config` 编辑集群部署配置文件，退出时会保存到上文的工作目录中。

```
obd cluster edit-config obce-single

保存时输出:
oceanbase-ce-3.1.0 already installed.
Search param plugin and load ok
Parameter check ok
Save deploy "obce-single" configuration
deploy "need reload"
```

使用命令 `edit-config` 退出后会提示 reload 集群配置。

```
[admin@obce00 ~]$ obd cluster reload obce-single
Get local repositories and plugins ok
Open ssh connection ok
Cluster status check ok
Connect to observer ok
obce-single reload
```

说明 如果 OBD 命令运行出错，可以运行命令 `tail -n 50 ~/.obd/log/obd` 查看日志。

进程启动时指定参数

前面说到 OBD 在启动集群节点进程 observer 时，会在命令行下通过 `-o` 指定参数。对于运维来说，如果某个节点的进程 `observer` 因为某种原因退出，启动进程是当务之急。可能需要调整某个参数再启动一次，通过 OBD 工具会导致效率低下。所以，掌握 OceanBase 集群节点进程 `observer` 的启动方法是很有必要的。

首先进入到工作目录。必须在上一次启动 `observer` 进程的工作目录（假设它是正确的）下再次尝试。前面分析过，工作目录在 OBD 集群部署配置文件中指定 `home_path`。本教程里工作目录都默认是 `/home/admin/oceanbase-ce`。进程 `observer` 启动后会在这个目录找目录 `etc`，找默认的参数文件 `observer.config.bin`。启动后的日志会默认写到 `log/{observer.log, rootservice.log, election.log}`。所以，工作目录不能错，目录的权限也不能错。

示例：不带参数启动进程 `observer` 。为了模拟故障，先强行杀掉进程 `observer`。

```
[admin@obce00 ~]$ cd
[admin@obce00 ~]$ cd oceanbase-ce/
[admin@obce00 oceanbase-ce]$ kill -9 `pidof observer`
[admin@obce00 oceanbase-ce]$ sleep 3
[admin@obce00 oceanbase-ce]$ ps -ef|grep observer
admin      35278   28904  0 11:26 pts/2    00:00:00 grep --color=auto observer
[admin@obce00 oceanbase-ce]$ pwd
/home/admin/oceanbase-ce
[admin@obce00 oceanbase-ce]$ bin/observer
bin/observer
[admin@obce00 oceanbase-ce]$ ps -ef|grep observer
admin      35280       1 99 11:26 ?        00:00:06 bin/observer
```

```
admin      35848   28904  0 11:26 pts/2    00:00:00 grep --color=auto observer
[admin@obce00 oceanbase-ce]$ netstat -ntlp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:2881            0.0.0.0:*               LISTEN      35280/bin/observer
tcp        0      0 0.0.0.0:2882            0.0.0.0:*               LISTEN      35280/bin/observer
```

示例：带参数启动进程 `observer`。为了模拟故障，先强行杀掉进程 `observer`。

```
[admin@obce00 oceanbase-ce]$ kill -9 `pidof observer`
[admin@obce00 oceanbase-ce]$ sleep 3
[admin@obce00 oceanbase-ce]$ bin/observer -o "max_syslog_file_count=15,datafile_size=60G"
bin/observer -o max_syslog_file_count=15,datafile_size=60G
optstr: max_syslog_file_count=15,datafile_size=60G
[admin@obce00 oceanbase-ce]$ ps -ef|grep observer
admin      35867       1 99 11:34 ?        00:00:09 bin/observer -o max_syslog_file_count=15,datafile_s
ize=60G
admin      36435   28904  0 11:34 pts/2    00:00:00 grep --color=auto observer
[admin@obce00 oceanbase-ce]$ netstat -ntlp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:2881            0.0.0.0:*               LISTEN      35867/bin/observer
tcp        0      0 0.0.0.0:2882            0.0.0.0:*               LISTEN      35867/bin/observer
```

2.9 如何部署 OBAgent

OBAgent 简介

OBAgent 是采用 GO 语言开发的监控采集框架，通常部署在 OBServer 节点上。OBAgent 支持推、拉两种数据采集模式，可以满足不同的应用场景。

OBAgent 默认支持的插件包括主机数据采集、OceanBase 数据库指标的采集、监控数据标签处理和 Prometheus 协议的 HTTP 服务。如果想让 OBAgent 支持其他数据源的采集，或者自定义数据的处理流程，您只需要开发对应的插件即可。

编辑 OBAgent 部署配置文件

OBAgent 部署配置文件可以跟 OceanBase 集群部署配置文件部署在一起，也可以后期单独部署。本节附录 A.1 示例了同时部署 OceanBase 集群和 OBAgent。

下面示例是采用单独的配置文件部署 OBAgent。编辑 OBAgent 的部署配置文件跟编辑 OceanBase 集群部署配置文件类似。

1. 指定部署节点，包括节点名称和 IP。节点名称保持唯一，可以是主机名（假设主机名是唯一的）。
2. 指定全局配置。各个节点共同的配置都放在 `global` 节下，节点定制化的配置则不用放在 `global` 节下。
3. 指定各个节点定制化的配置。比如说每个节点的 `zone` 名称是不一样的，其他的根据实际情况填写。

```
vim obagent-only.yaml

obagent:
  servers:
    - name: obce01
      # Please don't use hostname, only IP can be supported
      ip: 172.xx.xxx.53
    - name: obce02
      ip: 172.xx.xxx.55
    - name: obce03
      ip: 172.xx.xxx.56
  global:
    # The working directory for obagent. obagent is started under this directory. This is a required f
ield.
    home_path: /home/admin/obagent
    # The port that pulls and manages the metrics. The default port number is 8088.
    server_port: 8088
    # Debug port for pprof. The default port number is 8089.
    pprof_port: 8089
    sql_port: 2881
    rpc_port: 2882
    # Log level. The default value is INFO.
    log_level: INFO
```

```
    # Log path. The default value is log/monagent.log.
    log_path: log/monagent.log
    # Encryption method. OBD supports aes and plain. The default value is plain.
    crypto_method: plain
    # Path to store the crypto key. The default value is conf/.config_secret.key.
    # crypto_path: conf/.config_secret.key
    # Size for a single log file. Log size is measured in Megabytes. The default value is 30M.
    log_size: 30
    # Expiration time for logs. The default value is 7 days.
    log_expire_day: 7
    # The maximum number for log files. The default value is 10.
    log_file_count: 10
    # Whether to use local time for log files. The default value is true.
    # log_use_localtime: true
    # Whether to enable log compression. The default value is true.
    # log_compress: true
    # Username for HTTP authentication. The default value is admin.
    http_basic_auth_user: admin
    # Password for HTTP authentication. The default value is root.
    http_basic_auth_password: eIY****ZeT
    # Username for debug service. The default value is admin.
    pprof_basic_auth_user: admin
    # Password for debug service. The default value is root.
    pprof_basic_auth_password: eIY****ZeT

    # 以下配置必须与 OceanBase 数据库一致
    # Monitor username for OceanBase Database. The user must have read access to OceanBase Database a
s a system tenant. The default value is root.
    monitor_user: monitor
    # Monitor password for OceanBase Database. The default value is empty. When a depends exists, OBD
gets this value from the oceanbase-ce of the depends. The value is the same as the root_password in oc
eanbase-ce.
    monitor_password: fLy***rp2R
    # Cluster name for OceanBase Database. When a depends exists, OBD gets this value from the oceanba
se-ce of the depends. The value is the same as the appname in oceanbase-ce.
    cluster_name: obdemo
    # Cluster ID for OceanBase Database. When a depends exists, OBD gets this value from the oceanbase
-ce of the depends. The value is the same as the cluster_id in oceanbase-ce.
    cluster_id: 1

  obce01:
    zone: zone1
  obce02:
    zone: zone2
  obce03:
    zone: zone3
```

注意

1. 指定节点的连接端口用的是 `sql_port` 不是 `mysql_port`，跟 OBServer 节点配置不一样。
2. 监控用户（`monitor_user` 对应）和密码需要在 sys 租户下创建：`grant select on oceanbase.* to monitor identified by 'fLy***rp2R';`。

使用 OBD 部署 OBAgent

首次使用 `deploy` 命令时需指定 OBAgent 的配置文件。

```
[admin@obce00 ~]$ obd cluster deploy obagent-only -c obagent-only.yaml
obagent-1.0.0 already installed.
+-----------------------------------------------------------------------------+
|                                  Packages                                   |
+------------+---------+---------+------------------------------------------+
| Repository | Version | Release | Md5                                      |
+------------+---------+---------+------------------------------------------+
| obagent    | 1.0.0   | 2.el8   | 1d65fc3d2cd08b26d6142b6149eb6806260aa7db |
+------------+---------+---------+------------------------------------------+
Repository integrity check ok
Parameter check ok
Open ssh connection ok
Remote obagent-1.0.0-1d65fc3d2cd08b26d6142b6149eb6806260aa7db repository install ok
Remote obagent-1.0.0-1d65fc3d2cd08b26d6142b6149eb6806260aa7db repository lib check ok
Cluster status check ok
Initializes obagent work home ok
obagent-only deployed
[admin@obce00 ~]$

[admin@obce00 ~]$ obd cluster list
+-------------------------------------------------------------------------+
|                              Cluster List                               |
+--------------+-----------------------------------------+----------------+
| Name         | Configuration Path                      | Status (Cached) |
+--------------+-----------------------------------------+----------------+
| obdemo   | /home/admin/.obd/cluster/obdemo   | running         |
| obagent-only | /home/admin/.obd/cluster/obagent-only | deployed        |
+--------------+-----------------------------------------+----------------+
```

`deploy` 命令运行后，配置文件将被复制到 `~/.obd/cluster/obagent-only/config.yaml`。后续修改 `obagent-only.yaml` 文件则不会生效。

若您想修改配置文件，可根据改动的影响范围在以下两种方式中任选其一:

- 使用命令 `edit-config` 编辑使用的配置文件。
- 使用 `destroy` 命令清理部署，重新读取 `obagent-only.yaml` 后再次开始部署。

`deploy` 命令只在各个节点上部署 OBAgent 软件（直接解压缩方式，不是 RPM 安装），目录如下:

```
[admin@obce01 ~]$ pwd
/home/admin
[admin@obce01 ~]$ tree obagent/
obagent/
├── bin
│   └── monagent -> /home/admin/.obd/repository/obagent/1.0.0/1d65fc3d2cd08b26d6142b6149eb6806260aa7
db/bin/monagent
├── conf
│   ├── config_properties
│   │   ├── monagent_basic_auth.yaml
│   │   └── monagent_pipeline.yaml
│   ├── module_config
│   │   ├── monagent_basic_auth.yaml
```

```
|    |    ├── monagent_config.yaml
|    |    ├── monitor_node_host.yaml
|    |    └── monitor_ob.yaml
|    ├── monagent.yaml
|    └── prometheus_config
|        ├── prometheus.yaml
|        └── rules
|            ├── host_rules.yaml
|            └── ob_rules.yaml
├── lib
├── log
|    └── monagent.log
└── run

9 directories, 12 files
```

使用 OBD 启动 OBAgent

您可使用命令 `start` 启动 OBAgent。

```
[admin@obce00 ~]$ obd cluster start obagent-only
Get local repositories and plugins ok
Open ssh connection ok
Cluster param config check ok
Check before start obagent ok
obagent program health check ok
+-------------------------------------------------+
|                    obagent                      |
+---------------+-------------+------------+------+
| ip            | server_port | pprof_port | status |
+---------------+-------------+------------+------+
| 172.xx.xxx.53 | 8088        | 8089       | active |
| 172.xx.xxx.55 | 8088        | 8089       | active |
| 172.xx.xxx.56 | 8088        | 8089       | active |
+---------------+-------------+------------+------+
obagent-only running
[admin@obce00 ~]$ obd cluster list
+-------------------------------------------------------------------------+
|                             Cluster List                                |
+--------------+-----------------------------------------+----------------+
| Name         | Configuration Path                      | Status (Cached) |
+--------------+-----------------------------------------+----------------+
| obdemo  | /home/admin/.obd/cluster/obdemo  | running         |
| obagent-only | /home/admin/.obd/cluster/obagent-only | running         |
+--------------+-----------------------------------------+----------------+
```

启动之后可以通过 OBD 查看 OBAgent 的部署情况。

```
[admin@obce00 ~]$ obd cluster display obagent-only
Get local repositories and plugins ok
Open ssh connection ok
Cluster status check ok
+-------------------------------------------------+
```

```
|                     obagent                     |
+---------------+-------------+------------+--------+
| ip            | server_port | pprof_port | status |
+---------------+-------------+------------+--------+
| 172.xx.xxx.53 | 8088        | 8089       | active |
| 172.xx.xxx.55 | 8088        | 8089       | active |
| 172.xx.xxx.56 | 8088        | 8089       | active |
+---------------+-------------+------------+--------+
```

Prometheus 配置

OBAgent 启动后会在节点自动生成 `Prometheus` 配置文件，该配置文件被放在 OBAgent 安装目录下，如 `/home/admin/obagent/conf/prometheus_config/`。这个配置文件可以供 `Prometheus` 产品直接使用。

示例如下:

```
vim prometheus_config/prometheus.yaml

global:
  scrape_interval:     1s
  evaluation_interval: 10s

rule_files:
  - "rules/*rules.yaml"

scrape_configs:
  - job_name: prometheus
    metrics_path: /metrics
    scheme: http
    static_configs:
    - targets:
      - 'localhost:9090'
  - job_name: node
    basic_auth:
      username: admin
      password: eIY****ZeT
    metrics_path: /metrics/node/host
    scheme: http
    static_configs:
      - targets:
        - 172.xx.xxx.53:8088
        - 172.xx.xxx.55:8088
        - 172.xx.xxx.56:8088
  - job_name: ob_basic
    basic_auth:
      username: admin
      password: eIY****ZeT
    metrics_path: /metrics/ob/basic
    scheme: http
    static_configs:
      - targets:
        - 172.xx.xxx.53:8088
        - 172.xx.xxx.55:8088
        - 172.xx.xxx.56:8088
```

```
  - job_name: ob_extra
    basic_auth:
      username: admin
      password: eIY****ZeT
    metrics_path: /metrics/ob/extra
    scheme: http
    static_configs:
      - targets:
        - 172.xx.xxx.53:8088
        - 172.xx.xxx.55:8088
        - 172.xx.xxx.56:8088
  - job_name: agent
    basic_auth:
      username: admin
      password: eIY****ZeT
    metrics_path: /metrics/stat
    scheme: http
    static_configs:
      - targets:
        - 172.xx.xxx.53:8088
        - 172.xx.xxx.55:8088
        - 172.xx.xxx.56:8088
```

配置项说明:

配置项	默认值	说明
scrape_interval	1s	抓取间隔
evaluation_interval	10s	评估规则间隔
rule_files	rules/*rules.yaml	报警规则
scrape_configs	--	抓取配置

下载解压缩 Prometheus 后，参考下文启动 Prometheus。

```
cd prometheus-2.30.3.linux-amd64 && ./prometheus ./prometheus.yaml
level=info ts=2021-11-19T05:41:57.789Z caller=main.go:400 msg="No time or size retention was set so us
ing the default time retention" duration=15d
level=info ts=2021-11-19T05:41:57.789Z caller=main.go:438 msg="Starting Prometheus" version="(version=
2.30.3, branch=HEAD, revision=f29caccc42557f6a8ec30ea9b3c8c089391bd5df)"
level=info ts=2021-11-19T05:41:57.789Z caller=main.go:443 build_context="(go=go1.17.1, user=root@5cff4
265f0e3, date=20211005-16:10:52)"
level=info ts=2021-11-19T05:41:57.789Z caller=main.go:444 host_details="(Linux 3.10.0-1127.19.1.el7.x8
6_64 #1 SMP Tue Aug 25 17:23:54 UTC 2020 x86_64 obce00 (none))"
level=info ts=2021-11-19T05:41:57.789Z caller=main.go:445 fd_limits="(soft=65535, hard=65535)"
level=info ts=2021-11-19T05:41:57.789Z caller=main.go:446 vm_limits="(soft=unlimited, hard=unlimited)"
level=info ts=2021-11-19T05:41:57.791Z caller=web.go:541 component=web msg="Start listening for connec
tions" address=0.0.0.0:9090
level=info ts=2021-11-19T05:41:57.792Z caller=main.go:822 msg="Starting TSDB ..."
level=info ts=2021-11-19T05:41:57.792Z caller=tls_config.go:191 component=web msg="TLS is disabled." h
ttp2=false
level=info ts=2021-11-19T05:41:57.795Z caller=head.go:479 component=tsdb msg="Replaying on-disk memor
y mappable chunks if any"
level=info ts=2021-11-19T05:41:57.795Z caller=head.go:513 component=tsdb msg="On-disk memory mappable
```

```
chunks replay completed" duration=7.579µs
level=info ts=2021-11-19T05:41:57.795Z caller=head.go:519 component=tsdb msg="Replaying WAL, this may
take a while"
level=info ts=2021-11-19T05:41:57.795Z caller=head.go:590 component=tsdb msg="WAL segment loaded" segm
ent=0 maxSegment=0
level=info ts=2021-11-19T05:41:57.795Z caller=head.go:596 component=tsdb msg="WAL replay completed" ch
eckpoint_replay_duration=30.235µs wal_replay_duration=183.559µs total_replay_duration=238.771µs
level=info ts=2021-11-19T05:41:57.796Z caller=main.go:849 fs_type=EXT4_SUPER_MAGIC
level=info ts=2021-11-19T05:41:57.796Z caller=main.go:852 msg="TSDB started"
level=info ts=2021-11-19T05:41:57.796Z caller=main.go:979 msg="Loading configuration file" filename=pr
ometheus.yaml
level=info ts=2021-11-19T05:41:57.802Z caller=main.go:1016 msg="Completed loading of configuration fil
e" filename=prometheus.yaml totalDuration=6.363415ms db_storage=1.001µs remote_storage=4.404µs web_han
dler=3.04µs query_engine=886ns scrape=418.015µs scrape_sd=129.011µs notify=5.433µs notify_sd=4.872µs r
ules=5.284804ms
level=info ts=2021-11-19T05:41:57.802Z caller=main.go:794 msg="Server is ready to receive web requests
."
```

启动后可通过浏览器进行访问：http://172.xx.xx.39:9090/graph 。

说明

此处链接中的 IP 为示例中配置 Prometheus 的服务器 IP，仅供参考。您需根据实际情况将其转换为自身配置 Prometheus 的服务器 IP。

具体 Prometheus 使用方法可以参考 Prometheus 官方问答。

OBAgent 重启方法

您可参考如下命令直接重启某个节点的 `OBAgent`：

```
kill -9 `pidof monagent`

cd /home/admin/obagent && nohup bin/monagent -c conf/monagent.yaml &
```

如果是集中重启，您可使用 OBD 命令重启：

```
obd cluster restart obagent_only
```

如果 `OBAgent` 是跟 `OceanBase` 一起部署的，则只能重启 `obagent` 组件：

```
obd cluster restart obdemo-obagent -c obagent

[admin@obce00 ~]$ obd cluster restart obdemo-obagent -c obagent
Get local repositories and plugins ok
Open ssh connection ok
Stop obagent ok
succeed
Get local repositories and plugins ok
Open ssh connection ok
Cluster param config check ok
```

```
Check before start obagent ok
obagent program health check ok
+-------------------------------------------------+
|                     obagent                     |
+--------------+-------------+------------+-------+
| ip           | server_port | pprof_port | status |
+--------------+-------------+------------+-------+
| 172.xx.xx.37 | 8088        | 8089       | active |
| 172.xx.xx.40 | 8088        | 8089       | active |
| 172.xx.xx.38 | 8088        | 8089       | active |
+--------------+-------------+------------+-------+
succeed
```

Grafana 使用

1. 点击链接从 Grafana 官网下载最新版本，并安装启动。下载地址：https://grafana.com/grafana/download?pg=get&plcmt=selfmanaged-box1-cta1。
2. 在 Grafana 里新增 Datasource，填入 Prometheus 地址。
3. 从 Grafana 官网下载 OceanBase 提交的主机性能模板和 OceanBase 性能模板文件，文件是 json 格式。
   - 主机性能模板：https://grafana.com/grafana/dashboards/15216
   - OceanBase 性能模板：https://grafana.com/grafana/dashboards/15215
4. 下载到本机后，在 Grafana 里 Import 这两个 json 文件。

附录

A.1 OceanBase 和 OBAgent 一起的配置文件

```
# Only need to configure when remote login is required
user:
   username: admin
#   password: your password if need
   key_file: /home/admin/.ssh/id_rsa.pub
   port: your ssh port, default 22
#   timeout: ssh connection timeout (second), default 30
oceanbase-ce:
  servers:
    - name: obce01
      # Please don't use hostname, only IP can be supported
      ip: 172.xx.xx.37
    - name: obce02
      ip: 172.xx.xx.40
    - name: obce03
      ip: 172.xx.xx.38
  global:
    # Please set devname as the network adaptor's name whose ip is  in the setting of severs.
```

```
    # if set severs as "127.0.0.1", please set devname as "lo"
    # if current ip is 192.168.1.10, and the ip's network adaptor's name is "eth0", please use "eth0"
    devname: eth0
    cluster_id: 2
    # please set memory limit to a suitable value which is matching resource.
    memory_limit: 8G # The maximum running memory for an observer
    system_memory: 3G # The reserved system memory. system_memory is reserved for general tenants. Th
e default value is 30G.
    stack_size: 512K
    cpu_count: 16
    cache_wash_threshold: 1G
    __min_full_resource_pool_memory: 268435456
    workers_per_cpu_quota: 10
    schema_history_expire_time: 1d
    # The value of net_thread_count had better be same as cpu's core number.
    net_thread_count: 4
    major_freeze_duty_time: Disable
    minor_freeze_times: 10
    enable_separate_sys_clog: 0
    enable_merge_by_turn: FALSE
    #datafile_disk_percentage: 20 # The percentage of the data_dir space to the total disk space. Thi
s value takes effect only when datafile_size is 0. The default value is 90.
    datafile_size: 50G
    syslog_level: WARN # System log level. The default value is INFO.
    enable_syslog_wf: false # Print system logs whose levels are higher than WARNING to a separate lo
g file. The default value is true.
    enable_syslog_recycle: true # Enable auto system log recycling or not. The default value is false.
    max_syslog_file_count: 10 # The maximum number of reserved log files before enabling auto recyclin
g. The default value is 0.
    # observer cluster name, consistent with obproxy's cluster_name
    appname: obdemo
    root_password: 0E****8d # root user password, can be empty
    proxyro_password: uY****zx # proxyro user pasword, consistent with obproxy's observer_sys_password
, can be empty
  obce01:
    mysql_port: 3881 # External port for OceanBase Database. The default value is 3881.
    rpc_port: 3882 # Internal port for OceanBase Database. The default value is 3882.
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data/obce
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo/obce
    zone: zone1
  obce02:
    mysql_port: 3881 # External port for OceanBase Database. The default value is 3881.
    rpc_port: 3882 # Internal port for OceanBase Database. The default value is 3882.
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data/obce
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo/obce
    zone: zone2
  obce03:
    mysql_port: 3881 # External port for OceanBase Database. The default value is 3881.
```

```
    rpc_port: 3882 # Internal port for OceanBase Database. The default value is 3882.
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data/obce
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo/obce
    zone: zone3
obproxy:
  servers:
    - 172.xx.xx.39
    - 172.xx.xx.37
  # Set ependent components for the component.
  # When the associated configurations are not done, OBD will automatically get the these configuratio
ns from the dependent components.
  depends:
    - oceanbase-ce
  global:
    listen_port: 3883 # External port. The default value is 3883.
    prometheus_listen_port: 3884 # The Prometheus port. The default value is 3884.
    home_path: /home/admin/obproxy
    # oceanbase root server list
    # format: ip:mysql_port;ip:mysql_port
    rs_list: 172.xx.xx.37:3881;172.xx.xx.40:3881;172.xx.xx.38:3881
    enable_cluster_checkout: false
    # observer cluster name, consistent with oceanbase-ce's appname
    cluster_name: obdemo
    obproxy_sys_password: 0M****tm # obproxy sys user password, can be empty
    observer_sys_password: uY****zx # proxyro user pasword, consistent with oceanbase-ce's proxyro_pas
sword, can be empty
obagent:
  servers:
    - name: obce01
      # Please don't use hostname, only IP can be supported
      ip: 172.xx.xx.37
    - name: obce02
      ip: 172.xx.xx.40
    - name: obce03
      ip: 172.xx.xx.38
  depends:
    - oceanbase-ce
  global:
    # The working directory for obagent. obagent is started under this directory. This is a required f
ield.
    home_path: /home/admin/obagent
    # The port that pulls and manages the metrics. The default port number is 8088.
    server_port: 8088
    # Debug port for pprof. The default port number is 8089.
    pprof_port: 8089
    sql_port: 3881
    rpc_port: 3882
    # Log level. The default value is INFO.
    log_level: INFO
    # Log path. The default value is log/monagent.log.
    log_path: log/monagent.log
    # Encryption method. OBD supports aes and plain. The default value is plain.
    crypto_method: plain
```

```
    # Path to store the crypto key. The default value is conf/.config_secret.key.
    # crypto_path: conf/.config_secret.key
    # Size for a single log file. Log size is measured in Megabytes. The default value is 30M.
    log_size: 30
    # Expiration time for logs. The default value is 7 days.
    log_expire_day: 7
    # The maximum number for log files. The default value is 10.
    log_file_count: 10
    # Whether to use local time for log files. The default value is true.
    # log_use_localtime: true
    # Whether to enable log compression. The default value is true.
    # log_compress: true
    # Username for HTTP authentication. The default value is admin.
    http_basic_auth_user: admin
    # Password for HTTP authentication. The default value is root.
    http_basic_auth_password: eIY****ZeT
    # Username for debug service. The default value is admin.
    pprof_basic_auth_user: admin
    # Password for debug service. The default value is root.
    pprof_basic_auth_password: eIY****ZeT

    # 以下配置必须与 OceanBase 数据库一致
    # Monitor username for OceanBase Database. The user must have read access to OceanBase Database a
s a system tenant. The default value is root.
    monitor_user: monitor
    # Monitor password for OceanBase Database. The default value is empty. When a depends exists, OBD
gets this value from the oceanbase-ce of the depends. The value is the same as the root_password in oc
eanbase-ce.
    monitor_password: fLy***rp2R
    # Cluster name for OceanBase Database. When a depends exists, OBD gets this value from the oceanba
se-ce of the depends. The value is the same as the appname in oceanbase-ce.
    # cluster_name: obdemo
    # Cluster ID for OceanBase Database. When a depends exists, OBD gets this value from the oceanbase
-ce of the depends. The value is the same as the cluster_id in oceanbase-ce.
    # cluster_id: 2
```

A.2 OBAgent 输出的性能数据

示例数据:

```
[admin@obce00 ~]$ curl --user admin:eIY****ZeT -L 'http://172.xx.xx.40:8088/metrics/ob/basic'
# HELP ob_active_session_num monitor collected metric
# TYPE ob_active_session_num untyped
ob_active_session_num{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="172.xx
.xx.40",tenant_name="sys"} 6
# HELP ob_cache_size_bytes monitor collected metric
# TYPE ob_cache_size_bytes untyped
ob_cache_size_bytes{app="OB",cache_name="location_cache",ob_cluster_id="2",ob_cluster_name="obdemo",obz
one="zone1",svr_ip="172.xx.xx.40",tenant_name="sys"} 4.193216e+06
ob_cache_size_bytes{app="OB",cache_name="user_tab_col_stat_cache",ob_cluster_id="2",ob_cluster_name="ob
demo",obzone="zone1",svr_ip="172.xx.xx.40",tenant_name="sys"} 6.290304e+06
ob_cache_size_bytes{app="OB",cache_name="user_table_stat_cache",ob_cluster_id="2",ob_cluster_name="obde
mo",obzone="zone1",svr_ip="172.xx.xx.40",tenant_name="sys"} 6.290304e+06
# HELP ob_partition_num monitor collected metric
# TYPE ob_partition_num untyped
ob_partition_num{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",role="1",svr_ip="17
```

```
2.xx.xx.40",tenant_name="sys"} 1198
# HELP ob_plan_cache_access_total monitor collected metric
# TYPE ob_plan_cache_access_total untyped
ob_plan_cache_access_total{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="1
72.xx.xx.40",tenant_name="sys"} 22984
# HELP ob_plan_cache_hit_total monitor collected metric
# TYPE ob_plan_cache_hit_total untyped
ob_plan_cache_hit_total{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="172.
xx.xx.40",tenant_name="sys"} 8645
# HELP ob_plan_cache_memory_bytes monitor collected metric
# TYPE ob_plan_cache_memory_bytes untyped
ob_plan_cache_memory_bytes{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="1
72.xx.xx.40",tenant_name="sys"} 1.3404239e+07
# HELP ob_server_num monitor collected metric
# TYPE ob_server_num untyped
ob_server_num{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",server_ips="172.xx.xx.
37,172.xx.xx.38,172.xx.xx.40",status="active",svr_ip="172.xx.xx.40"} 3
# HELP ob_sysstat monitor collected metric
# TYPE ob_sysstat untyped
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="10000",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 47136
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="10001",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 3.0078186e+07
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="10002",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 46863
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="10003",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 4.291008e+07
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="10005",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} -2.050408e+06
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="10006",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 4096
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="130000",svr_ip="
172.xx.xx.40",tenant_id="1",tenant_name="sys"} 8.59442e+07
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="130001",svr_ip="
172.xx.xx.40",tenant_id="1",tenant_name="sys"} 8.8080384e+07
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="130002",svr_ip="
172.xx.xx.40",tenant_id="1",tenant_name="sys"} 4.0265315e+08
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="130004",svr_ip="
172.xx.xx.40",tenant_id="1",tenant_name="sys"} 8.0530635e+08
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="140002",svr_ip="
172.xx.xx.40",tenant_id="1",tenant_name="sys"} 1.610612736e+09
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="140003",svr_ip="
172.xx.xx.40",tenant_id="1",tenant_name="sys"} 3.81681664e+08
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="140005",svr_ip="
172.xx.xx.40",tenant_id="1",tenant_name="sys"} 500
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="140006",svr_ip="
172.xx.xx.40",tenant_id="1",tenant_name="sys"} 1
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="20001",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 4122
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="20002",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 104938
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="30000",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="30001",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="30002",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="30005",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 15330
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="30006",svr_ip="1
```

```
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 7.566136e+06
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="40000",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 463
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="40001",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 1.863916e+06
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="40002",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="40003",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="40004",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="40005",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="40006",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="40007",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="40008",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="40009",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="40010",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 108
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="40011",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 8339
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="40012",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 329
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="50000",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="50001",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="50008",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="50009",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="60000",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="60001",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="60002",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="60003",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 4
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="60004",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 43325
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="60005",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 8.388608e+06
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="60019",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="60020",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="60021",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="60022",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="60023",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="60024",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="80040",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 11980
```

```
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="80041",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 2.0937394e+07
ob_sysstat{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",stat_id="80057",svr_ip="1
72.xx.xx.40",tenant_id="1",tenant_name="sys"} 0
# HELP ob_table_num monitor collected metric
# TYPE ob_table_num untyped
ob_table_num{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="172.xx.xx.40",t
enant_name="sys"} 1013
# HELP ob_waitevent_wait_seconds_total monitor collected metric
# TYPE ob_waitevent_wait_seconds_total untyped
ob_waitevent_wait_seconds_total{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_
ip="172.xx.xx.40",tenant_name="sys"} 49578.6026
# HELP ob_waitevent_wait_total monitor collected metric
# TYPE ob_waitevent_wait_total untyped
ob_waitevent_wait_total{app="OB",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="172.
xx.xx.40",tenant_name="sys"} 787473
# HELP ob_zone_current_timestamp monitor collected metric
# TYPE ob_zone_current_timestamp untyped
ob_zone_current_timestamp{app="OB",name="all_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo"
,obzone="zone1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="all_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo"
,obzone="zone1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="all_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo"
,obzone="zone1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="broadcast_version",ob_cluster_id="2",ob_cluster_name="obdemo",
obzone="zone1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="broadcast_version",ob_cluster_id="2",ob_cluster_name="obdemo",
obzone="zone1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="broadcast_version",ob_cluster_id="2",ob_cluster_name="obdemo",
obzone="zone1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="cluster",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zo
ne1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="config_version",ob_cluster_id="2",ob_cluster_name="obdemo",obz
one="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="frozen_time",ob_cluster_id="2",ob_cluster_name="obdemo",obzone
="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="frozen_version",ob_cluster_id="2",ob_cluster_name="obdemo",obz
one="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="gc_schema_version",ob_cluster_id="2",ob_cluster_name="obdemo",
obzone="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="global_broadcast_version",ob_cluster_id="2",ob_cluster_name="o
bdemo",obzone="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="idc",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1"
,svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="idc",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1"
,svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="idc",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1"
,svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="is_merge_error",ob_cluster_id="2",ob_cluster_name="obdemo",obz
one="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="is_merge_timeout",ob_cluster_id="2",ob_cluster_name="obdemo",o
bzone="zone1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="is_merge_timeout",ob_cluster_id="2",ob_cluster_name="obdemo",o
bzone="zone1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="is_merge_timeout",ob_cluster_id="2",ob_cluster_name="obdemo",o
bzone="zone1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="is_merging",ob_cluster_id="2",ob_cluster_name="obdemo",obzone=
"zone1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="is_merging",ob_cluster_id="2",ob_cluster_name="obdemo",obzone=
"zone1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="is_merging",ob_cluster_id="2",ob_cluster_name="obdemo",obzone=
```

```
"zone1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="last_merged_time",ob_cluster_id="2",ob_cluster_name="obdemo",o
bzone="zone1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="last_merged_time",ob_cluster_id="2",ob_cluster_name="obdemo",o
bzone="zone1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="last_merged_time",ob_cluster_id="2",ob_cluster_name="obdemo",o
bzone="zone1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="last_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo
",obzone="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="last_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo
",obzone="zone1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="last_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo
",obzone="zone1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="last_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo
",obzone="zone1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="lease_info_version",ob_cluster_id="2",ob_cluster_name="obdemo"
,obzone="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="merge_start_time",ob_cluster_id="2",ob_cluster_name="obdemo",o
bzone="zone1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="merge_start_time",ob_cluster_id="2",ob_cluster_name="obdemo",o
bzone="zone1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="merge_start_time",ob_cluster_id="2",ob_cluster_name="obdemo",o
bzone="zone1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="merge_status",ob_cluster_id="2",ob_cluster_name="obdemo",obzon
e="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="merge_status",ob_cluster_id="2",ob_cluster_name="obdemo",obzon
e="zone1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="merge_status",ob_cluster_id="2",ob_cluster_name="obdemo",obzon
e="zone1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="merge_status",ob_cluster_id="2",ob_cluster_name="obdemo",obzon
e="zone1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="privilege_version",ob_cluster_id="2",ob_cluster_name="obdemo",
obzone="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="proposal_frozen_version",ob_cluster_id="2",ob_cluster_name="ob
demo",obzone="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="recovery_status",ob_cluster_id="2",ob_cluster_name="obdemo",ob
zone="zone1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="recovery_status",ob_cluster_id="2",ob_cluster_name="obdemo",ob
zone="zone1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="recovery_status",ob_cluster_id="2",ob_cluster_name="obdemo",ob
zone="zone1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="region",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zon
e1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="region",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zon
e1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="region",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zon
e1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="snapshot_gc_ts",ob_cluster_id="2",ob_cluster_name="obdemo",obz
one="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="status",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zon
e1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="status",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zon
e1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="status",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zon
e1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="storage_format_version",ob_cluster_id="2",ob_cluster_name="obd
emo",obzone="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="storage_type",ob_cluster_id="2",ob_cluster_name="obdemo",obzon
e="zone1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="storage_type",ob_cluster_id="2",ob_cluster_name="obdemo",obzon
e="zone1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
```

```
ob_zone_current_timestamp{app="OB",name="storage_type",ob_cluster_id="2",ob_cluster_name="obdemo",obzon
e="zone1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="suspend_merging",ob_cluster_id="2",ob_cluster_name="obdemo",ob
zone="zone1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="suspend_merging",ob_cluster_id="2",ob_cluster_name="obdemo",ob
zone="zone1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="suspend_merging",ob_cluster_id="2",ob_cluster_name="obdemo",ob
zone="zone1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="time_zone_info_version",ob_cluster_id="2",ob_cluster_name="obd
emo",obzone="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="try_frozen_version",ob_cluster_id="2",ob_cluster_name="obdemo"
,obzone="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="warm_up_start_time",ob_cluster_id="2",ob_cluster_name="obdemo"
,obzone="zone1",svr_ip="172.xx.xx.40",zone=""} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="zone_type",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="
zone1",svr_ip="172.xx.xx.40",zone="zone1"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="zone_type",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="
zone1",svr_ip="172.xx.xx.40",zone="zone2"} 1.637303541e+15
ob_zone_current_timestamp{app="OB",name="zone_type",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="
zone1",svr_ip="172.xx.xx.40",zone="zone3"} 1.637303541e+15
# HELP ob_zone_stat monitor collected metric
# TYPE ob_zone_stat untyped
ob_zone_stat{app="OB",name="all_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone
1",svr_ip="172.xx.xx.40",zone="zone1"} 1
ob_zone_stat{app="OB",name="all_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone
1",svr_ip="172.xx.xx.40",zone="zone2"} 1
ob_zone_stat{app="OB",name="all_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone
1",svr_ip="172.xx.xx.40",zone="zone3"} 1
ob_zone_stat{app="OB",name="broadcast_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1
",svr_ip="172.xx.xx.40",zone="zone1"} 1
ob_zone_stat{app="OB",name="broadcast_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1
",svr_ip="172.xx.xx.40",zone="zone2"} 1
ob_zone_stat{app="OB",name="broadcast_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1
",svr_ip="172.xx.xx.40",zone="zone3"} 1
ob_zone_stat{app="OB",name="cluster",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="
172.xx.xx.40",zone=""} 0
ob_zone_stat{app="OB",name="config_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",s
vr_ip="172.xx.xx.40",zone=""} 1.63730315985165e+15
ob_zone_stat{app="OB",name="frozen_time",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_
ip="172.xx.xx.40",zone=""} 0
ob_zone_stat{app="OB",name="frozen_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",s
vr_ip="172.xx.xx.40",zone=""} 1
ob_zone_stat{app="OB",name="gc_schema_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1
",svr_ip="172.xx.xx.40",zone=""} 0
ob_zone_stat{app="OB",name="global_broadcast_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone
="zone1",svr_ip="172.xx.xx.40",zone=""} 1
ob_zone_stat{app="OB",name="idc",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="172.
xx.xx.40",zone="zone1"} 0
ob_zone_stat{app="OB",name="idc",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="172.
xx.xx.40",zone="zone2"} 0
ob_zone_stat{app="OB",name="idc",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="172.
xx.xx.40",zone="zone3"} 0
ob_zone_stat{app="OB",name="is_merge_error",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",s
vr_ip="172.xx.xx.40",zone=""} 0
ob_zone_stat{app="OB",name="is_merge_timeout",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1"
,svr_ip="172.xx.xx.40",zone="zone1"} 0
ob_zone_stat{app="OB",name="is_merge_timeout",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1"
,svr_ip="172.xx.xx.40",zone="zone2"} 0
ob_zone_stat{app="OB",name="is_merge_timeout",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1"
,svr_ip="172.xx.xx.40",zone="zone3"} 0
ob_zone_stat{app="OB",name="is_merging",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_i
```

```
p="172.xx.xx.40",zone="zone1"} 0
ob_zone_stat{app="OB",name="is_merging",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_i
p="172.xx.xx.40",zone="zone2"} 0
ob_zone_stat{app="OB",name="is_merging",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_i
p="172.xx.xx.40",zone="zone3"} 0
ob_zone_stat{app="OB",name="last_merged_time",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1"
,svr_ip="172.xx.xx.40",zone="zone1"} 1.637303150866399e+15
ob_zone_stat{app="OB",name="last_merged_time",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1"
,svr_ip="172.xx.xx.40",zone="zone2"} 1.637303150867716e+15
ob_zone_stat{app="OB",name="last_merged_time",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1"
,svr_ip="172.xx.xx.40",zone="zone3"} 1.637303150868863e+15
ob_zone_stat{app="OB",name="last_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zon
e1",svr_ip="172.xx.xx.40",zone=""} 1
ob_zone_stat{app="OB",name="last_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zon
e1",svr_ip="172.xx.xx.40",zone="zone1"} 1
ob_zone_stat{app="OB",name="last_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zon
e1",svr_ip="172.xx.xx.40",zone="zone2"} 1
ob_zone_stat{app="OB",name="last_merged_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zon
e1",svr_ip="172.xx.xx.40",zone="zone3"} 1
ob_zone_stat{app="OB",name="lease_info_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone
1",svr_ip="172.xx.xx.40",zone=""} 1.63730315985492e+15
ob_zone_stat{app="OB",name="merge_start_time",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1"
,svr_ip="172.xx.xx.40",zone="zone1"} 1.637303150866399e+15
ob_zone_stat{app="OB",name="merge_start_time",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1"
,svr_ip="172.xx.xx.40",zone="zone2"} 1.637303150867716e+15
ob_zone_stat{app="OB",name="merge_start_time",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1"
,svr_ip="172.xx.xx.40",zone="zone3"} 1.637303150868863e+15
ob_zone_stat{app="OB",name="merge_status",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr
_ip="172.xx.xx.40",zone=""} 0
ob_zone_stat{app="OB",name="merge_status",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr
_ip="172.xx.xx.40",zone="zone1"} 0
ob_zone_stat{app="OB",name="merge_status",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr
_ip="172.xx.xx.40",zone="zone2"} 0
ob_zone_stat{app="OB",name="merge_status",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr
_ip="172.xx.xx.40",zone="zone3"} 0
ob_zone_stat{app="OB",name="privilege_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1
",svr_ip="172.xx.xx.40",zone=""} 0
ob_zone_stat{app="OB",name="proposal_frozen_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone=
"zone1",svr_ip="172.xx.xx.40",zone=""} 1
ob_zone_stat{app="OB",name="recovery_status",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",
svr_ip="172.xx.xx.40",zone="zone1"} 0
ob_zone_stat{app="OB",name="recovery_status",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",
svr_ip="172.xx.xx.40",zone="zone2"} 0
ob_zone_stat{app="OB",name="recovery_status",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",
svr_ip="172.xx.xx.40",zone="zone3"} 0
ob_zone_stat{app="OB",name="region",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="1
72.xx.xx.40",zone="zone1"} 0
ob_zone_stat{app="OB",name="region",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="1
72.xx.xx.40",zone="zone2"} 0
ob_zone_stat{app="OB",name="region",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="1
72.xx.xx.40",zone="zone3"} 0
ob_zone_stat{app="OB",name="snapshot_gc_ts",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",s
vr_ip="172.xx.xx.40",zone=""} 0
ob_zone_stat{app="OB",name="status",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="1
72.xx.xx.40",zone="zone1"} 2
ob_zone_stat{app="OB",name="status",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="1
72.xx.xx.40",zone="zone2"} 2
ob_zone_stat{app="OB",name="status",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip="1
72.xx.xx.40",zone="zone3"} 2
ob_zone_stat{app="OB",name="storage_format_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="
zone1",svr_ip="172.xx.xx.40",zone=""} 4
```

```
ob_zone_stat{app="OB",name="storage_type",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr
_ip="172.xx.xx.40",zone="zone1"} 0
ob_zone_stat{app="OB",name="storage_type",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr
_ip="172.xx.xx.40",zone="zone2"} 0
ob_zone_stat{app="OB",name="storage_type",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr
_ip="172.xx.xx.40",zone="zone3"} 0
ob_zone_stat{app="OB",name="suspend_merging",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",
svr_ip="172.xx.xx.40",zone="zone1"} 0
ob_zone_stat{app="OB",name="suspend_merging",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",
svr_ip="172.xx.xx.40",zone="zone2"} 0
ob_zone_stat{app="OB",name="suspend_merging",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",
svr_ip="172.xx.xx.40",zone="zone3"} 0
ob_zone_stat{app="OB",name="time_zone_info_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="
zone1",svr_ip="172.xx.xx.40",zone=""} 0
ob_zone_stat{app="OB",name="try_frozen_version",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone
1",svr_ip="172.xx.xx.40",zone=""} 1
ob_zone_stat{app="OB",name="warm_up_start_time",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone
1",svr_ip="172.xx.xx.40",zone=""} 0
ob_zone_stat{app="OB",name="zone_type",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip
="172.xx.xx.40",zone="zone1"} 0
ob_zone_stat{app="OB",name="zone_type",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip
="172.xx.xx.40",zone="zone2"} 0
ob_zone_stat{app="OB",name="zone_type",ob_cluster_id="2",ob_cluster_name="obdemo",obzone="zone1",svr_ip
="172.xx.xx.40",zone="zone3"} 0
```

2.10 如何重启 OceanBase 集群

OceanBase 数据库自身并没有提供重启集群的命令。OceanBase 数据库的核心能力就是高可用（前提是三副本部署）。当少数派节点故障时，OceanBase 数据库内部会自动切换，依然可以为业务提供读写服务。OceanBase 数据库提供了停止和启动某个副本（`zone` 级别或者 `server` 级别）的功能，并且只允许停止少数派节点。

所以，OceanBase 集群的重启是靠外部操作。比如说用 `kill` 命令杀进程，然后再启动进程 `observer`。

2.8 章节中已经演示了如何杀单副本集群里的节点进程（进程启动时指定参数），下面演示三副本集群里的重启集群方法。

在生产环境中为了尽可能缩短集群不可用时间，重启集群采取一种保险的策略：按 `zone` 或 `server` 逐个重启集群节点。这个过程可能会比较长，持续几分钟到十几分钟。刚开始学习 OceanBase 数据库时，我们先掌握简单的重启方法，后续深入介绍 OceanBase 运维时，再介绍安全稳妥的重启方法。

手动重启 OceanBase 集群节点

```
#  ssh 到 节点 1
ssh 172.20.249.52

#  正常 kill 进程，除非是测试用或者评估过风险，否则不要用`kill -9` 。
kill  `pidof observer`

#  等待 60s，等进程完全退出
sleep 60

#  反复确认进程完全退出
ps -ef | grep observer

#  配置 LIBRARY PATH
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/oceanbase-ce/lib/

#  启动进程
cd /home/admin/oceanbase-ce && bin/observer

#  等待 10s 进程启动
sleep 10

#  反复确认进程启动时没有退出
ps -ef | grep observer | grep -v grep

#  等待 60s，等进程完全启动并恢复完毕
sleep 60

#  查看进程监听成功（默认监听 2881 和  2882 端口）
netstat -ntlp

#  在集群中查看节点状态（`status`）、开始服务时间（`start_service_time`）是否正常。
select a.zone,concat(a.svr_ip,':',a.svr_port) observer, cpu_total, (cpu_total-cpu_assigned) cpu_free,
round(mem_total/1024/1024/1024) mem_total_gb, round((mem_total-mem_assigned)/1024/1024/1024) mem_free_
gb, usec_to_time(b.last_offline_time) last_offline_time, usec_to_time(b.start_service_time) start_serv
```

```
ice_time, b.status, usec_to_time(b.stop_time) stop_time, b.build_version  from __all_virtual_server_st
at a join __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_port) order by a.zone, a.svr_ip;
```

在第一个节点重启成功后，才可重复操作第二个节点。如果只是测试，不在乎可用性，则可以忽略上面的确认过程，直接杀掉所有集群节点的进程，然后启动进程。这个时候集群节点起来后需要几分钟来恢复数据和通信，如果集群重启之前有大量的数据读写，这个节点进程的恢复时间可能会很长，需要十几分钟甚至几十分钟。

使用 OBD 重启集群

上述内容是手动重启 OceanBase 集群节点的原理，如下介绍使用 OBD 工具自动化重启 OceanBase 集群节点。

注意 当前 OBD 的重启集群并没有包含必要的检查操作。所以，测试环境中可以使用，生产环境中需要谨慎使用。

使用 OBD 重启集群的命令是：`obd cluster restart`。

```
obd cluster restart obce-3zones

输出:
[admin@obce00 oceanbase-ce]$ obd cluster restart obce-3zones
Get local repositories and plugins ok
Open ssh connection ok
Stop observer ok
Stop obproxy ok
obce-3zones stopped
Get local repositories and plugins ok
Open ssh connection ok
Cluster param config check ok
Check before start observer ok
Check before start obproxy ok
Start observer ok
observer program health check ok
Connect to observer ok
Wait for observer init ok
+-------------------------------------------------+
|                    observer                     |
+---------------+---------+------+-------+--------+
| ip            | version | port | zone  | status |
+---------------+---------+------+-------+--------+
| 172.20.249.49 | 3.1.0   | 2881 | zone2 | active |
| 172.20.249.51 | 3.1.0   | 2881 | zone3 | active |
| 172.20.249.52 | 3.1.0   | 2881 | zone1 | active |
+---------------+---------+------+-------+--------+

Start obproxy ok
obproxy program health check ok
Connect to obproxy ok
Initialize cluster
+-------------------------------------------------+
|                    obproxy                      |
+---------------+------+-----------------+--------+
| ip            | port | prometheus_port | status |
```

```
+---------------+------+-----------------+--------+
| 172.20.249.52 | 2883 | 2884            | active |
| 172.20.249.49 | 2883 | 2884            | active |
| 172.20.249.51 | 2883 | 2884            | active |
+---------------+------+-----------------+--------+
obce-3zones running
```

此处 OBD 重启集群时，默认重启了所有组件（包括 `OBServer` 和 `OBProxy`）。您也可通过 `-c` 命令指定重启具体的组件。

OBProxy 的重启特点跟 OBServer 是一样的，也有工作目录和启动参数，此处不作介绍。后续在 OceanBase 的运维章节中将介绍 OBProxy 的相关运维。

2.11 （高级）如何手动部署 OceanBase 集群

当您熟悉了使用 OBD 部署 OceanBase 集群的方法原理后，就可以尝试手动部署一套 OceanBase 集群。学会手动部署 OceanBase 集群后，当 OBD 的功能不满足您的需求时，您可以自己写程序脚本部署 OceanBase 集群，或者在集群出现异常时，手动做一些应急处理。

部署规划

本文介绍手动部署 OceanBase 集群三节点的方法，使用该方法需要通过中控机直接远程登录到 OceanBase 节点上部署启动 OBServer 进程，并在中控机上部署 OBProxy 进程。

机器信息

机器类型	云主机 ECS
IP	172.20.249.50
网卡名	eth0
OS	CentOS Linux release 8.4.2105
CPU	4C
内存	总内存 14G，可用内存 11G
磁盘1	云盘 /dev/vda 100G
磁盘2	云盘 /dev/vdb 100G

机器划分

角色	机器	备注
OBD	172.20.249.50	中控机，自动化部署软件
OBServer	172.20.249.52	OceanBase 数据库 zone1
OBServer	172.20.249.49	OceanBase 数据库 zone2
OBServer	172.20.249.51	OceanBase 数据库 zone3
OBProxy	172.20.249.50	OceanBase 访问反向代理
OBClient	172.20.249.50	OceanBase 命令行客户端

部署之前需要初始化服务器环境，详细操作请参考 2.4 如何初始化服务器环境。

机器三节点之间时间同步检查

检查本机和目标节点时间误差常用命令是： `clockdiff`。

示例:

```
[admin@obce02 oceanbase]$ sudo clockdiff 172.20.249.52
[sudo] password for admin:
.
host=172.20.249.52 rtt=750(187)ms/0ms delta=0ms/0ms Sun Sep 12 14:52:24 2021
[admin@obce02 oceanbase]$ sudo clockdiff 172.20.249.51
.
host=172.20.249.51 rtt=750(187)ms/0ms delta=0ms/0ms Sun Sep 12 14:52:30 2021
```

有些机器使用 `clockdiff` 可能会报错。此时您可使用以下命令判断时间同步误差。

```
[admin@obce02 oceanbase]$ ping -T tsandaddr 172.20.249.52 -c 2
PING 172.20.249.52 (172.20.249.52) 56(124) bytes of data.
64 bytes from 172.20.249.52: icmp_seq=1 ttl=64 time=0.161 ms
TS:     172.20.249.49   24851014 absolute
        172.20.249.52   -1
        172.20.249.52   0
        172.20.249.49   1

64 bytes from 172.20.249.52: icmp_seq=2 ttl=64 time=0.172 ms
TS:     172.20.249.49   24852054 absolute
        172.20.249.52   -1
        172.20.249.52   0
        172.20.249.49   1
```

三节点时间同步误差如果超过 50ms，初始化集群一定会失败。

注意

节点的时间误差会缓慢递增，也许当前集群能正常工作，但一天后，由于节点时间误差扩大到 50ms 以外，该节点就会掉线。

安装 OceanBase 软件包

手动部署 OceanBase 集群时需要安装 OceanBase 数据库的 OBServer 软件。

说明

示例中的安装包可能不是最新版本，建议下载最新的安装包，详细信息请参考 OceanBase 发布说明。

```
[admin@obce02 ~]$ ls -lrth /tmp/oceanbase-ce-*.rpm
-rw-r--r-- 1 admin admin  45M Sep 12 13:36 /tmp/oceanbase-ce-3.1.0-3.el8.x86_64.rpm

[admin@obce02 ~]$ sudo rpm -ivh /tmp/oceanbase-ce-*.rpm
warning: /tmp/oceanbase-ce-3.1.0-3.el8.x86_64.rpm: Header V4 RSA/SHA1 Signature, key ID e9b4a7aa: NOKE
```

```
Y
Verifying...                          ################## [100%]
Preparing...                          ################## [100%]
Updating / installing...
   1:oceanbase-ce-libs-3.1.0-3.el8    ################## [ 50%]
   2:oceanbase-ce-3.1.0-3.el8         ################## [100%]
```

软件包默认安装目录是：`/home/admin/oceanbase`。目录结构如下：

```
[admin@obce01 ~]$ tree oceanbase
oceanbase
├── bin
│   ├── import_time_zone_info.py
│   └── observer
├── etc
│   └── timezone_V1.log
└── lib
    ├── libaio.so -> libaio.so.1.0.1
    ├── libaio.so.1 -> libaio.so.1.0.1
    ├── libaio.so.1.0.1
    ├── libmariadb.so -> libmariadb.so.3
    └── libmariadb.so.3
```

当然，您也可以将 RPM 包直接解压到指定目录。

（可选）清理目录和数据

说明

首次部署不需要执行此操作，此操作主要用于安装部署失败后。安装部署失败后需要在清空目录和数据后重新部署。

```
kill -9 `pidof observer`
/bin/rm -rf ~/oceanbase/store/obdemo/*/*
```

检查目录结构，确认是否跟下文中初始化的目录结构一致。

```
tree ~/oceanbase/store/ /data/ /redo/
```

输出:

```
[admin@obce02 ~]$ tree ~/oceanbase/store/ /data/ /redo/
/home/admin/oceanbase/store/
└── obdemo
    ├── clog -> /redo/obdemo/clog
    ├── etc2 -> /redo/obdemo/etc2
    ├── etc3 -> /data/obdemo/etc3
    ├── ilog -> /redo/obdemo/ilog
    ├── slog -> /redo/obdemo/slog
    └── sstable -> /data/obdemo/sstable
/data/
```

```
└── obdemo
    ├── etc3
    └── sstable
/redo/
└── obdemo
    ├── clog
    ├── etc2
    ├── ilog
    └── slog

15 directories, 0 files
```

初始化数据目录

手动部署时，OceanBase 节点上的相关目录均需要手动创建。

说明

此操作仅需在首次部署时执行。

```
su - admin
sudo chown admin:admin ~/oceanbase/
mkdir -p ~/oceanbase/store/obdemo  /data/obdemo/{sstable,etc3} /redo/obdemo/{clog,ilog,slog,etc2}
for f in {clog,ilog,slog,etc2}; do ln -s /redo/obdemo/$f ~/oceanbase/store/obdemo/$f ; done
for f in {sstable,etc3}; do ln -s /data/obdemo/$f ~/oceanbase/store/obdemo/$f; done
```

您需注意的是:

- 您需创建工作目录下的总数据目录 `~/oceanbase/store/obdemo`、数据文件目录 `/data/obdemo` 和日志相关目录 `/redo/obdemo`。和使用 OBD 自动化部署的 OceanBase 节点目录相比，不同的是目录里加入了集群名标识（`obdemo`）。
- `~/oceanbase/store/obdemo` 是真实的目录，该目录下的子目录是映射到其他两个文件系统的路径（指 `/data/` 和 `/redo/`）。生产环境要求这两个文件系统尽可能是两块独立的物理盘，或者最低要求是两个独立的逻辑盘。

初始化后的目录结构如下。这个目录结构很重要，有时候 observer 进程启动失败就和目录结构和权限不对有关。

```
[admin@obce02 ~]$ tree ~/oceanbase/store/ /data/ /redo/
/home/admin/oceanbase/store/
└── obdemo
    ├── clog -> /redo/obdemo/clog
    ├── etc2 -> /redo/obdemo/etc2
    ├── etc3 -> /data/obdemo/etc3
    ├── ilog -> /redo/obdemo/ilog
    ├── slog -> /redo/obdemo/slog
    └── sstable -> /data/obdemo/sstable
/data/
└── obdemo
    ├── etc3
```

```
    └── sstable
/redo/
└── obdemo
    ├── clog
    ├── etc2
    ├── ilog
    └── slog

15 directories, 0 files
```

启动 OBServer 进程

每个机器的启动参数大部分都相同，只有少数不一样，需要特别留意。

- `172.20.249.52`

```
su - admin
echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/oceanbase/lib' >> ~/.bash_profile
. ~/.bash_profile

cd ~/oceanbase && bin/observer -i eth0 -p 2881 -P 2882 -z zone1 -d ~/oceanbase/store/obdemo -r '
172.20.249.52:2882:2881;172.20.249.49:2882:2881;172.20.249.51:2882:2881' -c 20210912 -n obdemo -
o "memory_limit=8G,cache_wash_threshold=1G,__min_full_resource_pool_memory=268435456,system_memo
ry=3G,memory_chunk_cache_size=128M,cpu_count=16,net_thread_count=4,datafile_size=50G,stack_size=
1536K,config_additional_dir=/data/obdemo/etc3;/redo/obdemo/etc2"
```

- `172.20.249.49`

```
su - admin
echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/oceanbase/lib' >> ~/.bash_profile
. ~/.bash_profile

cd ~/oceanbase && bin/observer -i eth0 -p 2881 -P 2882 -z zone2 -d ~/oceanbase/store/obdemo -r '
172.20.249.52:2882:2881;172.20.249.49:2882:2881;172.20.249.51:2882:2881' -c 20210912 -n obdemo -
o "memory_limit=8G,cache_wash_threshold=1G,__min_full_resource_pool_memory=268435456,system_memo
ry=3G,memory_chunk_cache_size=128M,cpu_count=16,net_thread_count=4,datafile_size=50G,stack_size=
1536K,config_additional_dir=/data/obdemo/etc3;/redo/obdemo/etc2"
```

- `172.20.249.51`

```
su - admin
echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/oceanbase/lib' >> ~/.bash_profile
. ~/.bash_profile

cd ~/oceanbase && bin/observer -i eth0 -p 2881 -P 2882 -z zone3 -d ~/oceanbase/store/obdemo -r '
172.20.249.52:2882:2881;172.20.249.49:2882:2881;172.20.249.51:2882:2881' -c 20210912 -n obdemo -
o "memory_limit=8G,cache_wash_threshold=1G,__min_full_resource_pool_memory=268435456,system_memo
ry=3G,memory_chunk_cache_size=128M,cpu_count=16,net_thread_count=4,datafile_size=50G,stack_size=
1536K,config_additional_dir=/data/obdemo/etc3;/redo/obdemo/etc2"
```

如果三个节点机型都一致，那么启动参数里只有参数 `-z` 不一样（`-z` 参数指定该节点是哪个 `zone`）。三个 `zone` 的三个节点初始化为一个三副本集群。

后面 `-o` 参数不是必须的。此处因为测试机器内存不足，所以需要指定一些影响内存的参数。如果您的机器内存足够（如大于 64G），则可以不需要 `-o` 参数部分。

检查三个节点进程启动正常，主要看端口监听是否正常。您可在中控机上批量查询。

```
[admin@obce00 oceanbase-ce]$ for OceanBase in $IPS;do echo $ob; ssh $ob "netstat -ntlp"; done
172.20.249.52
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:2881            0.0.0.0:*               LISTEN      10084/bin/observer
tcp        0      0 0.0.0.0:2882            0.0.0.0:*               LISTEN      10084/bin/observer
172.20.249.49
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:2881            0.0.0.0:*               LISTEN      10213/bin/observer
tcp        0      0 0.0.0.0:2882            0.0.0.0:*               LISTEN      10213/bin/observer
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
172.20.249.51
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:2881            0.0.0.0:*               LISTEN      10103/bin/observer
tcp        0      0 0.0.0.0:2882            0.0.0.0:*               LISTEN      10103/bin/observer
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
```

集群自举（初始化）

当 OceanBase 集群三个节点都正常启动，并且监听正常时，您可连接到任一节点（通过 `2881` 端口直连），进行自举（`bootstrap` 集群初始化）操作。

```
mysql -h 172.20.249.49 -u root -P 2881 -p -c -A
```

注意

这里若使用命令 `mysql -h 172.20.249.49 -u root@sys -P 2881 -p -c -A` 连接，则会提示：ERROR 8001 (08004): Server is initializing。

```
set session ob_query_timeout=1000000000; alter system bootstrap ZONE 'zone1' SERVER '172.20.249.52:288
2', ZONE 'zone2' SERVER '172.20.249.49:2882', ZONE 'zone3' SERVER '172.20.249.51:2882' ;
```

输出：

```
[admin@obce00 ~]$ mysql -h 172.20.249.49 -u root -P 2881 -p -c -A
```

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 3221225472
Server version: 5.7.25 OceanBase 3.1.0 (r3-b20901e8c84d3ea774beeaca963c67d7802e4b4e) (Built Aug 10 202
1 08:10:38)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> set session ob_query_timeout=1000000000; alter system bootstrap ZONE 'zone1' SERVER '1
72.20.249.52:2882', ZONE 'zone2' SERVER '172.20.249.49:2882', ZONE 'zone3' SERVER '172.20.249.51:2882
' ;
Query OK, 0 rows affected (0.001 sec)

Query OK, 0 rows affected (28.839 sec)

MySQL [(none)]> Bye
[admin@obce00 ~]$ mysql -h 172.20.249.49 -u root@sys -P 2881 -p -c -A
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 3221751629
Server version: 5.7.25 OceanBase 3.1.0 (r3-b20901e8c84d3ea774beeaca963c67d7802e4b4e) (Built Aug 10 202
1 08:10:38)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| oceanbase          |
| information_schema |
| mysql              |
| SYS                |
| LBACSYS            |
| ORAAUDITOR         |
| test               |
+--------------------+
7 rows in set (0.016 sec)
```

通常来说，只要严格按照前面步骤设置目录结构和权限、启动参数，集群自举都能成功。如果不成功，常见原因如下:

- 集群节点之间时间同步延时超过 50ms。
- 集群节点之间网络延时超过 100ms 。
- 集群节点上 OBServer 相关目录结构不对或者目录权限不对。
- 集群节点上进程 observer 启动参数有误。您可查看隐含参数的名字（如`__min_full_resource_pool_memory`）和参数 `-d` 的目录是否正确，参数 `-z`跟 IP 的对应关系、 参数中是否多了空格或是否有分隔符错误（有的是`,`，有的是`;`）。

- 集群节点可用内存低于进程 observer 启动参数 `memory_limit` 值。

设置相关密码

- 集群管理员（`root@sys`）密码

  您需执行如下命令为 `root@sys` 设置一个密码。

  ```
  alter user root identified by '4S****Sr' ;
  ```

- OBProxy 用户（`proxyro`）密码

  默认 OBProxy 连接 OceanBase 集群时使用用户 `proxyro` 。该用户不存在，需要手动创建。

  ```
  grant select on oceanbase.* to proxyro identified by 'SW****RH' ;
  ```

安装 OBProxy 软件包

手动部署时需要安装 OceanBase 数据库的 OBProxy 软件。

```
sudo rpm -ivh /tmp/obproxy-*.rpm
```

社区版的 OBProxy 软件默认安装到 `/home/admin/obproxy-版本号` 下，请根据实际软件安装目录进行查询。

```
[admin@obce00 ~]$ tree ~/obproxy-3.1.0/
/home/admin/obproxy-3.1.0/
└── bin
    ├── obproxy
    └── obproxyd.sh

1 directory, 2 files
```

说明

目前社区版的 OBProxy 安装后的文件结构很简单，后面可能会微调。

启动 OBProxy 进程

启动 OBProxy 进程推荐放在软件安装目录，进程 obproxy 会在该目录下生成目录 `etc` 用以保存 OBProxy 的运行参数，以及目录 `log` 用以保存运行日志。

```
cd ~/obproxy-3.1.0/ && bin/obproxy -r "172.20.249.52:2881;172.20.249.49:2881;172.20.249.51:2881" -p 28
83 -o "enable_strict_kernel_release=false,enable_cluster_checkout=false,enable_metadb_used=false" -c o
bdemo
```

输出:

```
[admin@obce00 obproxy-3.1.0]$ cd ~/obproxy-3.1.0/ && bin/obproxy -r "172.20.249.52:2881;172.20.249.49:
2881;172.20.249.51:2881" -p 2883 -o "enable_strict_kernel_release=false,enable_cluster_checkout=false,
enable_metadb_used=false" -c obdemo
bin/obproxy -r 172.20.249.52:2881;172.20.249.49:2881;172.20.249.51:2881 -p 2883 -o enable_strict_kerne
l_release=false,enable_cluster_checkout=false,enable_metadb_used=false -c obdemo
rs list: 172.20.249.52:2881;172.20.249.49:2881;172.20.249.51:2881
listen port: 2883
optstr: enable_strict_kernel_release=false,enable_cluster_checkout=false,enable_metadb_used=false
cluster_name: obdemo
[admin@obce00 obproxy-3.1.0]$ ps -ef|grep obproxy
admin      38206       1  2 15:11 ?        00:00:00 bin/obproxy -r 172.20.249.52:2881;172.20.249.49:288
1;172.20.249.51:2881 -p 2883 -o enable_strict_kernel_release=false,enable_cluster_checkout=false,enabl
e_metadb_used=false -c obdemo
admin      38229   28904  0 15:11 pts/2    00:00:00 grep --color=auto obproxy
[admin@obce00 obproxy-3.1.0]$
```

- 检查 OBProxy 监听是否正常

  进程 obproxy 默认会监听2个端口：2883 和 2884。

  ```
  [admin@obce00 obproxy-3.1.0]$ netstat -ntlp |grep obproxy
  (Not all processes could be identified, non-owned process info
   will not be shown, you would have to be root to see it all.)
  tcp        0      0 0.0.0.0:2883            0.0.0.0:*               LISTEN      38206/bin/obprox
  y
  tcp        0      0 0.0.0.0:2884            0.0.0.0:*               LISTEN      38206/bin/obprox
  y
  ```

- 登录 OBProxy 修改密码

  1. 登录 OBProxy登录用户名：`root@proxysys`，端口：`2883`。

     ```
     mysql -h 172.20.249.50 -u root@proxysys -P 2883 -p

     MySQL [(none)]> show proxyconfig like '%sys_password%';
     +----------------------+-------+--------------------------------+-------------+-----------
     ----+
     | name                 | value | info                           | need_reboot | visible_le
     vel |
     +----------------------+-------+--------------------------------+-------------+-----------
     ----+
     | observer_sys_password |       | password for observer sys user | false       | SY
     S           |
     | obproxy_sys_password  |       | password for obproxy sys user  | false       | SY
     S           |
     +----------------------+-------+--------------------------------+-------------+-----------
     ----+
     2 rows in set (0.000 sec)
     ```

  2. 修改 OBProxy 用户密码您可通过修改参数的方式来修改 OBProxy 用户密码，使用命令为 `alter proxyconfig set`。

     ```
     alter proxyconfig set obproxy_sys_password = 'w******p' ;
     ```

3. 修改 OBProxy 连接 OceanBase 集群用户 `proxyro` 的密码在修改 OBProxy 用户密码的同时还需要修改 OBProxy 连接 OceanBase 集群用户 `proxyro` 的密码，这样 OBProxy 才能和 OceanBase 集群正常连接。OBProxy 连接 OceanBase 集群用户 `proxyro` 的密码需和前面 OceanBase 集群初始化后创建的用户 `proxyro` 的密码一致。

```
alter proxyconfig set observer_sys_password = 'S******H' ;
```

4. 查看是否部署成功退出后，您可尝试通过 OBProxy 连接 OceanBase 集群，如果能查看所有会话，则说明 OBProxy 部署成功。

```
mysql -h172.20.249.50 -uroot@sys#obdemo -P2883 -p4******r -c -A oceanbase

输出:
[admin@obce00 obproxy-3.1.0]$ mysql -h172.20.249.50 -uroot@sys#obdemo -P2883 -p4******r -c
-A oceanbase
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.6.25 OceanBase 3.1.0 (r3-b20901e8c84d3ea774beeaca963c67d7802e4b4e) (Buil
t Aug 10 2021 08:10:38)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [oceanbase]> show processlist;
+------+--------+------+--------------------+-----------+-------------+-------------------
+-------------------+-------+-------+
| Id   | Tenant | User | Host               | db        | trans_count | svr_session_count
| state             | tid   | pid   |
+------+--------+------+--------------------+-----------+-------------+-------------------
+-------------------+-------+-------+
|    5 | sys    | root | 172.20.249.50:41524 | oceanbase |           0 |                 1
| MCS_ACTIVE_READER | 38206 | 38206 |
+------+--------+------+--------------------+-----------+-------------+-------------------
+-------------------+-------+-------+
1 row in set (0.000 sec)

MySQL [oceanbase]> show full processlist;
+------------+---------+--------+--------------------+-----------+---------+------+-------
-+----------------------+---------------+------+--------------+
| Id         | User    | Tenant | Host               | db        | Command | Time | Stat
e  | Info                 | Ip            | Port | Proxy_sessid |
+------------+---------+--------+--------------------+-----------+---------+------+-------
-+----------------------+---------------+------+--------------+
| 3222013775 | root    | sys    | 172.20.249.50:57436 | oceanbase | Query   |    0 | ACTIV
E | show full processlist | 172.20.249.51 | 2881 |            4 |
| 3221751633 | proxyro | sys    | 172.20.249.50:49344 | oceanbase | Sleep   |    2 | SLEE
P  | NULL                 | 172.20.249.49 | 2881 |            3 |
+------------+---------+--------+--------------------+-----------+---------+------+-------
-+----------------------+---------------+------+--------------+
2 rows in set (0.022 sec)
```

2.12 常见问题

机器环境初始化问题

ulimit 设置不生效

- 现象

```
ulimit -a
...
stack size              (kbytes, -s) 1024
...
```

此时，使用 admin 用户通过 `ulimit -s` 命令修改栈大小，操作系统报错：`cannot modify limit: Operation not permitted`。

- 原因

admin 用户 ulimit 配置未生效的原因可能是操作系统关闭了 PAM。 PAM 可用来限制登录用户的 ulimit 配置，如果不开启 PAM，则会使用 SSHD 的默认值（即 1024）。

- 解决办法

1. 修改 SSHD 配置文件 `sshd_config`，取消对 `UsePAM yes` 的注释。

```
sudo vim /etc/ssh/sshd_config
UsePAM yes
```

2. 重启 SSHD 服务。

```
sudo systemctl restart sshd
```

3. 再次修改 `ulimit.conf` 文件。

```
vim  /etc/security/limits.conf
* soft nofile 655360
* hard nofile 655360
* soft nproc 655360
* hard nproc 655360
* soft core unlimited
* hard core unlimited
* soft stack unlimited
* hard stack unlimited
```

4. 使用命令 `ulimit -a` 重新登录检查实际值。

OBD 部署问题

目录非空

- 现象

```
Initializes cluster work home x
[ERROR] fail to init zone1(172.20.249.53) data path: /data is not empty
```

- 原因

  CentOS 8.0 刚初始化的文件系统里目录里会有一个默认文件夹 `lost+found`。

- 解决办法

  您可运行命令 `sudo /bin/rm -rf /data/* /redo/*` 清空刚建的文件系统目录。

其他通用报错

- 现象

  `obd` 命令出错。

- 原因

  查看 `obd` 命令日志。

```
vim ~/.obd/log/obd + R
```

- 解决办法

  根据错误描述去解决。

OBServer 启动失败

找不到共享库

- 现象

  手动启动进程 observer，提示找不到共享库。

```
[admin@obce02 ~]$ cd oceanbase-ce/
[admin@obce02 oceanbase-ce]$ bin/observer
bin/observer: error while loading shared libraries: libmariadb.so.3: cannot open shared object f
ile: No such file or directory
```

- 原因

  没有将 OceanBase 数据库的 LIB 加到环境变量 `LD_LIBRARY_PATH` 中。

  LIB 目录如下:

```
[admin@obce02 ~]$ tree oceanbase-ce/
oceanbase-ce/
├── admin
├── bin
│   └── observer -> /home/admin/.obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc57d8a3f6
8b4c50f94aab80/bin/observer
<....>
├── lib
│   ├── libaio.so -> /home/admin/.obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc57d8a3f
68b4c50f94aab80/lib/libaio.so
│   ├── libaio.so.1 -> /home/admin/.obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc57d8a
3f68b4c50f94aab80/lib/libaio.so.1
│   ├── libaio.so.1.0.1 -> /home/admin/.obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc5
7d8a3f68b4c50f94aab80/lib/libaio.so.1.0.1
│   ├── libmariadb.so -> /home/admin/.obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc57d
8a3f68b4c50f94aab80/lib/libmariadb.so
│   └── libmariadb.so.3 -> /home/admin/.obd/repository/oceanbase-ce/3.1.0/84bd2fe27f8b8243cc5
7d8a3f68b4c50f94aab80/lib/libmariadb.so.3
```

- 解决办法

  您可将 OceanBase 数据库的 LIB 加到环境变量 `LD_LIBRARY_PATH` 中，也可以写到 `.bash_profile` 中。

```
echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/oceanbase-ce/lib/' >> ~/.bash_profile
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/oceanbase-ce/lib/
```

2.13 附录

Q：生产环境三节点 OceanBase 集群部署配置文件

当生产环境中机器内存大于 256G 时，您可参考下面配置文件。

```
# Only need to configure when remote login is required
user:
   username: admin
#   password: your password if need
   key_file: /home/admin/.ssh/id_rsa.pub
   port: your ssh port, default 22
#   timeout: ssh connection timeout (second), default 30
oceanbase-ce:
  servers:
    - name: obce01
      # Please don't use hostname, only IP can be supported
      ip: 172.20.249.53
    - name: obce02
      ip: 172.20.249.55
    - name: obce03
      ip: 172.20.249.56
  global:
    # Please set devname as the network adaptor's name whose ip is  in the setting of severs.
    # if set severs as "127.0.0.1", please set devname as "lo"
    # if current ip is 192.168.1.10, and the ip's network adaptor's name is "eth0", please use "eth0"
    devname: bond0
    cluster_id: 2
    # please set memory limit to a suitable value which is matching resource.
    # memory_limit: 200G # The maximum running memory for an observer
    # system_memory: 30G # The reserved system memory. system_memory is reserved for general tenants.
The default value is 30G.
    minor_freeze_times: 100
    minor_warm_up_duration_time: 0
    freeze_trigger_percentage: 40
    enable_merge_by_turn: FALSE
    datafile_disk_percentage: 50 # The percentage of the data_dir space to the total disk space. This
value takes effect only when datafile_size is 0. The default value is 90.
    # datafile_size: 500G
    syslog_level: INFO # System log level. The default value is INFO.
    enable_syslog_wf: false # Print system logs whose levels are higher than WARNING to a separate lo
g file. The default value is true.
    enable_syslog_recycle: true # Enable auto system log recycling or not. The default value is false.
    max_syslog_file_count: 50 # The maximum number of reserved log files before enabling auto recyclin
g. The default value is 0.
    # observer cluster name, consistent with obproxy's cluster_name
    appname: obce-3zones
    root_password: 0******d # root user password, can be empty
    proxyro_password: uY****zx # proxyro user pasword, consistent with obproxy's observer_sys_password
, can be empty
  obce01:
    mysql_port: 2881 # External port for OceanBase Database. The default value is 2881.
    rpc_port: 2882 # Internal port for OceanBase Database. The default value is 2882.
```

```
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo
    zone: zone1
  obce02:
    mysql_port: 2881 # External port for OceanBase Database. The default value is 2881.
    rpc_port: 2882 # Internal port for OceanBase Database. The default value is 2882.
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo
    zone: zone2
  obce03:
    mysql_port: 2881 # External port for OceanBase Database. The default value is 2881.
    rpc_port: 2882 # Internal port for OceanBase Database. The default value is 2882.
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo
    zone: zone3
obproxy:
  servers:
    - 172.20.249.53
    - 172.20.249.55
    - 172.20.249.56
  # Set dependent components for the component.
  # When the associated configurations are not done, OBD will automatically get the these configuratio
ns from the dependent components.
  depends:
    - oceanbase-ce
  global:
    listen_port: 2883 # External port. The default value is 2883.
    prometheus_listen_port: 2884 # The Prometheus port. The default value is 2884.
    home_path: /home/admin/obproxy
    # oceanbase root server list
    # format: ip:mysql_port;ip:mysql_port
    rs_list: 172.20.249.53:2881;172.20.249.55:2881;172.20.249.56:2881
    enable_cluster_checkout: false
    # observer cluster name, consistent with oceanbase-ce's appname
    cluster_name: obce-3zones
    obproxy_sys_password: 0M****tm # obproxy sys user password, can be empty
    observer_sys_password: uY****zx # proxyro user pasword, consistent with oceanbase-ce's proxyro_pas
sword, can be empty
```

Q：测试环境 3 台 ECS 模拟 6 节点集群配置文件

每台机器起 2 个节点，分别监听 2881/2882 端口和 3881/3882 端口。

```
# Only need to configure when remote login is required
user:
   username: admin
#   password: your password if need
   key_file: /home/admin/.ssh/id_rsa.pub
   port: your ssh port, default 22
#   timeout: ssh connection timeout (second), default 30
oceanbase-ce:
  servers:
    - name: obce01
      # Please don't use hostname, only IP can be supported
      ip: 172.20.249.53
    - name: obce02
      ip: 172.20.249.55
    - name: obce03
      ip: 172.20.249.56
    - name: obce04
      # Please don't use hostname, only IP can be supported
      ip: 172.20.249.53
    - name: obce05
      ip: 172.20.249.55
    - name: obce06
      ip: 172.20.249.56
  global:
    # Please set devname as the network adaptor's name whose ip is  in the setting of severs.
    # if set severs as "127.0.0.1", please set devname as "lo"
    # if current ip is 192.168.1.10, and the ip's network adaptor's name is "eth0", please use "eth0"
    devname: eth0
    cluster_id: 2
    # please set memory limit to a suitable value which is matching resource.
    memory_limit: 10G # The maximum running memory for an observer
    system_memory: 3G # The reserved system memory. system_memory is reserved for general tenants. Th
e default value is 30G.
    stack_size: 512K
    cpu_count: 16
    cache_wash_threshold: 1G
    __min_full_resource_pool_memory: 268435456
    workers_per_cpu_quota: 10
    schema_history_expire_time: 1d
    # The value of net_thread_count had better be same as cpu's core number.
    net_thread_count: 4
    major_freeze_duty_time: Disable
    minor_warm_up_duration_time: 0
    freeze_trigger_percentage: 40
    enable_separate_sys_clog: 0
    enable_merge_by_turn: FALSE
    #datafile_disk_percentage: 20 # The percentage of the data_dir space to the total disk space. Thi
s value takes effect only when datafile_size is 0. The default value is 90.
    datafile_size: 50G
    syslog_level: WARN # System log level. The default value is INFO.
    enable_syslog_wf: false # Print system logs whose levels are higher than WARNING to a separate lo
g file. The default value is true.
    enable_syslog_recycle: true # Enable auto system log recycling or not. The default value is false.
    max_syslog_file_count: 10 # The maximum number of reserved log files before enabling auto recyclin
g. The default value is 0.
```

```
    # observer cluster name, consistent with obproxy's cluster_name
    appname: obce-3zones
    root_password: 0E****8d # root user password, can be empty
    proxyro_password: uY****zx # proxyro user pasword, consistent with obproxy's observer_sys_password
, can be empty
  obce01:
    mysql_port: 2881 # External port for OceanBase Database. The default value is 2881.
    rpc_port: 2882 # Internal port for OceanBase Database. The default value is 2882.
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data/1
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo/1
    zone: zone1
  obce02:
    mysql_port: 2881 # External port for OceanBase Database. The default value is 2881.
    rpc_port: 2882 # Internal port for OceanBase Database. The default value is 2882.
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data/1
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo/1
    zone: zone2
  obce03:
    mysql_port: 2881 # External port for OceanBase Database. The default value is 2881.
    rpc_port: 2882 # Internal port for OceanBase Database. The default value is 2882.
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data/1
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo/1
    zone: zone3
  obce04:
    mysql_port: 3881 # External port for OceanBase Database. The default value is 2881.
    rpc_port: 3882 # Internal port for OceanBase Database. The default value is 2882.
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce2
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data/2
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo/2
    zone: zone1
  obce05:
    mysql_port: 3881 # External port for OceanBase Database. The default value is 2881.
    rpc_port: 3882 # Internal port for OceanBase Database. The default value is 2882.
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce2
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data/2
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
```

```
    redo_dir: /redo/2
    zone: zone2
  obce06:
    mysql_port: 3881 # External port for OceanBase Database. The default value is 2881.
    rpc_port: 3882 # Internal port for OceanBase Database. The default value is 2882.
    #  The working directory for OceanBase Database. OceanBase Database is started under this director
y. This is a required field.
    home_path: /home/admin/oceanbase-ce2
    # The directory for data storage. The default value is $home_path/store.
    data_dir: /data/2
    # The directory for clog, ilog, and slog. The default value is the same as the data_dir value.
    redo_dir: /redo/2
    zone: zone3
obproxy:
  servers:
    - 172.20.249.54
  # Set dependent components for the component.
  # When the associated configurations are not done, OBD will automatically get the these configuratio
ns from the dependent components.
  depends:
    - oceanbase-ce
  global:
    listen_port: 2883 # External port. The default value is 2883.
    prometheus_listen_port: 2884 # The Prometheus port. The default value is 2884.
    home_path: /home/admin/obproxy
    # oceanbase root server list
    # format: ip:mysql_port;ip:mysql_port
    rs_list: 172.20.249.53:2881;172.20.249.55:2881;172.20.249.56:2881
    enable_cluster_checkout: false
    # observer cluster name, consistent with oceanbase-ce's appname
    # cluster_name: obce-3zones
    obproxy_sys_password: 0M****tm # obproxy sys user password, can be empty
    # observer_sys_password: uY****zx # proxyro user pasword, consistent with oceanbase-ce's proxyro_p
assword, can be empty
```

第 3 章：如何使用 OceanBase 社区版

上一章，我们介绍了如何部署 OceanBase 社区版，本章介绍如何使用 OceanBase 社区版进行业务开发。

本章目录

3.1 查看 OceanBase 集群资源的使用情况

在业务开发之前，DBA 需要在 OceanBase 集群中创建一个数据库实例，这个实例就是 OceanBase 租户。OceanBase 集群可以分配出多个租户，这个能力就称为多租户。

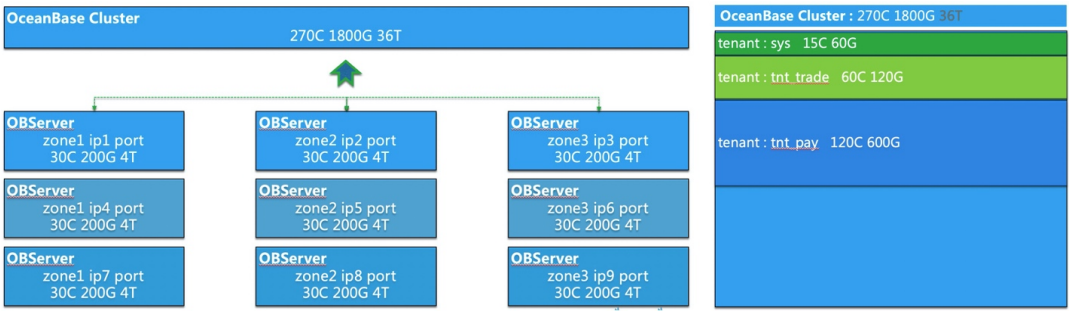
多租户原理简介

OceanBase 数据库以集群形态部署运行，提供给业务使用的是其中的租户。租户也叫实例，拥有一定的资源能力（如 CPU、内存和空间）。

OceanBase 是单进程软件，进程名为 `observer`。进程启动后，默认会将操作系统的大部分 CPU、内存和磁盘资源据为己有。当然，资源的使用情况也可以通过集群启动参数进行设置。

参数名	参数值	参数含义
`cpu_count`	16	默认取操作系统的 CPU 数减去 2。也可以自定义数目。
`memory_limit`	8G	进程默认使用的最大内存。如果设置为 `0` 表示该参数不受限制。和参数 `memory_limit_percentage` 二选一。
`memory_limit_percentage`	80	进程默认使用的最大内存占总可用内存的百分比。 和参数 `memory_limit` 二选一。
`datafile_size`	0	进程的数据文件（`block_file`）的初始化大小。如果设置为 `0` 表示该参数不受限制。
`datafile_disk_percentage`	75	进程的数据文件（`block_file`）的初始化大小占数据目录（`sstable`）所在文件系统可用空间的百分比。
`clog_disk_usage_limit_percentage`	95	进程的 clog 文件所在文件系统空间使用率上限百分比。达到这个值就会被认为“空间满”，clog 会停写。

进程 `observer` 取得的资源中 CPU 个数是声明式的，内存资源是独占的，磁盘空间是独占的（预分配）。



OceanBase 集群能把所有节点进程 `observer` 取得的资源集中管理，然后从集群中分配出多个租户，每个租户对应

一定的资源。这个资源的大小可以自行定义，并且资源可以在线调整，该功能也是弹性伸缩能力的体现。

OceanBase 数据库的租户资源定义包含 CPU、内存、空间、IOPS 和会话数。目前 OceanBase 只实现了 CPU 和内存的资源隔离，空间、IOPS 和会话数不起作用。建议创建资源时根据实际情况设置这些参数，尤其是空间资源，不要超出机器磁盘实际可用空间过多，否则将影响后期负载均衡。

查看集群可用资源

OceanBase 集群默认有个内部租户（`sys`），可以查看和管理集群的资源。计算集群剩余可用资源是为了避免创建业务租户时发生资源不足的情况。 您可使用以下 SQL 查看集群可用资源。

```
select a.zone,concat(a.svr_ip,':',a.svr_port) observer, cpu_total, cpu_assigned, (cpu_total-cpu_assign
ed) cpu_free, mem_total/1024/1024/1024 mem_total_gb, mem_assigned/1024/1024/1024 mem_assign_gb, (mem_t
otal-mem_assigned)/1024/1024/1024 mem_free_gb
from __all_virtual_server_stat a join __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_port)
order by a.zone, a.svr_ip;
```

集群可用资源如下:

```
+-------+--------------------+-----------+--------------+----------+----------------+----------------+
----------------+
| zone  | observer           | cpu_total | cpu_assigned | cpu_free | mem_total_gb   | mem_assign_gb
| mem_free_gb    |
+-------+--------------------+-----------+--------------+----------+----------------+----------------+
----------------+
| zone1 | 172.20.249.52:2882 |        14 |          2.5 |     11.5 | 5.000000000000 | 1.000000000000
| 4.000000000000 |
| zone2 | 172.20.249.49:2882 |        14 |          2.5 |     11.5 | 5.000000000000 | 1.000000000000
| 4.000000000000 |
| zone3 | 172.20.249.51:2882 |        14 |          2.5 |     11.5 | 5.000000000000 | 1.000000000000
| 4.000000000000 |
+-------+--------------------+-----------+--------------+----------+----------------+----------------+
----------------+
3 rows in set (0.008 sec)
```

您可运行以下 SQL 查看资源单元规格。

```
MySQL [oceanbase]> select * from __all_unit_config;
+----------------------------+----------------------------+----------------+-----------------+--------
-+---------+------------+------------+----------+----------+---------------+---------------------+
| gmt_create                 | gmt_modified               | unit_config_id | name            | max_cp
u | min_cpu | max_memory | min_memory | max_iops | min_iops | max_disk_size | max_session_num     |
+----------------------------+----------------------------+----------------+-----------------+--------
-+---------+------------+------------+----------+----------+---------------+---------------------+
| 2021-09-12 14:49:43.422194 | 2021-09-13 13:55:55.778987 |              1 | sys_unit_config |
5 |     2.5 | 1610612736 | 1073741824 |    10000 |     5000 |   53687091200 | 9223372036854775807 |
+----------------------------+----------------------------+----------------+-----------------+--------
-+---------+------------+------------+----------+----------+---------------+---------------------+
1 row in set (0.001 sec)

alter resource unit sys_unit_config min_cpu=5,max_cpu=5,min_memory='1610612736B',max_memory='161061273
6B';
```

从上面可以看出，资源单元规格 `sys_unit_config` 的定义里 CPU 和内存的最小值和最大值定义不一样，而前面统计资源中的已分配资源时是按最小值进行计算的。这将会导致剩余的可用资源计算不那么准确。所以，建议把该资源单元规格中定义的 CPU 和内存的最小值和最大值拉平。

此时查询剩余可用资源将较为准确。

```
MySQL [oceanbase]> select a.zone,concat(a.svr_ip,':',a.svr_port) observer, cpu_total, cpu_assigned, (c
pu_total-cpu_assigned) cpu_free, mem_total/1024/1024/1024 mem_total_gb, mem_assigned/1024/1024/1024 me
m_assign_gb, (mem_total-mem_assigned)/1024/1024/1024 mem_free_gb  from __all_virtual_server_stat a joi
n __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_port) order by a.zone, a.svr_ip;
```

剩余可用资源如下:

```
+-------+--------------------+-----------+--------------+----------+----------------+----------------+
----------------+
| zone  | observer           | cpu_total | cpu_assigned | cpu_free | mem_total_gb   | mem_assign_gb
| mem_free_gb    |
+-------+--------------------+-----------+--------------+----------+----------------+----------------+
----------------+
| zone1 | 172.20.249.52:2882 |        14 |            5 |        9 | 5.000000000000 | 1.500000000000
| 3.500000000000 |
| zone2 | 172.20.249.49:2882 |        14 |            5 |        9 | 5.000000000000 | 1.500000000000
| 3.500000000000 |
| zone3 | 172.20.249.51:2882 |        14 |            5 |        9 | 5.000000000000 | 1.500000000000
| 3.500000000000 |
+-------+--------------------+-----------+--------------+----------+----------------+----------------+
----------------+
3 rows in set (0.005 sec)
```

查看资源分配细节。

```
select t1.name resource_pool_name, t2.`name` unit_config_name, t2.max_cpu, t2.min_cpu, t2.max_memory/1
024/1024/1024 max_mem_gb, t2.min_memory/1024/1024/1024 min_mem_gb, t3.unit_id, t3.zone, concat(t3.svr_
ip,':',t3.`svr_port`) observer,t4.tenant_id, t4.tenant_name
from __all_resource_pool t1 join __all_unit_config t2 on (t1.unit_config_id=t2.unit_config_id)
    join __all_unit t3 on (t1.`resource_pool_id` = t3.`resource_pool_id`)
    left join __all_tenant t4 on (t1.tenant_id=t4.tenant_id)
order by t1.`resource_pool_id`, t2.`unit_config_id`, t3.unit_id
;
```

查看资源分配结果如下:

```
+--------------------+------------------+---------+---------+----------------+----------------+-------
--+-------+--------------------+-----------+-------------+
| resource_pool_name | unit_config_name | max_cpu | min_cpu | max_mem_gb     | min_mem_gb     | unit_i
d | zone  | observer           | tenant_id | tenant_name |
+--------------------+------------------+---------+---------+----------------+----------------+-------
--+-------+--------------------+-----------+-------------+
| sys_pool           | sys_unit_config  |       5 |       5 | 1.500000000000 | 1.500000000000 |
1 | zone1 | 172.20.249.52:2882 |         1 | sys         |
| sys_pool           | sys_unit_config  |       5 |       5 | 1.500000000000 | 1.500000000000 |
2 | zone2 | 172.20.249.49:2882 |         1 | sys         |
| sys_pool           | sys_unit_config  |       5 |       5 | 1.500000000000 | 1.500000000000 |
3 | zone3 | 172.20.249.51:2882 |         1 | sys         |
```

```
+--------------------+-----------------+---------+---------+----------------+----------------+-------
--+-------+--------------------+-----------+-------------+
3 rows in set (0.006 sec)
```

该运行结果显示内部租户的资源池（`resource pool`），由每个 `zone` 里的一个节点上的资源单元（`resource unit`）组成，每个资源单元使用同一规格（`sys_unit_config`）。

3.2 如何创建和连接 MySQL 租户

创建租户分为以下三步:

1. 创建资源单元规格:该步骤为可选步骤,如果有合适的规格可以跳过此步骤,直接进行复用。
2. 创建资源池:可以每个 `zone` 一个资源池,使用独立的资源单元规格,也可以所有 `zone` 使用同一个资源单元规格,都在一个资源池下。
3. 创建租户:创建租户时需关联到第 2 步中创建的资源池。

创建资源单元规格

创建资源单元规格后并不会立即分配资源。资源单元规格元数据在视图 `__all_unit_config` 里,建议创建之前先查看此视图,如果有合适的规格,可直接复用。

- 语法

  资源单元规格创建语法如下:

```
CREATE RESOURCE UNIT unit_name
MAX_CPU [=] cpu_num,
MAX_MEMORY [=] mem_size,
MAX_IOPS [=] iops_num,
MAX_DISK_SIZE [=] disk_size,
MAX_SESSION_NUM [=] session_num,
[MIN_CPU [=] cpu_num,]
[MIN_MEMORY [=] mem_size,]
[MIN_IOPS [=] iops_num] ;
```

  参数解释:

参数	描述
MAX_CPU	指定 CPU 的最多数量。
MAX_MEMORY	指定最大内存容量,取值范围为 [1073741824, +∞) 。单位为字节 B,最小值为 1G。
MAX_IOPS	指定 IOPS 的最多数量,取值范围为 [128,+∞)。
MAX_DISK_SIZE	指定最大硬盘容量,取值范围为 [536870912,+∞] 。单位为字节,最小值为 512M。
MAX_SESSION_NUM	指定 Session 的最多数量,取值范围为 [64,+∞)。
MIN_CPU	指定 CPU 的最少数量。
MIN_MEMORY	指定最小内存容量。

MIN_IOPS	指定 IOPS 的最少数量。

- 示例：分别创建两个资源单元规格 S1、S2

```
CREATE resource unit S1 max_cpu=3, min_cpu=3, max_memory='3G', min_memory='3G', max_iops=10000,
min_iops=1000, max_session_num=1000000, max_disk_size='1024G';
```

```
CREATE resource unit S2 max_cpu=4, min_cpu=4, max_memory='3G', min_memory='3G', max_iops=10000,
min_iops=1000, max_session_num=1000000, max_disk_size='1024G';
```

查看创建的资源单元规格:

```
MySQL [oceanbase]> select * from __all_unit_config;
+----------------------------+----------------------------+----------------+-----------------+--
-------+---------+------------+------------+----------+----------+---------------+-------------
-------+
| gmt_create                 | gmt_modified               | unit_config_id | name            | m
ax_cpu | min_cpu | max_memory | min_memory | max_iops | min_iops | max_disk_size | max_session_n
um     |
+----------------------------+----------------------------+----------------+-----------------+--
-------+---------+------------+------------+----------+----------+---------------+-------------
-------+
| 2021-09-12 14:49:43.422194 | 2021-09-13 14:02:08.782027 |              1 | sys_unit_config
|       5 |       5 | 1610612736 | 1610612736 |    10000 |     5000 |   53687091200 | 9223372036
854775807 |
| 2021-09-14 11:10:51.296235 | 2021-09-14 11:10:51.296235 |           1002 | S1
|       3 |       3 | 3221225472 | 3221225472 |    10000 |     1000 | 1099511627776
|             1000000 |
| 2021-09-14 11:13:46.773639 | 2021-09-14 11:13:46.773639 |           1004 | S2
|       4 |       4 | 3221225472 | 3221225472 |    10000 |     1000 | 1099511627776
|             1000000 |
+----------------------------+----------------------------+----------------+-----------------+--
-------+---------+------------+------------+----------+----------+---------------+-------------
-------+
3 rows in set (0.001 sec)
```

创建资源池

资源池中的资源是从节点资源中分配而来。资源池在每个节点里的部分被称为资源单元。每个资源单元的大小通过创建时指定的资源单元规格大小确定。

- 语法

  创建资源池的语法如下:

```
CREATE RESOURCE POOL poolname
UNIT [=] unitname,
UNIT_NUM [=] unitnum,
ZONE_LIST [=] ('zone' [, 'zone' ...]);
```

参数解释:

参数	描述
poolname	指定要创建的资源池的名称。
UNIT = unitname	指定资源规格的名称。
UNIT_NUM = unitnum	指定要创建的单个 `Zone` 下的 `Unit` 个数。每个单元会根据当前集群负载，自动在每个 `Zone` 中选择一个 `Server` 负载，但同一个资源池的多个 Unit 不能分配到同一个 `Server`，即一个资源池包含的 `Unit` 个数不能超过单 `Zone` 内 `Server` 的个数。
ZONE_LIST = ('zone' [, 'zone' ...])	指定要创建的资源池所属的 `Zone`。 如果不指定，就默认在所有 `Zone` 创建这个资源单元。

- 示例

本次示例中创建 2 个资源池，分别使用不同的资源单元规格。且其中一个资源池横跨两个 `Zone`。 这样操作主要是为了演示资源池创建时的灵活性。生产环境中，为了管理方便，可以一个资源池横跨三个 `Zone`，并且使用同一种资源单元规格。

```
create resource pool pool_1 unit='S1' , unit_num=1, zone_list=('zone1' ,'zone2') ;
```

```
create resource pool pool_2 unit='S2' , unit_num=1, zone_list=('zone3');
```

- 查看资源池详情：资源池详情如下:

```
+--------------------+-----------------+---------+---------+----------------+-------------
---+---------+-------+--------------------+-----------+-------------+
| resource_pool_name | unit_config_name | max_cpu | min_cpu | max_mem_gb     | min_mem_g
b     | unit_id | zone  | observer           | tenant_id | tenant_name |
+--------------------+-----------------+---------+---------+----------------+-------------
---+---------+-------+--------------------+-----------+-------------+
| sys_pool           | sys_unit_config  |       5 |       5 | 1.500000000000 | 1.5000000000
00 |       1 | zone1 | 172.20.249.52:2882 |         1 | sys         |
| sys_pool           | sys_unit_config  |       5 |       5 | 1.500000000000 | 1.5000000000
00 |       2 | zone2 | 172.20.249.49:2882 |         1 | sys         |
| sys_pool           | sys_unit_config  |       5 |       5 | 1.500000000000 | 1.5000000000
00 |       3 | zone3 | 172.20.249.51:2882 |         1 | sys         |
| pool_1             | S1               |       3 |       3 | 3.000000000000 | 3.0000000000
00 |    1006 | zone1 | 172.20.249.52:2882 |      NULL |    NULL     |
| pool_1             | S1               |       3 |       3 | 3.000000000000 | 3.0000000000
00 |    1007 | zone2 | 172.20.249.49:2882 |      NULL |    NULL     |
| pool_2             | S2               |       4 |       4 | 3.000000000000 | 3.0000000000
00 |    1008 | zone3 | 172.20.249.51:2882 |      NULL |    NULL     |
+--------------------+-----------------+---------+---------+----------------+-------------
---+---------+-------+--------------------+-----------+-------------+
6 rows in set (0.037 sec)
```

- 查看集群剩余可用资源：剩余可用资源如下:

```
+-------+--------------------+-----------+--------------+----------+----------------+------
----------+----------------+
| zone  | observer           | cpu_total | cpu_assigned | cpu_free | mem_total_gb   | mem_a
ssign_gb  | mem_free_gb    |
+-------+--------------------+-----------+--------------+----------+----------------+------
----------+----------------+
| zone1 | 172.20.249.52:2882 |        14 |            8 |        6 | 5.000000000000 | 4.500
000000000 | 0.500000000000 |
| zone2 | 172.20.249.49:2882 |        14 |            8 |        6 | 5.000000000000 | 4.500
000000000 | 0.500000000000 |
| zone3 | 172.20.249.51:2882 |        14 |            9 |        5 | 5.000000000000 | 4.500
000000000 | 0.500000000000 |
+-------+--------------------+-----------+--------------+----------+----------------+------
----------+----------------+
3 rows in set (0.026 sec)
```

说明 资源池创建出来后，集群的可用资源就减少了。但是这个资源池还没有关联到具体租户，所以无法被业务使用。

创建租户

租户就是实例，创建租户需要关联到某个资源池。

- 语法

  创建租户的语法如下:

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

tenant_characteristic_list:
  tenant_characteristic [, tenant_characteristic...]

tenant_characteristic:
      COMMENT 'string'
    | {CHARACTER SET | CHARSET} [=] charsetname
    | COLLATE [=]  collationname
    | REPLICA_NUM [=] num
    | ZONE_LIST [=] (zone [, zone...])
    | PRIMARY_ZONE [=] zone
    | DEFAULT TABLEGROUP [=] {NULL | tablegroup}
    | RESOURCE_POOL_LIST [=](poolname [, poolname...])
    | LOGONLY_REPLICA_NUM [=] num
    | LOCALITY [=] 'locality description'

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

  参数解释:

参数	描述

tenant_name	指定租户名。最长 64 个字节，只能由大小写英文字母、数字和下划线组成，而且必须以字母或下划线开头，并且不能是 OceanBase 数据库的关键字。
IF NOT EXISTS	如果要创建的租户名已存在，并且没有指定 IF NOT EXISTS，则会报错。
RESOURCE_POOL_LIST	资源池列表，为创建租户时的必填项 。
DEFAULT TABLEGROUP	设置租户默认表分组信息，NULL 表示取消默认表分组。如果不指定，默认为 NULL。
COMMENT	修改注释。
CHARACTER SET	CHARSET
COLLATE	指定校对规则。
REPLICA_NUM	指定副本数。
ZONE_LIST	指定要修改的 Zone 列表。
PRIMARY_ZONE	指定 Primary Zone。
LOGONLY_REPLICA_NUM	指定日志副本数。
LOCALITY	描述副本在 Zone 间的分布情况，如：F@z1,F@z2,F@z3,R@z4 表示 z1、z2、z3 为全功能副本，z4 为只读副本。
system_var_name	指定租户系统变量值。其中 `ob_compatibility_mode` 用于指定租户的兼容模式（可选 `mysql` 与 `oracle` 。社区版只支持 `mysql` ），只能在创建时指定。如果不指定 `ob_compatibility_mode`，默认兼容模式为 `mysql`。

- 示例

  创建租户时可以通过 `set` 命令指定租户变量（参数）值。

```
create tenant obmysql resource_pool_list=('pool_1','pool_2'), primary_zone='RANDOM',comment 'mys
ql tenant/instance', charset='utf8'  set ob_tcp_invited_nodes='%' ;
```

  查看创建结果:

```
MySQL [oceanbase]> select * from gv$tenant;
+-----------+-------------+------------------+------------------+----------------+------------
-----------+-----------+----------------------------------------------+
| tenant_id | tenant_name | zone_list        | primary_zone     | collation_type | inf
o                  | read_only | locality                                     |
+-----------+-------------+------------------+------------------+----------------+------------
-----------+-----------+----------------------------------------------+
|         1 | sys         | zone1;zone2;zone3 | zone1;zone2,zone3 |              0 | system tena
nt         |         0 | FULL{1}@zone1, FULL{1}@zone2, FULL{1}@zone3 |
|      1001 | obmysql     | zone1;zone2;zone3 | RANDOM            |              0 | mysql tenan
t/instance |         0 | FULL{1}@zone1, FULL{1}@zone2, FULL{1}@zone3 |
+-----------+-------------+------------------+------------------+----------------+------------
-----------+-----------+----------------------------------------------+
```

```
2 rows in set (0.005 sec)
```

3.3 如何连接租户

OceanBase 开源版的租户只兼容 MySQL，连接协议兼容 MySQL 5.6 。因此 MySQL 命令行客户端或者图形化工具理论上也能连接 OceanBase 数据库的租户。此外，OceanBase 数据库也提供专属的命令行客户端工具 OBClient 和图形化客户端工具 ODC。

MySQL 客户端连接

OceanBase MySQL 租户支持传统 MySQL 客户端连接，连接方式基本不变，跟传统 MySQL 不一样的地方是用户名的格式。

示例:

```
mysql -h172.20.249.50 -uroot@sys#obdemo -P2883 -p4S****Sr -c -A oceanbase

Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 5.6.25 OceanBase 3.1.0 (r3-b20901e8c84d3ea774beeaca963c67d7802e4b4e) (Built Aug 10 202
1 08:10:38)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [oceanbase]> show databases;
+--------------------+
| Database           |
+--------------------+
| oceanbase          |
| information_schema |
| mysql              |
| SYS                |
| LBACSYS            |
| ORAAUDITOR         |
| test               |
+--------------------+
7 rows in set (0.008 sec)
```

说明:

- `-h`：提供 OceanBase 数据库连接 IP，通常是一个 OBProxy 地址。
- `-u`：提供租户的连接账户，格式有两种：**用户名@租户名#集群名** 或者 **集群名:租户名:用户名** 。MySQL 租户的管理员用户名默认是 root。
- `-P`：提供 OceanBase 数据库连接端口，也就是 OBProxy 的监听端口，默认是 `2883`，可以自定义。
- `-p`：提供账户密码，为了安全可以不提供，改为在后面提示符下输入，密码文本不可见。
- `-c`：表示在 MySQL 运行环境中不要忽略注释。

- `-A`：表示在 MySQL 连接数据库时不自动获取统计信息。
- `oceanbase`：访问的数据库名，可以改为业务数据库。

新创建的业务租户的管理员（`root`）密码默认为空，需要修改密码。

```
$ strings /dev/urandom |tr -dc A-Za-z0-9 | head -c8; echo
b*******t

mysql -h172.20.249.50 -uroot@obmysql#obdemo -P2883 -p -c -A oceanbase

MySQL [oceanbase]> alter user root identified by 'b*******t' ;
Query OK, 0 rows affected (0.118 sec)
```

如果没有安装 MySQL 客户端，可以安装 `mariadb-server`。MySQL 官方 8.0 的客户端连接协议在密码处调整了逻辑，导致无法通过早期的 OBProxy 连接到 OceanBase 的 MySQL 租户，会报密码错误。可以通过选项 `--default-auth=mysql_native_password` 解决这个问题。

```
# 安装 mariadb
sudo yum -y install mariadb-server.x86_64

# 或者
# 安装官方 mysql 客户端
sudo yum -y install mysql.x86_64

mysql -h172.20.249.50 -uroot@obmysql#obdemo -P2883 -pbJVqqEVt -c -A --default-auth=mysql_native_passwo
rd  oceanbase
```

说明

OBProxy 2.0 版本已经修复了这个问题。

OBClient 客户端连接

OceanBase 数据库提供了专用的命令行客户端工具 `obclient`。使用方法和使用 MySQL 客户端一样。

示例：

```
obclient -h172.20.249.50 -uroot@obmysql#obdemo -P2883 -pbJ****Vt -c -A oceanbase

Welcome to the OceanBase.  Commands end with ; or \g.
Your OceanBase connection id is 17
Server version: 5.6.25 OceanBase 3.1.0 (r3-b20901e8c84d3ea774beeaca963c67d7802e4b4e) (Built Aug 10 202
1 08:10:38)

Copyright (c) 2000, 2022, OceanBase and/or its affiliates. All rights reserved.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

obclient [oceanbase]> show databases;
+--------------------+
```

```
| Database           |
+--------------------+
| oceanbase          |
| information_schema |
| mysql              |
| test               |
+--------------------+
4 rows in set (0.004 sec)
```

OceanBase 连接驱动（JDBC）

OceanBase 数据库目前支持的应用主要是 `Java` 和 `C/C++` 。

- Java 语言

  MySQL 官方 JDBC 驱动下载地址：MySQL Connector/J 5.1.46

  OceanBase 官方 JDBC 驱动下载地址：OceanBase-Client

- C 语言

  具体驱动说明请参考官网文档：OceanBase Connector/C 简介

  下载地址：OceanBase Connector/C 下载

DBEAVER 客户端连接

DBeaver 是一款通用的数据库客户端工具，其原理是使用各个数据库提供的 JDBC 驱动连接数据库，支持常见的关系型数据库、非关系型数据库、分布式数据库等等。使用 OceanBase 提供的 JDBC 驱动或者 MySQL 官方驱动，DBeaver 也可以连接 OceanBase 数据库的 MySQL 租户。

官方下载地址：https://dbeaver.io/download/

DBeaver 连接 OceanBase 数据库的时候选择 MySQL 数据库类型。第一次使用会自动下载官方 MySQL 驱动。与 MySQL 不同， DBeaver 连接端口默认是 `2883`，用户名格式是：**`用户名@租户名#集群名`** 或 **`集群名:租户名:用户名`**。

ODC 客户端连接

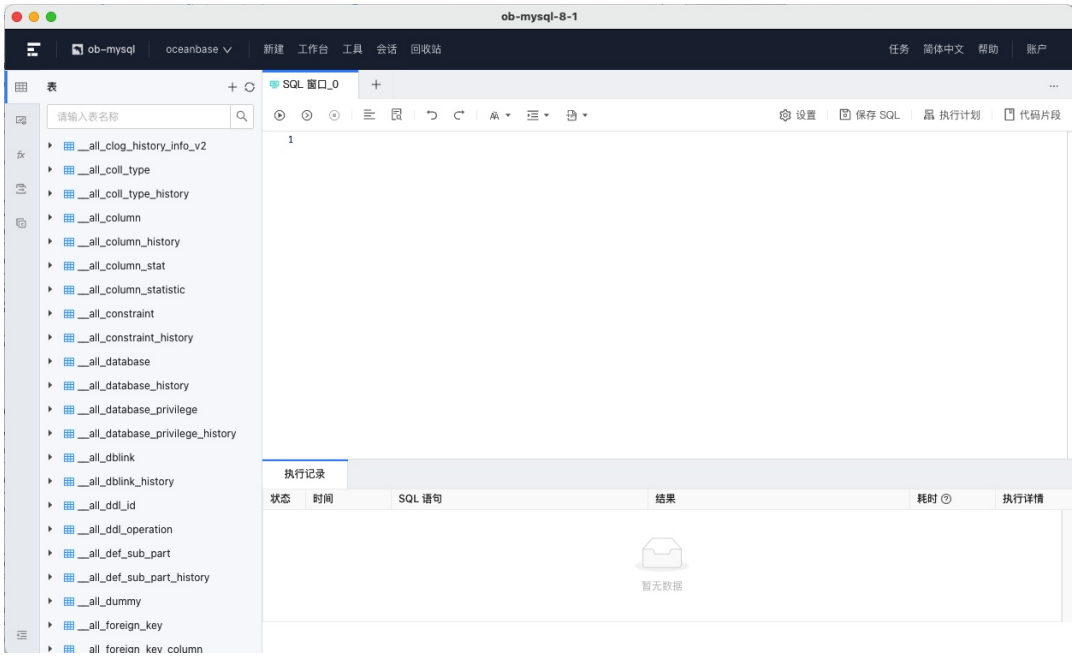
OceanBase 提供官方图形化客户端工具 `OceanBase Developer Center`，简称 `ODC`。该工具的详细信息请参考官网文档 OceanBase 开发者中心 。

ODC 下载地址：下载客户端版 ODC 。

适用 ODC 连接 OceanBase 数据库的方法如下：


新建连接
连接名称
ob-mysql
设置标签
连接地址：
* 主机名
172.20.249.50
* 端口
2883
集群
obdemo
* 租户
obmysql
默认数据库（选填）
oceanbase
* 数据库用户名
root
数据库密码
测试连接
智能解析
mysql -h172.20.249.50 -uroot@obmysql#obdemo -P2883 -pbJVqqEVt -c -A -D oceanbase
智能解析
SQL 查询超时时间
复制连接串
取 消
保 存

保存连接后，打开连接，即进入工作界面。



目前 ODC 是对 OceanBase 数据库适配性最好的客户端工具。

3.4 如何对租户参数（或变量）进行设置

通过 SYS 租户修改业务租户参数

上一节介绍了 OceanBase 集群参数设置，其中有部分参数生效范围是租户（`TENANT`）。在 OceanBase 内部租户（`sys`）里，可以修改业务实例的部分参数。比如参数 `writing_throttling_trigger_percentage`，用于对指定租户进行内存限流（增量内存使用率达到这个阈值就对写入降速）。

```
MySQL [oceanbase]> show parameters like 'writing_throttling_trigger_percentage%'\G
*************************** 1. row ***************************
      zone: zone1
  svr_type: observer
    svr_ip: 172.20.249.50
  svr_port: 2882
      name: writing_throttling_trigger_percentage
 data_type: NULL
     value: 100
      info: the threshold of the size of the mem store when writing_limit will be triggered. Ran
   section: TRANS
     scope: TENANT
    source: DEFAULT
edit_level: DYNAMIC_EFFECTIVE
1 row in set (0.002 sec)

MySQL [oceanbase]> alter system set writing_throttling_trigger_percentage = 90 tenant='obmysq
Query OK, 0 rows affected (0.011 sec)
```

修改后的参数值只能在对应租户里查看。

```
$ mysql -h 172.20.249.50 -u root@obmysql -P 2881 -p -c -A oceanbase
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 3221538749
Server version: 5.7.25 OceanBase 3.1.0 (r3-b20901e8c84d3ea774beeaca963c67d7802e4b4e) (Built Aug

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [oceanbase]> show parameters like 'writing_throttling_trigger_percentage%'\G
*************************** 1. row ***************************
      zone: zone1
  svr_type: observer
    svr_ip: 172.20.249.50
  svr_port: 2882
      name: writing_throttling_trigger_percentage
 data_type: NULL
     value: 90
      info: the threshold of the size of the mem store when writing_limit will be triggered. Ran
   section: TRANS
```

```
     scope: TENANT
    source: DEFAULT
edit_level: DYNAMIC_EFFECTIVE
1 row in set (0.004 sec)
```

修改业务租户参数

在业务租户里，可以自己设置参数。比如，参数 `writing_throttling_maximum_duration`，用于控制增量内存的剩余内存根据当前写入速度的最长写入时间。触发写入限速后，剩余 `memstore` 的内存量预期在 `writing_throttling_maximum_duration` 时间内分配完。

说明

该参数仅供参考，准确性不及参数 `writing_throttling_trigger_percentage`。

```
# 在业务租户里修改参数，后面就不需要指定租户名。

MySQL [oceanbase]> alter system set writing_throttling_maximum_duration = '2h';
Query OK, 0 rows affected (0.006 sec)

MySQL [oceanbase]> show parameters like 'writing_throttling_maximum_duration'\G
*************************** 1. row ***************************
      zone: zone1
  svr_type: observer
    svr_ip: 172.20.249.50
  svr_port: 2882
      name: writing_throttling_maximum_duration
 data_type: NULL
     value: 2h
      info: maximum duration of writing throttling(in minutes), max value is 3 days
   section: TRANS
     scope: TENANT
    source: DEFAULT
edit_level: DYNAMIC_EFFECTIVE
1 row in set (0.004 sec)
```

修改业务租户变量

OceanBase 租户还有一个名为变量（`VARIABLE`）的设计，这个和 MySQL 实例很像。变量其实就是租户的参数。可以在租户全局层面修改，也可以在会话层面修改，很多变量和对应的 SQL HINT 还可在语句级别修改。

全局层面的修改影响的是后续的会话，会话层面的修改仅影响当前会话，语句级别的修改只影响当前语句。

初次使用 OceanBase 租户时，建议调整租户的几个超时参数。

- `ob_query_timeout`：语句执行超时时间，单位 `us`，默认值是 `10000000` （即 10s）。建议根据业务 SQL 的平均执行时间水平调整。OLTP 场景调整小一些，OLAP 场景调整大一些。初学者建议调大 10 倍。

- `ob_trx_idle_timeout`：事务空闲超时时间，单位 `us`，默认值是 `120000000`（即 120s）。建议根据业务事务平均空闲时间水平调整。空闲事务会占用连接，并可能持有锁不释放，导致高并发时阻塞和死锁概率增加，不建议调大。
- `ob_trx_timeout`：事务未提交超时时间，单位 `us`，默认值是 `100000000`（即 100s）。建议根据业务事务平均持续时间水平调整。事务长期不提交，会占用连接、可能持有锁不释放，导致高并发时阻塞和死锁概率增加，不建议调大。如果是后台跑批业务，建议在会话级别调大。
- `ob_trx_lock_timeout`：事务申请加锁等待超时时间，单位 `us`，默认值是 `-1`，即不控制。超时依然会受 `ob_query_timeout` 限制。当调大语句超时时间变量（`ob_query_timeout`）后，可以将这个锁等待超时改为 10000000 （即 10s），以减少阻塞和死锁的概率。

您可运行以下命令查看和修改变量：

```
show global | session variables like '%变量名部分字段%' ;

set global | session 变量名 =  '变量值' ;
```

示例：

```
MySQL [oceanbase]> show global variables like '%timeout%';
+---------------------+------------------+
| Variable_name       | Value            |
+---------------------+------------------+
| connect_timeout     | 10               |
| interactive_timeout | 28800            |
| lock_wait_timeout   | 31536000         |
| net_read_timeout    | 30               |
| net_write_timeout   | 60               |
| ob_pl_block_timeout | 3216672000000000 |
| ob_query_timeout    | 10000000         |
| ob_trx_idle_timeout | 120000000        |
| ob_trx_lock_timeout | -1               |
| ob_trx_timeout      | 100000000        |
| wait_timeout        | 28800            |
+---------------------+------------------+
11 rows in set (0.002 sec)

MySQL [oceanbase]> set global ob_query_timeout = 100000000;
Query OK, 0 rows affected (0.015 sec)

MySQL [oceanbase]> set global ob_trx_timeout = 1000000000;
Query OK, 0 rows affected (0.014 sec)

MySQL [oceanbase]> set global ob_trx_idle_timeout = 1200000000;
Query OK, 0 rows affected (0.010 sec)

MySQL [oceanbase]> SET GLOBAL ob_trx_lock_timeout=10000000;
Query OK, 0 rows affected (0.011 sec)
```

对于复杂的 SQL 场景或者 OLAP 场景，租户还需要调整 `ob_sql_work_area_percentage` 变量。该变量影响 SQL 里

排序统计能利用的内存大小，可以根据情况进行调整。

```
set global ob_sql_work_area_percentage=50;
```

通过 SYS 租户修改业务租户变量

注意

部分变量属于租户初始化变量，不能在业务租户里直接修改，需要在 sys 租户里修改。

示例:

```
$ mysql -h127.1 -uroot@obmysql#obdemo -P2883 -p1****6 -c -A oceanbase -Ns
MySQL [oceanbase]> set global lower_case_table_names=0;
ERROR 1238 (HY000): Variable 'lower_case_table_names' is a read only variable

$mysql -h127.1 -uroot@sys#obdemo -P2883 -p1****6 -c -A oceanbase -Ns
MySQL [oceanbase]> alter tenant obmysql set variables lower_case_table_names=0;

$ mysql -h127.1 -uroot@obmysql#obdemo -P2883 -p1****6 -c -A oceanbase -Ns
MySQL [oceanbase]> show global variables like 'lower_case_table_names';
lower_case_table_names  0
```

有些变量比较特殊，比如:

- 变量 `ob_tcp_invited_nodes`，表示租户访问 IP 白名单。初始化租户的时候在 sys 租户中设置，后期可以在业务租户里修改。

  ```
  set global ob_tcp_invited_nodes='11.xxx.xxx.0/16,127.0.0.1';
  ```

  如果业务租户设置错误导致无法登录，可以通过 sys 租户再改回正确值。

- 变量 `ob_compatibility_mode` 表示租户兼容性。这个在租户创建时指定，后期不能修改。

3.5 如何使用 MySQL 租户做常见数据库开发

用户管理

这里的用户指租户里的用户，和传统 MySQL 里的用户概念一样。 OceanBase MySQL 租户创建用户有两个方法:

- `create user` 创建用户。
- `grant` 语句自动创建用户。

示例:

```
MySQL [oceanbase]> create user user01 identified by 'zf******MG';
Query OK, 0 rows affected (0.024 sec)

MySQL [oceanbase]> grant all privileges on test.* to user01 ;
Query OK, 0 rows affected (0.013 sec)

MySQL [oceanbase]> grant all privileges on test.* to user02 identified by 'dQ******M8';
Query OK, 0 rows affected (0.028 sec)
```

OceanBase MySQL 租户不支持更新用户元数据的密码字段。

您可使用 `show grants` 语句查看用户权限。

示例:

```
MySQL [oceanbase]> show grants for user01;
+-----------------------------------------------+
| Grants for user01@%                           |
+-----------------------------------------------+
| GRANT USAGE ON *.* TO 'user01'                |
| GRANT ALL PRIVILEGES ON `test`.* TO 'user01' |
+-----------------------------------------------+
2 rows in set (0.001 sec)

MySQL [oceanbase]> show grants for user02;
+-----------------------------------------------+
| Grants for user02@%                           |
+-----------------------------------------------+
| GRANT USAGE ON *.* TO 'user02'                |
| GRANT ALL PRIVILEGES ON `test`.* TO 'user02' |
+-----------------------------------------------+
2 rows in set (0.001 sec)
```

数据库管理

OceanBase MySQL 租户下可以创建多个数据库（`database`），表只能在具体的数据库下新建。

示例:

```
# 创建数据库 tpccdb

MySQL [test]> create database tpccdb;
Query OK, 1 row affected (0.012 sec)

MySQL [test]> show databases;
+--------------------+
| Database           |
+--------------------+
| oceanbase          |
| information_schema |
| mysql              |
| test               |
| tpccdb             |
+--------------------+
5 rows in set (0.002 sec)

# 在 test 数据库下建表

create table ware(w_id int
   , w_ytd decimal(12,2)
    , w_tax decimal(4,4)
    , w_name varchar(10)
    , w_street_1 varchar(20)
    , w_street_2 varchar(20)
    , w_city varchar(20)
    , w_state char(2)
    , w_zip char(9)
    , unique(w_name, w_city)
    , primary key(w_id)
    );

create table cust (c_w_id int NOT NULL
     , c_d_id int NOT null
     , c_id int NOT null
     , c_discount decimal(4, 4)
     , c_credit char(2)
     , c_last varchar(16)
     , c_first varchar(16)
     , c_middle char(2)
     , c_balance decimal(12, 2)
     , c_ytd_payment decimal(12, 2)
     , c_payment_cnt int
     , c_credit_lim decimal(12, 2)
     , c_street_1 varchar(20)
     , c_street_2 varchar(20)
     , c_city varchar(20)
     , c_state char(2)
     , c_zip char(9)
     , c_phone char(16)
     , c_since date
     , c_delivery_cnt int
     , c_data varchar(500)
```

```
      , index icust(c_last, c_d_id, c_w_id, c_first, c_id)
      , FOREIGN KEY (c_w_id) REFERENCES ware(w_id)
      , primary key (c_w_id, c_d_id, c_id)
      );
```

OceanBase MySQL 租户支持外键。不过在分布式数据库里，如果读写并发很高，不建议在数据库层面使用外键约束。外键会增加不必要的阻塞和死锁，可能会给性能带来负面影响。

复制表的结构使用 `like`，表的结构中包括主键、唯一键、索引名称都会复制。在 MySQL 语法里，主键名、唯一约束和索引名在一个表内不能重复，但是不同表之间可以重复。

```
create table t1 like ware;
```

复制表的结构和数据使用 `create table ... as select`。不过要注意的是，该语句复制的是表的基本数据类型，对于主键、唯一约束和索引信息等不会复制。

```
create table t2 as select * from ware;

MySQL [test]> show create table t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `w_id` int(11) NOT NULL,
  `w_ytd` decimal(12,2) DEFAULT NULL,
  `w_tax` decimal(4,4) DEFAULT NULL,
  `w_name` varchar(10) DEFAULT NULL,
  `w_street_1` varchar(20) DEFAULT NULL,
  `w_street_2` varchar(20) DEFAULT NULL,
  `w_city` varchar(20) DEFAULT NULL,
  `w_state` char(2) DEFAULT NULL,
  `w_zip` char(9) DEFAULT NULL,
  PRIMARY KEY (`w_id`),
  UNIQUE KEY `w_name` (`w_name`, `w_city`) BLOCK_SIZE 16384 GLOBAL
) DEFAULT CHARSET = utf8mb4 ROW_FORMAT = COMPACT COMPRESSION = 'zstd_1.3.8' REPLICA_NUM = 1 BLOC
1 row in set (0.003 sec)

MySQL [test]> show create table t2\G
*************************** 1. row ***************************
       Table: t2
Create Table: CREATE TABLE `t2` (
  `w_id` int(11) NOT NULL,
  `w_ytd` decimal(12,2) DEFAULT NULL,
  `w_tax` decimal(4,4) DEFAULT NULL,
  `w_name` varchar(10) DEFAULT NULL,
  `w_street_1` varchar(20) DEFAULT NULL,
  `w_street_2` varchar(20) DEFAULT NULL,
  `w_city` varchar(20) DEFAULT NULL,
  `w_state` char(2) DEFAULT NULL,
  `w_zip` char(9) DEFAULT NULL
) DEFAULT CHARSET = utf8mb4 ROW_FORMAT = COMPACT COMPRESSION = 'zstd_1.3.8' REPLICA_NUM = 1 BLOC
1 row in set (0.002 sec)
```

观察上面两个表结构可以看出：主键、唯一键和索引不同。

3.6 如何使用 OceanBase 分区表进行水平拆分

分区技术（Partitioning）是 OceanBase 数据库非常重要的分布式能力之一，它能解决大表的容量问题和高并发访问时的性能问题。分区技术将大表拆分为更多更小的结构相同的独立对象，即分区。普通的表只有一个分区，可以看作分区表的特例。每个分区只能存在于一个节点内部，分区表的不同分区可以分散在不同节点上。

分区路由

OceanBase 数据库的分区表是内建功能，您只需要在建表的时候指定分区策略和分区数即可。分区表的查询 SQL 跟普通表是一样的，OceanBase 的 OBProxy 或 OBServer 会自动将用户 SQL 路由到相应节点内。因此，分区表的分区细节对业务是透明的。

如果知道要读取的数据所在的分区号，可以通过 SQL 直接访问分区表的某个分区。简单语法格式如下:

```
part_table partition ( p[0,1,...][sp[0,1,...]] )
```

除了表定义了分区名这一特殊情况，默认情况下，分区名都是按一定规则编号，例如:

一级分区名为：p0, p1, p2, ... 二级分区名为：p0sp0, p0sp1, p0sp2, ... ; p1sp0, p1sp1, p1sp2, ...

示例：访问分区表的具体分区。

```
select * from t1 partition (p0) ;

select * from t1 partition (p5sp0) ;
```

分区策略

OceanBase 数据库支持多种分区策略:

- 范围（`RANGE`）分区
- `RANGE COLUMNS` 分区
- 列表（`LIST`）分区
- `LIST COLUMNS` 分区
- 哈希（`HASH`）分区
- 组合分区

范围（RANGE）分区

RANGE 分区根据分区表定义时为每个分区建立的分区键值范围，将数据映射到相应的分区中。它是常见的分区类

型，经常和日期类型一起使用。比如，可以将业务日志表按日/周/月分区。

RANGE 分区的语法格式如下:

```
CREATE TABLE table_name (
    column_name1        column_type
    [, column_nameN     column_type]
) PARTITION BY RANGE ( expr(column_name1) )
(
    PARTITION   p0      VALUES LESS THAN ( expr )
    [, PARTITION pN     VALUES LESS THAN (expr ) ]
 [, PARTITION pX    VALUES LESS THAN (maxvalue) ]
);
```

当使用 RANGE 分区时，需要遵守如下几个规则:

- `PARTITION BY RANGE ( expr )` 中的 expr 表达式的结果必须为整型。
- 每个分区都有一个 VALUES LESS THAN 子句，它为分区指定一个非包含的上限值。分区键中等于或大于这个上限值将被映射到下一个分区中。
- 除第一个分区外，所有分区都隐含一个下限值，即上一个分区的上限值。
- 允许且只允许最后一个分区上限定义为 `MAXVALUE`，该值没有具体的数值，比其他所有分区的上限都要大，也包含空值。

注意

RANGE 分区可以新增、删除分区。如果最后一个 RANGE 分区指定了 `MAXVALUE`，则不能新增分区。所以建议不要使用 `MAXVALUE` 定义最后一个分区。

`RANGE` 分区要求表拆分键表达式的结果必须为整型，如果要按时间类型列做 `RANGE` 分区，则必须使用 `timestamp` 类型，并且使用函数 `UNIX_TIMESTAMP` 将时间类型转换为数值。这个需求也可以使用 `RANGE COLUMNS` 分区实现，则没有整型的限制。

示例:

```
CREATE TABLE test_range(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('2015-01-01 00:00:00')),
PARTITION p1 VALUES LESS THAN (UNIX_TIMESTAMP('2016-01-01 00:00:00')),
PARTITION p2 VALUES LESS THAN (UNIX_TIMESTAMP('2017-01-01 00:00:00')));
```

哈希（HASH）分区

HASH 分区适合于不能使用 RANGE 分区、LIST 分区方法的场景。HASH 分区的实现方法简单，通过对分区键上的 HASH 函数值来散列记录到不同分区中。如果您的数据符合下列特点，使用 HASH 分区是个很好的选择:

- 不能指定数据的分区键的列表特征。

- 不同范围内的数据大小相差非常大，并且很难手动调整均衡。
- 使用 RANGE 分区后数据聚集严重。
- 并行 DML、分区剪枝和分区连接等性能非常重要。

示例：创建一个 `HASH` 分区表。

```
CREATE TABLE ware2(
    w_id int
    , w_ytd number(12,2)
    , w_tax number(4,4)
    , w_name varchar(10)
    , w_street_1 varchar(20)
    , w_street_2 varchar(20)
    , w_city varchar(20)
    , w_state char(2)
    , w_zip char(9)
    , primary key(w_id)
) PARTITION by hash(w_id) partitions 60;
```

HASH 分区不支持新增或删除分区。

组合分区（二级分区）

组合分区通常是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，适合于业务表的数据量非常大时。使用组合分区能发挥多种分区策略的优点。

在指定二级分区分区策略细节时，可以使用 `SUBPARTITION TEMPLATE` 子句。

示例：创建组合分区表。

```
CREATE TABLE t_ordr_part_by_hash_range (o_w_id int
, o_d_id int
, o_id int
, o_c_id int
, o_carrier_id int
, o_ol_cnt int
, o_all_local int
, o_entry_d TIMESTAMP NOT NULL
, index idx_ordr(o_w_id, o_d_id, o_c_id, o_id) LOCAL
, primary key ( o_w_id, o_d_id, o_id, o_entry_d )
)
PARTITION BY hash(o_w_id)
SUBPARTITION BY RANGE(UNIX_TIMESTAMP(o_entry_d))
SUBPARTITION template
(
    SUBPARTITION M202001 VALUES LESS THAN(UNIX_TIMESTAMP('2020/02/01'))
    , SUBPARTITION M202002 VALUES LESS THAN(UNIX_TIMESTAMP('2020/03/01'))
    , SUBPARTITION M202003 VALUES LESS THAN(UNIX_TIMESTAMP('2020/04/01'))
    , SUBPARTITION M202004 VALUES LESS THAN(UNIX_TIMESTAMP('2020/05/01'))
    , SUBPARTITION M202005 VALUES LESS THAN(UNIX_TIMESTAMP('2020/06/01'))
    , SUBPARTITION M202006 VALUES LESS THAN(UNIX_TIMESTAMP('2020/07/01'))
    , SUBPARTITION M202007 VALUES LESS THAN(UNIX_TIMESTAMP('2020/08/01'))
```

```
    , SUBPARTITION M202008 VALUES LESS THAN(UNIX_TIMESTAMP('2020/09/01'))
    , SUBPARTITION M202009 VALUES LESS THAN(UNIX_TIMESTAMP('2020/10/01'))
    , SUBPARTITION M202010 VALUES LESS THAN(UNIX_TIMESTAMP('2020/11/01'))
    , SUBPARTITION M202011 VALUES LESS THAN(UNIX_TIMESTAMP('2020/12/01'))
    , SUBPARTITION M202012 VALUES LESS THAN(UNIX_TIMESTAMP('2021/01/01'))
)
partitions 16;

CREATE TABLE t_log_part_by_range_hash (
    log_id      int NOT NULL
    , log_value varchar(50)
    , log_date  TIMESTAMP NOT NULL
    , PRIMARY key(log_id, log_date)
) PARTITION BY RANGE(UNIX_TIMESTAMP(log_date))
SUBPARTITION BY HASH(log_id) SUBPARTITIONS 16
(
    PARTITION M202001 VALUES LESS THAN(UNIX_TIMESTAMP('2020/02/01'))
    , PARTITION M202002 VALUES LESS THAN(UNIX_TIMESTAMP('2020/03/01'))
    , PARTITION M202003 VALUES LESS THAN(UNIX_TIMESTAMP('2020/04/01'))
    , PARTITION M202004 VALUES LESS THAN(UNIX_TIMESTAMP('2020/05/01'))
    , PARTITION M202005 VALUES LESS THAN(UNIX_TIMESTAMP('2020/06/01'))
    , PARTITION M202006 VALUES LESS THAN(UNIX_TIMESTAMP('2020/07/01'))
    , PARTITION M202007 VALUES LESS THAN(UNIX_TIMESTAMP('2020/08/01'))
    , PARTITION M202008 VALUES LESS THAN(UNIX_TIMESTAMP('2020/09/01'))
    , PARTITION M202009 VALUES LESS THAN(UNIX_TIMESTAMP('2020/10/01'))
    , PARTITION M202010 VALUES LESS THAN(UNIX_TIMESTAMP('2020/11/01'))
    , PARTITION M202011 VALUES LESS THAN(UNIX_TIMESTAMP('2020/12/01'))
    , PARTITION M202012 VALUES LESS THAN(UNIX_TIMESTAMP('2021/01/01'))
);
```

尽管 OceanBase 数据库在组合分区上支持 `RANGE + HASH` 和 `HASH + RANGE` 两种组合，但是对于一个流水大表，为了维护方便（新增和删除分区），建议使用 `RANGE + HASH` 组合方式。

分区表的索引

分区表的查询性能跟 SQL 的条件有关。当 SQL 带上拆分键时，OceanBase 数据库会根据条件做分区检索，只搜索特定的分区；如果没有拆分键，则要扫描所有分区。

分区表也可以通过创建索引来提升性能。跟分区表一样，分区表的索引也可以分区或者不分区。

- 如果分区表的索引不分区，就是一个全局索引（`GLOBAL`），是一个独立的分区，索引数据覆盖整个分区表。
- 如果分区表的索引分区，根据分区策略又可以分为两类。
  - 一类是跟分区表保持一致的分区策略，则每个索引分区的索引数据覆盖相应的分区表的分区，这个索引又叫本地索引（`LOCAL`）。
  - 另一类则是分区策略和分区表不一致的全局索引（`GLOBAL`）。

注意

通常创建索引时默认都是全局索引，本地索引需要在后面增加关键字 `LOCAL`。 建议尽可能的使用本地索引，只在必要时使用全局索引。因为全局索引会降低 DML 的性能，DML 可能会因此产生分布式事务。

示例：创建分区表的本地索引和全局索引。

```
CREATE INDEX idx_log_date ON t_log_part_by_range_hash(log_date) LOCAL;

CREATE INDEX idx_log_date2 ON t_log_part_by_range_hash(log_value, log_date) GLOBAL;
```

注意

OceanBase 数据库的分区表主键和唯一键不需要单独建索引。OceanBase 分区表的一个功能限制是如果分区表有主键，主键必须包含分区键。

分区表的实践

通常当表的数据量非常大，以致于可能使数据库空间紧张，或者导致相关 SQL 查询性能变慢时，可以考虑使用分区表。

使用分区表时要选择合适的拆分键以及拆分策略。

- 如果是日志类型的大表，则根据时间类型的列做 RANGE 分区。
- 如果是并发访问非常高的表，则结合业务特点选择能满足绝大部分核心业务查询的列作为拆分键。无论选哪个列做为分区键，都不大可能满足所有的查询性能。

分区表中的全局唯一性需求可以通过主键约束和唯一约束实现。OceanBase 数据库的分区表的主键约束必须包含拆分键。唯一约束是一个全局索引。全局唯一的需求可以通过本地唯一索引实现（在唯一索引里包含拆分键），也可以通过全局索引实现。

```
CREATE TABLE account(
         id bigint NOT NULL PRIMARY KEY
         , name varchar(50) NOT NULL UNIQUE
         , value number NOT NULL
         , gmt_create timestamp DEFAULT current_timestamp NOT NULL
         , gmt_modified timestamp DEFAULT current_timestamp NOT NULL
     ) PARTITION BY HASH(id) PARTITIONS 16;

CREATE TABLE account2(
         id bigint NOT NULL PRIMARY KEY
         , name varchar(50) NOT NULL
         , value number NOT NULL
         , gmt_create timestamp DEFAULT current_timestamp NOT NULL
         , gmt_modified timestamp DEFAULT current_timestamp NOT NULL
     ) PARTITION BY HASH(id) PARTITIONS 16;


CREATE UNIQUE INDEX account2_uk ON account2(name, id) LOCAL ;

CREATE UNIQUE INDEX t_log_part_by_range_hash_uk2 on t_log_part_by_range_hash(log_value);


obclient> show indexes from account;
+---------+------------+----------+--------------+-------------+-----------+-------------+------
```

```
| Table   | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_p
+---------+------------+----------+--------------+-------------+-----------+-------------+------
| account |          0 | PRIMARY  |            1 | id          | A         |        NULL | NULL
| account |          0 | name     |            1 | name        | A         |        NULL | NULL
+---------+------------+----------+--------------+-------------+-----------+-------------+------
2 rows in set (0.002 sec)

obclient> show indexes from account2;
+----------+------------+-------------+--------------+-------------+-----------+-------------+--
| Table    | Non_unique | Key_name    | Seq_in_index | Column_name | Collation | Cardinality | S
+----------+------------+-------------+--------------+-------------+-----------+-------------+--
| account2 |          0 | PRIMARY     |            1 | id          | A         |        NULL | N
| account2 |          0 | account2_uk |            1 | name        | A         |        NULL | N
| account2 |          0 | account2_uk |            2 | id          | A         |        NULL | N
+----------+------------+-------------+--------------+-------------+-----------+-------------+--
3 rows in set (0.002 sec)

obclient> SELECT * FROM information_schema.`TABLE_CONSTRAINTS` WHERE table_schema='TEST' AND
+--------------------+-------------------+-----------------+--------------+------------+--------
| CONSTRAINT_CATALOG | CONSTRAINT_SCHEMA | CONSTRAINT_NAME | TABLE_SCHEMA | TABLE_NAME | CONSTRA
+--------------------+-------------------+-----------------+--------------+------------+--------
| def                | test              | PRIMARY         | test         | account    | PRIMARY
| def                | test              | name            | test         | account    | UNIQUE
| def                | test              | PRIMARY         | test         | account2   | PRIMARY
| def                | test              | account2_uk     | test         | account2   | UNIQUE
+--------------------+-------------------+-----------------+--------------+------------+--------
4 rows in set (0.001 sec)
```

3.7 （高级）如何使用 OceanBase 表分组

表分组简介

表分组（`TABLE GROUP`）是 OceanBase 数据库作为分布式数据库的一个特色功能。表分组是表的属性，会影响多个表的分区在 OceanBase 机器上的分布特征。

不同表的分区有可能分布在不同的节点上，当两个表做表连接查询时，OceanBase 数据库会跨节点请求数据，执行时间跟节点间请求延时有关。

在 SQL 调优时，OceanBase 建议对业务上关系密切的表，设置相同的表分组。OceanBase 对于同一个表分组中的表的同号分区会管理为一个分区组。同一个分区组中的分区，OceanBase 会尽可能的分配到同一个节点内部，这样就可以规避跨节点的请求。

创建表分组

创建表分组时，首先要规划好表分组的用途。

- 如果是用于普通表，表分组就不用分区。
- 如果是用于分区表，表分组就要指定分区策略，并且要跟分区表的分区策略保持一致。

示例：创建表分组和查看表分组。

```
MySQL [test]> create tablegroup tpcc_group partition by hash partitions 6 ;

MySQL [test]> show tablegroups;
+-----------------+------------+---------------+
| Tablegroup_name | Table_name | Database_name |
+-----------------+------------+---------------+
| oceanbase       | NULL       | NULL          |
| tpcc_group      | NULL       | NULL          |
+-----------------+------------+---------------+
2 rows in set (0.004 sec)

MySQL [test]> show create tablegroup tpcc_group;
+------------+-----------------------------------------------------------------------------------------
| Tablegroup | Create Tablegroup
+------------+-----------------------------------------------------------------------------------------
| tpcc_group | CREATE TABLEGROUP IF NOT EXISTS `tpcc_group`  BINDING = FALSE
  partition by hash partitions 6
 |
+------------+-----------------------------------------------------------------------------------------
1 row in set (0.001 sec)
```

示例中的 `show tablegroups;` 语句作用就是查看表分组内的表。

有了表分组后，在建表时就可以指定表分组。

```
create table ordr (
    o_w_id int
    , o_d_id int
    , o_id int
    , o_c_id int
    , o_carrier_id int
    , o_ol_cnt int
    , o_all_local int
    , o_entry_d date
    , index iordr(o_w_id, o_d_id, o_c_id, o_id) local
    , primary key ( o_w_id, o_d_id, o_id )
)tablegroup tpcc_group partition by hash(o_w_id) partitions 6;
create table ordl (
    ol_w_id int
    , ol_d_id int
    , ol_o_id int
    , ol_number int
    , ol_delivery_d date
    , ol_amount decimal(6, 2)
    , ol_i_id int
    , ol_supply_w_id int
    , ol_quantity int
    , ol_dist_info char(24)
    , primary key (ol_w_id, ol_d_id, ol_o_id, ol_number )
)tablegroup tpcc_group partition by hash(ol_w_id) partitions 6;
```

查看建表后的表分组:

```
MySQL [test]> show tablegroups;
+-----------------+------------+---------------+
| Tablegroup_name | Table_name | Database_name |
+-----------------+------------+---------------+
| oceanbase       | NULL       | NULL          |
| tpcc_group      | ordl       | test          |
| tpcc_group      | ordr       | test          |
+-----------------+------------+---------------+
3 rows in set (0.004 sec)
```

您也可以后期使用语句 `alter tablegroup ... add` 将一个表加入到表分组， 使用语句 `alter table ... tablegroup = '';` 将表从表分组中移出。

```
MySQL [test]> alter table ordl tablegroup = '';
Query OK, 0 rows affected (0.148 sec)

MySQL [test]> alter table ordr tablegroup = '';
Query OK, 0 rows affected (0.018 sec)

MySQL [test]> show tablegroups;
+-----------------+------------+---------------+
| Tablegroup_name | Table_name | Database_name |
+-----------------+------------+---------------+
| oceanbase       | NULL       | NULL          |
| tpcc_group      | NULL       | NULL          |
+-----------------+------------+---------------+
```

```
2 rows in set (0.004 sec)

MySQL [test]> alter tablegroup tpcc_group add ordl , ordr ;
Query OK, 0 rows affected (0.016 sec)

MySQL [test]> show tablegroups;
+-----------------+------------+---------------+
| Tablegroup_name | Table_name | Database_name |
+-----------------+------------+---------------+
| oceanbase       | NULL       | NULL          |
| tpcc_group      | ordl       | test          |
| tpcc_group      | ordr       | test          |
+-----------------+------------+---------------+
3 rows in set (0.004 sec)
```

3.8（高级）如何使用 OceanBase 复制表

复制表原理

复制表指的是一种特殊的表。在生产环境中，普通的表默认为三副本，即一个主副本和两个备副本。备副本通过同步主副本的事务日志 `clog` 保持同步，同步协议是 Paxos 协议。

主副本的事务日志只有在多数成员确认落盘后，事务修改才会生效。默认情况下，读写都是在主副本上，备副本不提供读写服务。应用开启会话或语句级别的弱一致性读时，备副本可能会提供只读服务。但是备副本的读会有些许延迟。

普通表可以变为复制表，然后主副本和所有备副本之间使用全同步协议。主副本的事务日志只有在所有副本成员确认落盘后，事务修改才会生效。所以主副本跟所有备副本的数据理论上都是强一致的。

复制表场景

传统普通的表，主副本可能会成为读写瓶颈，此时业务会使用读写分离技术，将只读查询分离出去，运维将只读查询路由到备副本。该操作的风险在于备副本理论上有一定延时。

如果表是复制表，则备副本没有延迟问题。这是复制表的一种使用场景，前提是开启弱一致性读。

复制表最有用的场景是业务数据库做了水平拆分后，有部分业务表不适合拆分。前者的数据主副本有可能在所有机器上，后者的主副本只会在某台机器上。OceanBase 数据库里一个事务的 SQL 都会跟随到事务开始时那条 SQL 的路由，如果某个 SQL 被路由到的节点不是该 SQL 访问的分区的主副本节点，这个 SQL 就属于远程 SQL。如果这个分区所在的表是复制表，则这条 SQL 就会在本机执行，从而提升性能。

复制表使用的前提是表的修改频率不能太高，每个事务的平均延时比普通表的事务延时要大。

复制表语法

您可以在创建表时就指定复制表属性 `DUPLICATE_SCOPE`。复制表属性有以下两个值：

- `NONE`：这个是默认值，表示是普通的表。
- `CLUSTER`：表的备副本分布在租户资源池所在的所有机器上。

示例：

```
mysql> create table t1(id bigint not null auto_increment , c1 varchar(50), c2 timestamp not null defau
lt current_timestamp) duplicate_scope='cluster' ;                                                  Qu
ery OK, 0 rows affected (0.12 sec)
```

当然，您也可以在表创建好后修改复制表属性，具体参考以下示例。

```
mysql> alter table t1 duplicate_scope = 'NONE';
Query OK, 0 rows affected (0.05 sec)

mysql> alter table t1 duplicate_scope = 'CLUSTER';
Query OK, 0 rows affected (0.04 sec)
```

3.9 常见问题

租户创建或扩容时提示机器资源不足

- 现象

  在创建资源池时，或者在调整资源池资源规格时，遇到报错，信息如下:

```
ERROR 4624 (HY000):  machine resource 'zone1' is not enough to hold a new unit
```

- 原因

  报错信息提示某个 ZONE 没有足够的资源创建该资源单元。

  出现该报错通常是资源单元规格超出了集群资源可用资源。所以创建资源池或者调整资源池之前需要计算集群可用资源。

- 解决方法

  您可通过查看视图 `__all_virtual_server_stat` 了解集群剩余可用资源。 由于默认 sys 租户的资源规格的 `min_cpu` 和 `min_memory` 与对应的 `max_cpu` 和 `max_memory` 不一致，所以集群剩余资源的展示将会不准确。

  查看剩余资源的 SQL:

```
select a.zone,concat(a.svr_ip,':',a.svr_port) observer, cpu_total, cpu_assigned, (cpu_total-cpu_
assigned) cpu_free, mem_total/1024/1024/1024 mem_total_gb, mem_assigned/1024/1024/1024 mem_assig
n_gb, (mem_total-mem_assigned)/1024/1024/1024 mem_free_gb
from __all_virtual_server_stat a join __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_
port)
order by a.zone, a.svr_ip
;
```

> **注意**
>
> 以上 SQL 的查询结果是四舍五入的。如果所有资源的分配采用整数，则查询结果为准确结果。

建表提示机器资源不足

- 现象

  使用普通的建表语句，遇到报错，提示机器资源不足。

```
MySQL [test]> create table t01(id bigint not null primary key auto_increment, c1 varchar(50), c
2 datetime not null default current_timestamp);
ERROR 4624 (HY000): machine resource is not enough to hold a new unit
```

- 原因

  在三副本集群里，默认建表时会创建三个副本。如果有节点掉线，并且该租户在掉线的节点上还有资源单元（`resource unit`）存在，则这个建表语句就无法创建三个副本。默认情况下，OceanBase 数据库为了保证表元数据强一致，就会报错。

  这个报错信息跟机器资源有关，但又不是那么直接。可以通过修改租户参数（`ob_create_table_strict_mode`）关闭这个强约束。

- 解决方法

  方法一：找出租户的资源单元所在节点，查看该节点掉线的原因，解决它，然后建表。这个时间可能有点长。

  方法二：会话级别或者全局级别关闭参数 `ob_create_table_strict_mode`，允许节点掉线情况下建表能成功。

```
MySQL [test]> show global variables like '%strict%';
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| ob_create_table_strict_mode | ON    |
+-----------------------------+-------+
1 row in set (0.009 sec)

MySQL [test]> set session ob_create_table_strict_mode=off;
Query OK, 0 rows affected (0.001 sec)

MySQL [test]> create table t01(id bigint not null primary key auto_increment, c1 varchar(50), c
2 datetime not null default current_timestamp);
Query OK, 0 rows affected (0.071 sec)

MySQL [test]> desc t01;
+-------+-------------+------+-----+-------------------+----------------+
| Field | Type        | Null | Key | Default           | Extra          |
+-------+-------------+------+-----+-------------------+----------------+
| id    | bigint(20)  | NO   | PRI | NULL              | auto_increment |
| c1    | varchar(50) | YES  |     | NULL              |                |
| c2    | datetime    | NO   |     | CURRENT_TIMESTAMP |                |
+-------+-------------+------+-----+-------------------+----------------+
3 rows in set (0.011 sec)
```

注意

节点异常要尽快修复。方法二理论上存在风险。

第 4 章：向 OceanBase 数据库迁移数据

OceanBase 社区版兼容 MySQL 数据库常用语法，主要兼容 5.5/5.6/5.7 版本的常用语法。从 MySQL 迁移数据到 OceanBase 时，需要判断 MySQL 数据库表的类型在 OceanBase 中是否兼容。然后，再根据实际情况使用不同的数据同步技术。

本章目录

4.1 OceanBase 的 MySQL 兼容性简介

MySQL 兼容性主要和表的数据类型、业务 SQL 的兼容性有关。 MySQL 的函数、触发器、存储过程在 OceanBase 的 MySQL 模式中不能良好的支持，因此不推荐使用。所以本章数据迁移只包含数据库表对象及其数据的迁移。

支持的数据类型

OceanBase 数据库支持的数据类型有:

- 数值类型
- 日期时间类型
- 字符类型
- 大对象类型

与 MySQL 数据库相比，OceanBase 数据库暂不支持空间数据类型和 JSON 数据类型，其他类别的数据类型中除大对象类型外，支持情况等于或大于 MySQL 数据库。

支持的 SQL 语法

此处仅列举常用的查询和修改 SQL。

- `SELECT`
  支持通过如下方式查看执行计划:

  ```
  EXPLAIN <SQL Statement> \G

  EXPLAIN extended_noaddr <SQL Statement> \G

  EXPLAIN extended <SQL Statement> \G
  ```

- `INSERT`
  - 支持单行、多行插入和指定分区插入
  - 支持 `INSERT INTO ... SELECT ...` 语句
- `UPDATE`
  - 支持单列和多列更新
  - 支持使用子查询
  - 支持集合更新
- `DELETE`

支持单表和多表删除

- `TRUNCATE`
  支持完全清空指定表

支持的变量

MySQL 实例中的参数叫 `variable`。可以通过以下方式设置:

- 启动时通过命令行设置
- 在启动的配置文件中设置
- SQL 命令行下设置
  - 全局级别
  - 会话级别

MySQL 的部分变量在 OceanBase 的 MySQL 租户中同样适用。但需注意源端 MySQL 和目标端 OceanBase 变量值的差异。

变量名	变量默认值 (OceanBase)	变量含义
autocommit	ON	是否自动提交。 ON：表示每个 DML 之后会自动添加 commit 语句。 OFF：表示不会自动 commit，需要用户主动 commit 或 rollback 事务。
auto_increment_cache_size	1000000	自增列的缓存值。自增列在节点重启发生切换后可能会跳号，该值越大，跳号间隔越大。
auto_increment_increment	1	自增列类型值的增长步长。
auto_increment_offset	1	自增列类型值的起始值。
character_set_client	utf8mb4	连接的客户端环境的字符集。保持默认即可。
character_set_connection	utf8mb4	连接自身的字符集。保持默认即可。
character_set_database	utf8mb4	连接的数据库的字符集。保持默认即可。

collation_connection	utf8mb4_general_ci	用于设置连接使用的字符集和字符序。
collation_database	utf8mb4_general_ci	设置创建数据库默认字符集和字符序。
collation_server	utf8mb4_general_ci	用于设置服务器默认字符集和字符序。
max_allowed_packet	4194304	需要调大，否则 SQL 文本或数据量很大时可能会报错。
sql_mode	STRICT_ALL_TABLES	用于设置 SQL 模式，不同的 SQL 模式对于插入等行为有很大影响。
transaction_isolation	READ-COMMITTED	用于设置事务的隔离级别。
tx_isolation	READ-COMMITTED	用于设置事务隔离级别。
lower_case_table_names	1	`1`：表示自动将表名转小写，对于表名的大小写不敏感。 `0`：表示不对表名大小写转换，对于表名的大小写敏感。
foreign_key_checks	ON	`ON`：检查外键约束。 `OFF`：不检查外键约束。

这些变量中要留意字符集相关设置、`sql_mode` 设置以及 `lower_case_table_names` 设置。这些设置建议在数据迁移之前完成，后期修改会带来风险。

支持的字符集

OceanBase MySQL 租户支持字符集 `binary` 和 `utf8mb4`，默认字符集是 `utf8mb4`。字符集 `utf8mb4` 是 `utf8` 的超集，比 `utf8` 多一些表情字符的编码。租户的字符集在租户创建时指定，是租户全局级别的字符集。

迁移数据到 OceanBase 时建议使用 `utf8mb4` 字符集。 在 `utf8mb4` 字符集下，每个英文字符长度为 1 字节，每个汉字长度为 3 字节，每个表情字符长度为 4 字节。如果迁移数据时更换了字符集，字符串列的长度需要适当增加。

```
MySQL [test]> create table t1(c1 varchar(50));
Query OK, 0 rows affected (0.04 sec)

MySQL [test]> insert into t1 values('a'),('中');
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

MySQL [test]> select length(c1) from t1;
+------------+
| length(c1) |
```

```
+------------+
|          1 |
|          3 |
+------------+
2 rows in set (0.01 sec)
```

从 MySQL 中导出数据时，需确保数据库中的字符都能正确输出到文件中。通过 `vim` 命令下的 `:set fileencoding` 命令可以查看文件的编码，一般为 `utf-8`，这样通过文件迁移 MySQL 数据时不会出现乱码现象。

4.2 如何使用 mysqldump 迁移 MySQL 表到 OceanBase

mysqldump 是 MySQL 提供的用于导出 MySQL 数据库对象和数据的工具。您可通过参数 `--help` 查看使用帮助，此处主要列举 mysqldump 常用的场景命令和参数搭配。

导出指定数据库的表结构（不包括数据）

您可运行下述命令导出指定数据库的表结构（不包括数据）。

```
mysqldump -h 127.1 -uroot -P3306 -p1******6  -d TPCH --compact > tpch_ddl.sql
```

导出来的脚本有如下特征:

- 脚本包含视图定义，但是会以 `/ *!* /` 注释。我们不关注视图，可以删除这部分内容。
- 脚本包含 OceanBase MySQL 不支持的语法，但是不影响，替换了即可。比如变量 `SQL_NOTES` 和 `DEFINER` 语句等。

  例如：导出的脚本里包含 `MAX_ROWS=` 的设置，这个是 MySQL 特有的语法，OceanBase MySQL 不支持该语法，执行会报错。

  ```
  /*!40101 SET character_set_client = @saved_cs_client */;
  /*!40101 SET @saved_cs_client     = @@character_set_client */;
  /*!40101 SET character_set_client = utf8 */;
  CREATE TABLE `NATION` (
   `N_NATIONKEY` int(11) NOT NULL,
    `N_NAME` char(25) COLLATE utf8_unicode_ci NOT NULL,
   `N_REGIONKEY` int(11) NOT NULL,
    `N_COMMENT` varchar(152) COLLATE utf8_unicode_ci DEFAULT NULL,
    PRIMARY KEY (`N_NATIONKEY`)
  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci MAX_ROWS=4294967295;
  ```

  您可把所有 `MAX_ROWS=` 以及后面部分批量注释掉。如在 `vim` 中使用 `:%s/MAX_ROWS=/; -- MAX_ROWS=/g`。

  注意

  上面导出的 SQL 中表名是大写，说明源端 MySQL 里设置表名默认很可能是大小写敏感。因此目标端 OceanBase MySQL 租户也需如此设置。

- 在导出的表结构语句里，可能包含外键。在导入 OceanBase MySQL 时，如果外键依赖的表没有创建，导入脚本会报错。因此导入之前需禁用外键检查约束。

  ```
  MySQL [oceanbase]> set global foreign_key_checks=off;
  Query OK, 0 rows affected (0.01 sec)

  MySQL [oceanbase]> show global variables like '%foreign%';
  +--------------------+-------+
  ```

```
| Variable_name      | Value |
+--------------------+-------+
| foreign_key_checks | OFF   |
+--------------------+-------+
1 row in set (0.00 sec)
```

修改后，退出会话并重新登录。在 obclient 客户端通过 `source` 命令执行外部 SQL 脚本文件。

导出指定数据库的表数据（不包括结构）

您可运行下述命令导出指定数据库的表数据（不包含结构）。

```
mysqldump -h 127.1 -uroot -P3306 -p1******6 -t TPCH  > tpch_data.sql
```

mysqldump 导出的数据初始化 SQL 里会首先将表锁住，禁止其他会话写，之后使用 `insert` 写入数据。每个 `insert` 后面的 `value` 里会有很多值，即批量 `insert`。

```
LOCK TABLES `t1` WRITE;
/*!40000 ALTER TABLE `t1` DISABLE KEYS */;
INSERT INTO `t1` VALUES ('a'),('中');
/*!40000 ALTER TABLE `t1` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;
```

注意

mysqldump 也支持导出 csv 文件。

如果您遇到了其他此处尚未提及的报错，欢迎到 OceanBase 社区版官网问答区反馈。反馈地址：https://ask.oceanbase.com/

4.3 如何使用 DBCAT 迁移 MySQL 表结构到 OceanBase

DBCAT 是一款轻量级的命令行工具，可用于提供源数据库到 OceanBase 数据库的 DDL 转换和 `Schema` 比对等功能。工具文件名为 `dbcat-[版本号]-SNAPSHOT.tar.gz`，下载后解压缩即可使用，可执行文件名为 `dbcat`。
OceanBase 社区版只兼容 MySQL，所以这里只演示 MySQL 表结构转换。

> **注意**
>
> DBCAT 是 OMS 的一个组件，OceanBase 社区版目前没有独立的包。

环境准备

DBCAT 能运行在 CentOS、MacXOS 和 Windows 下。需要安装 `JDK 1.8` 以上（含）版本。可以使用 `OpenJDK`，安装好后配置环境变量 `JAVA_HOME`。

CentOS 安装 `OpenJDK` 示例:

```
$sudo yum -y install java-1.8.0-openjdk.x86_64

$which java
/usr/local/java/jdk1.8.0_261/bin/java

echo 'export JAVA_HOME=/usr/local/java/jdk1.8.0_261/' >> ~/.bash_profile
. ~/.bash_profile
```

解压安装文件:

```
tar zxvf dbcat-1.3.0-SNAPSHOT.tar.gz
cd dbcat-1.3.0-SNAPSHOT/
chmod +x bin/dbcat

$tree -L 3 --filelimit 30
.
├── bin
│   ├── dbcat
│   ├── dbcat.bat
│   └── dbcat-debug
├── conf
│   ├── dbcat.properties
│   └── logback.xml
├── docs
│   ├── README.docx
│   ├── README.md
│   └── README.txt
├── LEGAL.md
├── lib [45 entries exceeds filelimit, not opening dir]
├── LICENSE
├── meta
│   └── README
```

```
└── NOTICE

5 directories, 12 files
```

安装文件中有以下几个目录需要了解:

目录名	说明
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。
meta	离线转换场景下，导出字典表数据。
~/output	SQL 文件与报告文件，运行时生成。

在线转换

在线转换是指 DBCAT 能直连源端数据库，将数据库中的对象导出。当对象非常多时（如超过 1 万），导出过程可能会有点慢。

您可运行命令 `bin/dbcat help convert` 查看转换命令帮助，具体参数使用请查看使用文档。

```
bin/dbcat convert -H<host> -P<port> -u<user> -p<password>  -D <database> --from <from> --to <to> --all

bin/dbcat convert -H 127.1 -P 3306 -uroot -p1****6 -D tpccdb --from mysql56 --to obmysql2230 --all
```

特别说明:

- 目前源端 MySQL 版本只支持 MySQL 5.5/5.6/5.7。`--from` 只支持 `mysql56` 和 `mysql57`。
- 目标端 OceanBase 版本的参数 `--to` 只支持 `obmysql2230` 和 `obmysql2250`。即使是 OceanBase 2.2.7 和 OceanBase 3.1 版本，也可以写成 `obmysql2250`。因为在 MySQL 兼容性方面，这些版本的 MySQL 语法是一样的。

运行后的输出文件在用户 `home` 目录的 `output` 下。

```
$tree ~/output/dbcat-2021-09-19-164533/
/home/qing.meiq/output/dbcat-2021-09-19-164533/
├── tpccdb
│   └── TABLE-schema.sql
└── tpccdb-conversion.html

1 directory, 2 files
```

示例：查看一个表结构在原生 MySQL 里的书写方式 和 OceanBase 数据库里的书写方式。

```
MariaDB [tpccdb]> show create table bmsql_customer \G
```

```
*************************** 1. row ***************************
       Table: bmsql_customer
Create Table: CREATE TABLE `bmsql_customer` (
  `c_w_id` bigint(20) NOT NULL,
  `c_d_id` bigint(20) NOT NULL,
  `c_id` bigint(20) NOT NULL,
  `c_discount` decimal(4,4) DEFAULT NULL,
  `c_credit` char(2) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_last` varchar(16) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_first` varchar(16) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_credit_lim` decimal(12,2) DEFAULT NULL,
  `c_balance` decimal(12,2) DEFAULT NULL,
  `c_ytd_payment` decimal(12,2) DEFAULT NULL,
  `c_payment_cnt` bigint(20) DEFAULT NULL,
  `c_delivery_cnt` bigint(20) DEFAULT NULL,
  `c_street_1` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_street_2` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_city` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_state` char(2) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_zip` char(9) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_phone` char(16) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_since` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `c_middle` char(2) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_data` varchar(500) COLLATE utf8_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`c_w_id`,`c_d_id`,`c_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
1 row in set (0.01 sec)

vim ~/output/dbcat-2021-09-19-164533/tpccdb/TABLE-schema.sql

create table if not exists tpccdb.bmsql_customer (
        c_w_id bigint(20) not null,
        c_d_id bigint(20) not null,
        c_id bigint(20) not null,
        c_discount decimal(4,4),
        c_credit char(2),
        c_last varchar(16),
        c_first varchar(16),
        c_credit_lim decimal(12,2),
        c_balance decimal(12,2),
        c_ytd_payment decimal(12,2),
        c_payment_cnt bigint(20),
        c_delivery_cnt bigint(20),
        c_street_1 varchar(20),
        c_street_2 varchar(20),
        c_city varchar(20),
        c_state char(2),
        c_zip char(9),
        c_phone char(16),
        c_since timestamp not null default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
        c_middle char(2),
        c_data varchar(500),
        primary key (c_w_id, c_d_id, c_id)
)
default charset=utf8mb4
default collate=utf8mb4_unicode_ci;
```

4.4 如何把 MySQL 表数据导出到 CSV 文件

CSV 文件简介

CSV 全称为 `Comma Separate Values`，这种文件格式经常被用来在不同程序之间做数据交互。

CSV 文件需满足以下要求：

- 有固定行分隔符，以区分不同的行。通常行分隔符是换行符，但并不完全是这样。
- 有固定列分隔符，以区分不同的列。默认列分隔符前后的空格会忽略。
- 如果列的内容里出现行分隔符和列分隔符，则会做转义处理。否则不能正确识别列或者行。

下面用一个典型的例子来加深对 CSV 文件的印象。

```
create table t1(id bigint not null  auto_increment, c1 char(10), c2 varchar(10), primary key(id)
insert into t1 (c1, c2) values(' 中 国 ',' 中 国 ');
insert into t1 (c1, c2) values(' 中,国 ',' "中 国" ');
insert into t1 (c1, c2) values(' 中
国 ','中
国 ');
insert into t1 (c1, c2) values(' 中\\国 ',' "中\\国" ');

MariaDB [test]> select * from t1;
+----+----------+-------------+
| id | c1       | c2          |
+----+----------+-------------+
|  1 |  中 国   |  中 国      |
|  2 |  中,国   |  "中 国"    |
|  3 |  中
国   | 中
国       |
|  4 |  中\国   |  "中\国"    |
+----+----------+-------------+
4 rows in set (0.00 sec)

MariaDB [test]> show global variables like '%secure_file_priv%';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| secure_file_priv |       |
+------------------+-------+
1 row in set (0.01 sec)
```

`secure_file_priv` 为空表示不限制导出导入，可以修改 MySQL 的启动参数文件（默认是 `/etc/my.cnf`），添加 `secure_file_priv=/tmp`。然后重启 MySQL。

```
MariaDB [test]> show global variables like '%secure_file_priv%';
+------------------+-------+
| Variable_name    | Value |
```

```
+------------------+-------+
| secure_file_priv | /tmp/ |
+------------------+-------+
1 row in set (0.00 sec)

MariaDB [test]> select * from t1 into outfile '/tmp/t1.csv' fields terminated by ',' enclosed
Query OK, 3 rows affected (0.00 sec)
```

如果不指定 FIELDS 或 LINES，缺省值为:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
LINES TERMINATED BY '\n' STARTING BY ''
```

查看导出文件 `/tmp/t1.csv`。

说明

通常在目录 `/tmp` 下能找到文件，但是个别比较老的系统，如在使用 `systemd` 的 CentOS 里的 Mariadb 5.5 环境下，根据该目录找不到文件。文件实际在目录 `/tmp/systemd-private-55b199ca1f6f42dfad8bfa065113e790-mariadb.service-EFJMNX/tmp/` 下。

```
#cat -n tmp/t1.csv
     1  "1"," 中 国"," 中 国 "
     2  "2"," 中,国"," \"中 国\" "
     3  "3"," 中\
4  国","中 \
5  国 "
     6  "4"," 中\\国"," \"中\\国\" "
```

从导出文件可以看出，字段的值由双引号封装，因此字段内的分隔符无影响。但是字段内如果包含双引号，将被默认转义，转义引导符是默认的 `\`。字段内的换行符亦无影响。

将此文件导入 MySQL。

```
MariaDB [test]> create table t2 like t1;
Query OK, 0 rows affected (0.01 sec)

MariaDB [test]> load data infile '/tmp/t1.csv' into table t2 fields terminated by ',' enclose
Query OK, 3 rows affected (0.00 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0

MariaDB [test]> select * from t2;
+----+----------+-------------+
| id | c1       | c2          |
+----+----------+-------------+
|  1 |  中 国   |  中 国      |
|  2 |  中,国   |  "中 国"    |
|  3 |  中
国   | 中
国      |
|  4 |  中\国   |  "中\国"    |
```

```
+----+----------+-------------+
4 rows in set (0.00 sec)
```

MySQL 的 `SELECT INTO OUTFILE` 和 `LOAD DATA` 还有很多高级用法，此处不详细列出。 建议导出和导入使用选项 `fields terminated by ',' enclosed by '"' lines terminated by '\n'`，可满足日常需求。

使用此方式导出的 CSV 文件，能导入 MySQL，同时也能导入 OceanBase MySQL。

4.5 如何使用 OceanBase 的 LOAD 命令加载 CSV 数据文件到 OceanBase

OceanBase MySQL 的 `load data` 命令和 MySQL 的 `load data` 命令相同。本节讲解导入操作所使用的文件为上节导出的 `t1.csv` 文件，不过需要将 `csv` 文件放到 observer 节点机器上。 OceanBase 数据库暂时不支持加载本地文件，该功能尚在研发中。

LOAD 语法

```
LOAD DATA
  [/*+ parallel(N) load_batch_size(M)*/]
    INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number {LINES | ROWS}]
    [(col_name_var
        [, col_name_var] ...)]
    [SET col_name={expr | DEFAULT},
        [, col_name={expr | DEFAULT}] ...]
```

LOAD 原理

Load Data 目前可以对 CSV 格式的文件进行导入，导入的流程如下:

1. 解析文件：OceanBase 数据库根据用户输入的文件名，读取文件中的数据，并且根据用户输入的并行度来决定使用并行或者串行解析输入文件中的数据。
2. 分发数据：由于 OceanBase 数据库是分布式数据库系统，各个分区的数据可能分布在各个不同的 observer，Load Data 会对解析出来的数据进行计算，决定数据需要被发送到哪个 observer。
3. 插入数据：当目标 observer 收到发送过来的数据之后，在本地执行 insert 操作把数据插入到对应的分区中。

为提高 Load Data 语句的性能，您可以执行加载数据的并行度。Load Data 在解析文件、计算分区和数据分发阶段均可多个线程并行工作。

为了避免分布式事务对性能的影响，Load Data 会按照分区将数据分组，并分发到 observer 上进行多次写入，每次写入开启一个独立的事务，写入的数据来源于一个分组。如果在 Load Data 语句执行过程中出现了错误，您需手工删除已经被加载的数据。如果导入数据文件很大，每个节点插入数据的时间可能会很长，请根据需要调整 `ob_query_timeout` 参数。

Load Data 提供了不同选项支持用户不同的需求，目前支持的选项有：

- 并行度

  `/ *+ parallel(N)* /` 选项指定加载数据的并行度，默认值是 N=4，建议使用的值范围是 [0 ~ 租户的最大 CPU 数]。

  ```
  load data /*+ parallel(4) */infile '/home/admin/a.csv' into table t
  ```

- 批量

  `/ *+ load_batch_size(M)* /` 选项指定每次插入的批量大小，默认是 M=1000。根据导入数据行的总长度调整 M 的大小，建议使用的范围是 [100-1000]。

- 输入文件

  `INFILE 'file_name'` 关键字指定输入文件的路径和文件名，目前 Load Data 只支持加载 observer 本地的输入文件。所以，您需在导入之前把文件拷贝到某一个 observer 所在机器上，并在文件所在的 observer 运行 Load Data 语句。

- 目标表索引

  为了提高导入效率，建议先建基础表，待导入完成后，再创建表的索引。对全局索引，务必要导入之后再创建，否则可能会报 `not support` 错误。

- 执行权限

  您需要授予权限才能访问机器上文件。授予权限有两步：

  1. 修改安全文件所在路径您可将安全文件路径设置为空，即无需检查。设置后需退出当前会话并重新登录方可生效。

     ```
     set global secure_file_priv = "";
     ```

  2. 对用户授予权限您可执行下述命令在 MySQL 模式下为用户授予 file 权限。

     ```
     `grant file on *.* to USER_NAME;`
     ```

- 重复数据处理

  `Replace` 表示将表中原有的数据替换为输入文件中的数据。`Ignore` 表示忽略重复的数据。

  Load Data 通过表的主键来判断数据是否重复，如果表不存在主键，那么 Load Data 语句就无法判断数据是否重复，replace 和 ignore 选项没有区别。

如果您不指定该选项，在发现重复数据时，Load Data 语句将把错误的数据记录到日志文件中。

- 目标表选项

  `NTO TABLE tbl_name` 关键字用于指定目标表名称。Load Data 支持分区表和非分区表。

- 字段格式

  字段格式用来指定输入文件的各个字段的分隔符选项，通过 `Fields\|Columns` 子句来指定。其中：

  - `Terminated By` 关键字用来指定字段的分隔符。
  - `Enclosed By` 关键字指定每个字段的开始和结束是否包含特定的字符。
  - `Escaped By` 关键字用来指定字段中的通配符。

- 行格式

  行格式指定输入文件中每一行的开始和结束字符，通过 `Lines` 子句设置。其中：

  - `Starting By` 用于指定每一行开始的字符。
  - `Terminated By` 用于指定每一行的结束字符。
  - `IGNORE number{LINES \| ROWS}` 子句指定忽略掉输入文件的前 `number` 行数据。

```
load data /*+ parallel(4) */infile '/home/admin/a.csv' into table t fields terminated by ',' lin
es terminated by '\n';
```

日志文件

如果导入的过程中出现了错误，基于 Load Data 的设计，出现错误的 insert 语句会被回滚，并且 Load Data 语句会在 observer 安装路径的 log 子目录下产生名称为 `obloaddata.log` 的日志文件，以下是一个日志文件的示例：

```
Tenant name: mysql
File name: /home/admin/a.csv
Into table: `test`.`t`
Parallel: 1
Batch size: 1000
SQL trace: YD7A20BA65670-0005AADAAA3CAB52
Start time: 2020-07-29 21:08:13.073741
Load query:
load data infile '/home/admin/test.csv' into table t fields terminated by ',' lines terminated by '\n'

Row ErrCode ErrMsg
1 1062 Duplicated primary key
2 1062 Duplicated primary key
```

日志中包含 Load Data 产生的任务的基本信息，例如：租户名、输入文件名、目标表名、并行度和使用的 Load Data 命令。并且以行为单位给出具体错误的信息。

示例

以上一节 MySQL 里导出的 `t1.csv` 为例，导入到 OceanBase 数据库。

```
$ mysql -h127.1 -uroot@test#obdemo -P2881 -p***** -c -A test -Ns
MySQL [test]> select * from t1;
MySQL [test]> load data infile '/tmp/t1.csv' into table t1 fields terminated by ',' enclosed by '"' li
nes terminated by '\n' ;
MySQL [test]> select * from t1;
1        中 国   中 国
2        中,国   "中 国"
3        中\n国 中 \n国
4        中\\国  "中\\国"
MySQL [test]>
```

如果有使用问题，欢迎到 OceanBase 社区版官网问答区反馈。反馈地址：https://open.oceanbase.com/answer。

4.6 如何使用 DataX 加载 CSV 数据文件到 OceanBase

DataX 简介

DataX 是阿里云 DataWorks 数据集成的开源版本，是阿里巴巴集团内被广泛使用的离线数据同步工具/平台。DataX 实现了包括 MySQL、Oracle、SQLserver、Postgre、HDFS、Hive、ADS、HBase、TableStore(OTS)、MaxCompute(ODPS)、Hologres、DRDS 、OceanBase 等各种异构数据源之间高效的数据同步功能。

OceanBase 企业版客户可以向 OceanBase 的技术人员索取 DataX 内部版本（RPM 包）。OceanBase 社区版客户，可以在 DataX 开源网站 内下载源码，自行编译。编译时，注意在 `pom.xml` 中剔除掉不用的数据库插件，否则编译出来的包会非常大。

DataX 配置文件

DataX 以任务的形式迁移数据。每个任务只处理一个表，每个任务有一个 `json` 格式的配置文件，配置文件里会包含 `reader` 和 `writer` 两节。具体的 `reader` 和 `writer` 都是 DataX 支持的数据库插件，可以随意搭配使用。

最新版本的 DataX 还提供了一个 WEB 管理界面。

配置文件示例:

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "sliceRecordCount": 10,
            "column": [
              {
                "type": "long",
                "value": "10"
              },
              {
                "type": "string",
                "value": "hello，你好，世界-DataX"
              }
            ]
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "encoding": "UTF-8",
            "print": true
          }
```

```
        }
      }
    ],
    "setting": {
      "speed": {
        "channel": 2
      }
    }
  }
}
```

将 `json` 配置文件放到 DataX 的目录 `job` 下，或者自定义路径。执行方法如下:

```
$bin/datax.py job/stream2stream.json
```

输出信息:

```
<.....>

2021-08-26 11:06:09.217 [job-0] INFO  JobContainer - PerfTrace not enable!
2021-08-26 11:06:09.218 [job-0] INFO  StandAloneJobContainerCommunicator - Total 20 records, 380 byte
s | Speed 38B/s, 2 records/s | Error 0 records, 0 bytes |  All Task WaitWriterTime 0.000s |  All Task
WaitReaderTime 0.000s | Percentage 100.00%
2021-08-26 11:06:09.223 [job-0] INFO  JobContainer -
任务启动时刻                    : 2021-08-26 11:05:59
任务结束时刻                    : 2021-08-26 11:06:09
任务总计耗时                    :                 10s
任务平均流量                    :               38B/s
记录写入速度                    :              2rec/s
读出记录总数                    :                  20
读写失败总数                    :                   0
```

DataX 任务执行结束会有个简单的任务报告，包含平均流量、写入速度和读写失败总数等。

DataX 的 `job` 的参数 `settings` 可以指定速度参数和错误记录容忍度等。

```
"setting": {
            "speed": {
                "channel": 10
            },
            "errorLimit": {
                "record": 10,
                "percentage": 0.1
            }
        }
```

参数说明:

- `speed` 限速 `bytes` 有 bug，不建议使用。`errorLimit` 表示报错记录数的容忍度，超出这个限制后任务就中断退出。
- `channel` 是并发数，理论上并发越大，迁移性能越好。但实际操作中也要考虑源端的读压力、网络传输性能以

及目标端写入性能。

下面介绍常用数据源（MySQL、Oracle 、CSV 和 OceanBase）的读写插件。

txtfilereader 插件

txtfilereader 插件提供了读取本地文件系统数据存储的能力。在底层实现上，txtfilereader 获取本地文件数据，并转换为 DataX 传输协议传递给 Writer。

本地文件内容存放的是一张逻辑意义上的二维表，例如 CSV 格式的文本信息。

txtfilereader 有一些功能限制和参数，请先参考官方说明：https://github.com/alibaba/DataX/blob/master/txtfilereader/doc/txtfilereader.md。

下面是 txtfilereader 的配置示例。

```
"reader": {
                    "name": "txtfilereader",
                    "parameter": {
                        "path": ["/tmp/tpcc/bmsql_oorder"],
                        "fileName": "bmsql_oorder",
                        "encoding": "UTF-8",
                        "column": ["*"],
                        "dateFormat": "yyyy-MM-dd hh:mm:ss" ,
                        "nullFormat": "\\N" ,
                        "fieldDelimiter": ","
                    }
                }
```

参数说明

- `path` 指定到路径即可。
- `fileName` 是生成的文件前缀，完整的文件名很长，有随机字符串（避免重复）。
- `column` 可以指定为 `*`，此时所有字段值均作为字符串。该操作虽然方便，但不能保证完全没有问题，目前测试常用数据类型是可以的。
- `nullFormat` 指定空值的记录，默认是 `null`，这个读入 Oracle 时会出现问题。建议导出文件时指定为 `"\\N"`，即空值和。
- `fieldDelimiter` 指定 CSV 文件的列分隔符，这个和导出的时候指定的列分隔符保持一致。通常导出的列内容中如果含有列分隔符，会用双引号（`,`）进行封装。用逗号（`,`）也可以，只是太过常见，建议用生僻一点的单字符，如 `|` 或 `^` 等。

mysqlreader 插件

`mysqlreader` 插件实现了从 MySQL 读取数据。在底层实现上，MysqlReader 通过 JDBC 连接远程 MySQL 数据库，并执行相应的 SQL 语句将数据从 MySQL 库中 SELECT 出来。

不同于其他关系型数据库，MysqlReader 不支持 FetchSize。

实现原理方面，简而言之，MysqlReader 通过 JDBC 连接器连接到远程的 MySQL 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 MySQL 数据库，并将该 SQL 执行返回结果使用 DataX 自定义的数据类型拼装为抽象的数据集，并传递给下游 Writer 处理。

详细功能和参数说明请参考官方说明：https://github.com/alibaba/DataX/blob/master/mysqlreader/doc/mysqlreader.md。

下面是 `mysqlreader` 的配置示例。

```
"reader": {
                    "name": "mysqlreader",
                    "parameter": {
                        "username": "tpcc",
                        "password": "********",
                        "column": [
                            "*"
                        ],
                        "connection": [
                            {
                                "table": [
                                    "bmsql_oorder"
                                ],
                                "jdbcUrl": ["jdbc:mysql://127.0.0.1:3306/tpccdb?useUnicode=true&charact
erEncoding=utf8"]
                            }
                        ]
                    }
                }
```

参数说明

- 如果表的主键是单列主键（比如 `id`），那么可以在 `parameter` 下加一个配置： `"splitPk": "db_id",`。如果加在最后，则需去掉后面的逗号（`,`）。
- `column` 指定读取的列。通常建议写具体的列，可以在列上用函数做逻辑转换。使用 `*` 就是要时刻确保列跟目标端写入的列能对应上。

oceanbasev10writer 插件

`oceanbasev10writer` 插件实现了写入数据到 OceanBase 主库的目标表的功能。在底层实现上，OceanbaseV10Writer 通过 Java 客户端（底层 MySQL JDBC 或 OceanBase Client）连接以 obproxy 远程 OceanBase 数据库，并执行相应的 insert sql 语句将数据写入 OceanBase 数据库，内部会分批次提交入库。

实现原理

Oceanbasev10Writer 通过 DataX 框架获取 Reader 生成的协议数据，生成 insert 语句。对于 MySQL 租户，在主键或唯一键冲突时，可以选择 `replace` 模式更新表中的所有字段。对于 Oracle 租户，目前只有 insert 行为。

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。

下面是 `oceanbasev10writer` 的配置示例。

```
"writer": {
                    "name": "oceanbasev10writer",
                    "parameter": {
                        "obWriteMode": "insert",
                        "column": [
                            "*"
                        ],
                        "preSql": [
                            "truncate table bmsql_oorder"
                        ],
                        "connection": [
                            {
                                "jdbcUrl": "||_dsc_ob10_dsc_||obdemo:oboracle||_dsc_ob10_dsc_||jdbc:oc
eanbase://127.0.0.1:2883/tpcc?useLocalSessionState=true&allowBatch=true&allowMultiQueries=true&rewriteB
atchedStatements=true",
                                "table": [
                                    "bmsql_oorder"
                                ]
                            }
                        ],
                        "username": "tpcc",
                        "password":"********",
                        "batchSize": 512,
                        "writerThreadCount":10,
                        "memstoreThreshold": "0.9"
                    }
                }
```

参数说明

仅供参考，以后可能会发生改变，详细情况请关注 `DataX` 开源地址：https://github.com/alibaba/DataX/tree/master/oceanbasev10writer。

参数	是否必选	默认值	参数说明
`jdbcUrl`	是	无	目的数据库的连接信息，包含了 `OceanBase` 的集群、租户、`obproxy` 的地址和端口以及库名。 使用域名可能会报错，建议使用 `ip`。
`table`	是	无	目的表的表名称。支持写入一个或者多个表。当配置为多张表时，必须确保所有表结构保持一致，表名中一般不含库名。
`column`	是	否	目的表需要写入数据的字段，字段之间用英文逗号分隔。例如：`"column": ["id","name","age"]`。 `column` 配置项必须指定，不能留空。
`preSql`	否	无	写入数据到目的表之前，会先执行这里的标准语句。如果 SQL 中包含您需要

			操作的表名称，请使用
			`@table`
			表示，这样在实际执行 SQL 语句时，会对变量按照实际表名称进行替换。
			
			比如：您的任务是要写入到目的端 100 个同构分表（表名称为
			`datax_00,datax01, ... datax_98,datax_99`
			），并且你希望导入数据前，先对表中数据进行删除操作。那么您可以采用配置
			`"preSql":\["delete from @table"\]`
			。这样在执行到每个表写入数据前，会先执行对应的
			`delete from`
			对应表名称。
			
			说明
			该配置只支持 delete 语句。

`batchSize`	否	1000	一次性批量提交的记录数大小，该值可以极大减少 DataX 与 OceanBase 的网络交互次数，并提升整体吞吐量。但是该值设置过大可能会造成 DataX 运行进程 OOM。
`memstoreThreshold`	否	0	OceanBase 租户的 memstore 使用率，当达到这个阀值时暂停导入，等释放内存后继续导入，防止租户内存溢出。
`username`	是	无	访问 oceanbase1.0 的用户名，需要注意的是，此处不配置 OceanBase 的集群名和租户名。
`password`	是	无	访问 oceanbase1.0 的密码。
`writerThreadCount`	否	1	每个通道（channel）中写入使用的线程数。

MySQL 数据导出为 CSV 文件

将 MySQL 数据导出为 CSV 文件。

配置文件如下:

```
$cat job/bmsql_oorder_mysql2csv.json
{
    "job": {
        "setting": {
            "speed": {
                "channel": 4
            },
            "errorLimit": {
                "record": 0,
                "percentage": 0.1
            }
```

```
        },
        "content": [
            {
                "reader": {
                    "name": "mysqlreader",
                    "parameter": {
                        "username": "tpcc",
                        "password": "********",
                        "column": [
                            "*"
                        ],
                        "connection": [
                            {
                                "table": [
                                    "bmsql_oorder"
                                ],
                                "jdbcUrl": ["jdbc:mysql://127.0.0.1:3306/tpccdb?useUnicode=true&charact
erEncoding=utf8"]
                            }
                        ]
                    }
                },
                "writer": {
                    "name": "txtfilewriter",
                    "parameter": {
                        "path": "/tmp/tpcc/bmsql_oorder",
                        "fileName": "bmsql_oorder",
                        "encoding": "UTF-8",
                        "writeMode": "truncate",
                        "dateFormat": "yyyy-MM-dd hh:mm:ss" ,
                        "nullFormat": "\\N" ,
                        "fileFormat": "csv" ,
                        "fieldDelimiter": ","
                    }
                }
            }
        ]
    }
}
```

CSV 文件导入到 OceanBase

将源端导出的 CSV 文件复制到目标端的 DataX 服务器上，然后导入到目标端 OceanBase 数据库中。

配置文件如下:

```
$cat job/bmsql_oorder_csv2ob.json
{
    "job": {
        "setting": {
            "speed": {
                "channel": 4
            },
            "errorLimit": {
```

```
                "record": 0,
                "percentage": 0.1
            }
        },
        "content": [
            {
                "reader": {
                    "name": "txtfilereader",
                    "parameter": {
                        "path": ["/tmp/tpcc/bmsql_oorder"],
                        "fileName": "bmsql_oorder",
                        "encoding": "UTF-8",
                        "column": ["*"],
                        "dateFormat": "yyyy-MM-dd hh:mm:ss" ,
                        "nullFormat": "\\N" ,
                        "fieldDelimiter": ","
                    }
                },
                "writer": {
                    "name": "oceanbasev10writer",
                    "parameter": {
                        "obWriteMode": "insert",
                        "column": [
                            "*"
                        ],
                        "preSql": [
                            "truncate table bmsql_oorder"
                        ],
                        "connection": [
                            {
                                "jdbcUrl": "||_dsc_ob10_dsc_||obdemo:oboracle||_dsc_ob10_dsc_||jdbc:oc
eanbase://127.0.0.1:2883/tpcc?useLocalSessionState=true&allowBatch=true&allowMultiQueries=true&rewriteB
atchedStatements=true",
                                "table": [
                                    "bmsql_oorder"
                                ]
                            }
                        ],
                        "username": "tpcc",
                        "password":"********",
                        "writerThreadCount":10,
                        "batchSize": 1000,
                        "memstoreThreshold": "0.9"
                    }
                }
            }
        ]
    }
}
```

4.7 如何使用 DataX 迁移 MySQL 数据到 OceanBase

将 MySQL 数据迁移到 OceanBase 数据库，如果源端和目标端不能同时跟 DataX 服务器网络联通，则使用上节方法通过 CSV 文件中转。如果源端数据库和目标端数据库能同时跟 DataX 所在服务器联通，则可以使用 DataX 直接将数据从源端迁移到目标端。

MySQL 数据同步到 OceanBase

配置文件如下:

```
{
    "job": {
        "setting": {
            "speed": {
                "channel": 4
            },
            "errorLimit": {
                "record": 0,
                "percentage": 0.1
            }
        },
        "content": [
            {
                "reader": {
                    "name": "mysqlreader",
                    "parameter": {
                        "username": "tpcc",
                        "password": "********",
                        "column": [
                            "*"
                        ],
                        "connection": [
                            {
                                "table": [
                                    "bmsql_oorder"
                                ],
                                "jdbcUrl": ["jdbc:mysql://127.0.0.1:3306/tpccdb?useUnicode=true&charact
erEncoding=utf8"]
                            }
                        ]
                    }
                },

                "writer": {
                    "name": "oceanbasev10writer",
                    "parameter": {
                        "obWriteMode": "insert",
                        "column": [
                            "*"
                        ],
                        "preSql": [
                            "truncate table bmsql_oorder"
```

```
                        ],
                        "connection": [
                            {
                                "jdbcUrl": "||_dsc_ob10_dsc_||obdemo:oboracle||_dsc_ob10_dsc_||jdbc:oc
eanbase://127.0.0.1:2883/tpcc?useLocalSessionState=true&allowBatch=true&allowMultiQueries=true&rewriteB
atchedStatements=true",
                                "table": [
                                    "bmsql_oorder"
                                ]
                            }
                        ],
                        "username": "tpcc",
                        "password":"********",
                        "writerThreadCount":10,
                        "batchSize": 1000,
                        "memstoreThreshold": "0.9"
                    }
                }
            }
        ]
    }
}
```

常见报错问题解决

MySQL 端 ssl 相关的报错

- 现象

```
Mon Dec 13 15:44:13 CST 2021 WARN: Establishing SSL connection without server's identity verific
ation is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+ requirements SSL connec
tion must be established by default if explicit option isn't set. For compliance with existing a
pplications not using SSL the verifyServerCertificate property is set to 'false'. You need eithe
r to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore f
or server certificate verification
```

- 解决方案

  您可在 jdbcurl 里关闭 ssl，示例:

```
"jdbcUrl": ["jdbc:mysql://127.0.0.1:3306/tpccdb?useUnicode=true&characterEncoding=utf8"]
# 调整为:
"jdbcUrl": ["jdbc:mysql://127.0.0.1:3306/tpccdb?useUnicode=true&characterEncoding=utf8&useSSL=fa
lse"]
```

存在外键导入报错

当有外键的表进 truncate 时报错，可以使用 `delete from table` 代替，也可以在 prepare sql 语句里修改 `set foreign_key_checks='off'` 以避免报错。

导入数据时事物超时

出现该报错时您可在 prepare sql 中增加 timeout 的设置，命令为：`set ob_trx_timeout=1000000000`。

oceanbasev10reader 和 oceanbasev10writer 插件找不到

出现该报错说明 OceanBase 读写插件在默认的二进制 DataX 包里不存在，需要编译安装后创建对应的目录并复制相关的 json 文件，文件的路径和内容如下：

```
target/datax/datax/plugin/reader/oceanbasev10reader/plugin_job_template.json
target/datax/datax/plugin/writer/oceanbasev10writer/plugin_job_template.json
```

```
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "oceanbasev10reader",
                    "parameter": {
                        "column": [
                            "*"
                        ],
                        "connection": [
                            {
                                "jdbcUrl": [
                                    "||_dsc_ob10_dsc_||clusterName:tenantName||_dsc_ob10_dsc_||jdbc:oc
eanbase://obproxyIp:port/dbname"
                                ],
                                "table": [
                                    "tabName"
                                ]
                            }
                        ],
                        "password": "",
                        "readBatchSize": 100000,
                        "username": "",
                        "weakRead": false
                    }
                },
                "writer": {
                    "name": "oceanbasev10writer",
                    "parameter": {
                        "column": [
                            "*"
                        ],
                        "connection": [
                            {
                                "jdbcUrl": "||_dsc_ob10_dsc_||clusterName:tenantName||_dsc_ob10_dsc_||
jdbc:oceanbase://obproxyIp:port/dbname?yearIsDateType=false&ZeroDateTimeBehavior=convertToNull&tinyIntl
isBit=false&rewriteBatchedStatements=true",
                                "table": [
                                    "tabName"
                                ]
```

```
                            }
                        ],
                        "obWriteMode": "insert",
                        "password": "",
                        "preSql": [
                            ""
                        ],
                        "username": ""
                    }
                }
            }
        ],
        "setting": {
            "speed": {
                "channel": ""
            }
        }
    }
}
```

插件 mysqlreader,txtfilewriter 加载失败

解决方案:

删除 `plugin[reader,writer]` 目录下所有 `._` 开头的文件夹/文件。

```
rm -rf ._*
```

4.8 如何使用 OBDUMPER / OBLOADER 工具导出/导入 OceanBase 数据

OBDUMPER 导出 OceanBase 数据库

和导出 MySQL 数据库一样，导出 OceanBase 数据库建议结构和数据分开导出。

命令帮助如下，需要指定业务租户的用户名和密码。比较特别的地方是，如果要导出表结构，还需要指定 sys 租户的用户 `root` 和密码或者用户 `proxyro` 和密码。

为了安全性，OBProxy 默认禁止使用用户 `proxyro` 登录 OceanBase 集群，需要先测试其连通性。因为`root` 用户的权限过于大，所以这里推荐使用用户 `proxyro` 获取表的元数据信息。

```
bin/obdumper -h <主机IP> -P <端口> -u <用户> -p <密码> --sys-user <sys 租户下的root 用户或 proxyro 用户> --s
ys-password <sys 租户下的账户密码> -c <集群> -t <租户> -D <Schema 库名> [--ddl] [--csv|--sql] [--all|--tabl
e '表名'] -f<数据文件或者目录>
```

测试 proxyro 连通性

`proxyro` 的密码在 OBD 部署集群配置文件里指定。

```
[root@obce00 ~]# mysql -h 172.xx.xxx.52 -P 2883 -u proxyro@sys#obdemo -puY****zx -c -A
ERROR 2013 (HY000): Lost connection to MySQL server at 'reading authorization packet', system error: 1
1
```

上面密码并没有错，登录失败是因为 OBProxy 默认禁止用户 `proxyro` 登录。修改 OBProxy 参数跳过用户 `proxyro` 登录检查。

```
mysql -h 172.xx.xxx.52 -u root@proxysys -P 2883 -p0M****tm
alter proxyconfig set skip_proxyro_check=true;
MySQL [(none)]> show proxyconfig like '%skip_proxyro_check%';
+--------------------+-------+-----------------------------------------------------+-------------+-------
--------+
| name               | value | info                                                | need_reboot | visibl
e_level |
+--------------------+-------+-----------------------------------------------------+-------------+-------
--------+
| skip_proxyro_check | True  | used for proxro@sys, if set false, access denied | false       | SY
S           |
+--------------------+-------+-----------------------------------------------------+-------------+-------
--------+
1 row in set (0.010 sec)
```

再次测试 `proxyro` 连通性。

```
[root@obce00 ~]# mysql -h 172.xx.xxx.52 -P 2883 -u proxyro@sys#obdemo -puY****zx -c -A -Ns -e "show da
tabases;"
oceanbase
information_schema
[root@obce00 ~]#
```

只导出表结构

下面导出业务租户 `obmysql` 下的数据库 `test` 下的表。

```
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]# bin/obdumper -h 172.xx.xxx.52 -P 2883 -u dumper -p ***
** --sys-user=proxyro --sys-password=uY****zx -c obdemo -t obmysql -D test --ddl --all -f /tmp/obdumpe
r
2021-09-29 21:14:55 [INFO] Parsed args:
-h[--host] 172.xx.xxx.52
-P[--port] 2883
-u[--user] dumper
-p[--password] ******
[--sys-user] proxyro
[--sys-password] ******
-c[--cluster] obdemo
-t[--tenant] obmysql
-D[--database] test
[--ddl]
[--all]
-f[--file-path] /tmp/obdumper

obproxy Druid LogFactory, userDefinedLogType=null, logInfo=public com.alipay.oceanbase.obproxy.druid.s
upport.logging.Log4j2Impl(java.lang.String)
2021-09-29 21:14:55 [INFO] {dataSource-1} inited
2021-09-29 21:14:55 [INFO] {dataSource-2} inited
2021-09-29 21:14:55 [INFO] The manifest file: "/tmp/obdumper/data/MANIFEST.bin" has been saved
2021-09-29 21:14:57 [WARN] No views are exist in the schema: "test"
2021-09-29 21:14:57 [INFO] Generate 1 dump tasks finished
2021-09-29 21:14:57 [INFO] Start 1 schema dump threads finished
2021-09-29 21:14:57 [INFO] Build direct com.alibaba.druid.pool.DruidDataSource finished
2021-09-29 21:14:57 [INFO] Build proxyro com.alibaba.druid.pool.DruidDataSource finished
2021-09-29 21:14:57 [INFO] Return the latest compatible version: 3.1.0 -> 2.2.71
2021-09-29 21:14:57 [INFO] DbType: OBMYSQL Version: 3.1.0
2021-09-29 21:14:57 [INFO] ObMySql(3.1.0) is older than 2.2.71 ? false
2021-09-29 21:14:57 [INFO] Load meta/mysql/mysql56.xml, meta/ob/obmysql14x.xml, meta/ob/obmysql22x.xml
, meta/ob/obmysql2271.xml successed
2021-09-29 21:14:58 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquireDependencies()
2021-09-29 21:14:58 [INFO] Query 0 dependencies elapsed 846.9 ms
2021-09-29 21:15:01 [INFO]
Finished Tasks: 0       Running Tasks: 1        Progress: 0.00%
2021-09-29 21:15:04 [INFO] Query table: "t1" attr finished. Remain: 0
2021-09-29 21:15:04 [INFO] Query 1 tables elapsed 5.775 s
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquireTablespaceMapping()
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquireSequenceMapping()
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquireMaterializedViewMapp
ing()
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquireAliasMapping()
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquireSynonymMapping()
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquireTypeMapping()
```

```
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquireTypeBodyMapping()
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquirePackageMapping()
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquirePackageBodyMapping()
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquireTriggerMapping()
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquireProcedureMapping()
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquireFunctionMapping()
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquireDatabaseLinkMapping(
)
2021-09-29 21:15:04 [WARN] c.o.o.d.m.o.ObMySql14xDatabase does't implement acquireDependencies()
2021-09-29 21:15:04 [INFO] Dump [TABLE] t1 to "/tmp/obdumper/data/test/TABLE/t1-schema.sql" finished
2021-09-29 21:15:04 [INFO] No.1 It has dumped 1 tables finished. Remain: 0
2021-09-29 21:15:04 [INFO] Total dumped 1 tables finished
2021-09-29 21:15:04 [INFO] Dump the ddl of schema: "test" finished
2021-09-29 21:15:04 [INFO] {dataSource-1} closing ...
2021-09-29 21:15:04 [INFO] {dataSource-1} closed
2021-09-29 21:15:04 [INFO] Close count: 7
2021-09-29 21:15:04 [INFO] {dataSource-2} closing ...
2021-09-29 21:15:04 [INFO] {dataSource-2} closed
2021-09-29 21:15:04 [INFO] Close count: 6
2021-09-29 21:15:04 [INFO] Shutdown task context finished
2021-09-29 21:15:04 [INFO]
Finished Tasks: 1       Running Tasks: 0        Progress: 100.00%
2021-09-29 21:15:04 [INFO]

All Dump Tasks Finished:

----------------------------------------------------------------------------------------------------
----------------------
No.#        |        Type        |             Name             |            Count
|       Status
----------------------------------------------------------------------------------------------------
----------------------
1           |       TABLE        |              t1              |              1               |      S
UCCESS
----------------------------------------------------------------------------------------------------
----------------------

Total Count: 1          End Time: 2021-09-29 21:15:04


2021-09-29 21:15:04 [INFO] Dump schema finished. Total Elapsed: 7.051 s
2021-09-29 21:15:04 [INFO] System exit 0
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]#
```

OBDUMPER 导出的目录结构需要熟悉，方便快速找到导出脚本或者日志，具体如下:

```
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]# tree /tmp/obdumper/
/tmp/obdumper/
├── CHECKPOINT.bin
├── data
│   ├── MANIFEST.bin
│   └── test
│       └── TABLE
│           └── t1-schema.sql
└── logs
    ├── ob-loader-dumper.error
    ├── ob-loader-dumper.info
```

```
    └── ob-loader-dumper.warn

4 directories, 6 files
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]#
```

如果导出报错，可以在目录 `logs/` 下查看日志。

只导出数据

```
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]# bin/obdumper -h 172.xx.xxx.52 -P 2883 -u dumper -p ***
*** --sys-user=proxyro --sys-password=uY****zx -c obdemo -t obmysql -D test --csv --all -f /tmp/obdump
er
2021-09-29 21:16:54 [INFO] Parsed args:
-h[--host] 172.xx.xxx.52
-P[--port] 2883
-u[--user] dumper
-p[--password] ******
[--sys-user] proxyro
[--sys-password] ******
-c[--cluster] obdemo
-t[--tenant] obmysql
-D[--database] test
[--csv]
[--all]
-f[--file-path] /tmp/obdumper

obproxy Druid LogFactory, userDefinedLogType=null, logInfo=public com.alipay.oceanbase.obproxy.druid.s
upport.logging.Log4j2Impl(java.lang.String)
2021-09-29 21:16:54 [INFO] {dataSource-1} inited
2021-09-29 21:16:54 [INFO] {dataSource-2} inited
2021-09-29 21:16:54 [INFO] The manifest file: "/tmp/obdumper/data/MANIFEST.bin" has been saved
2021-09-29 21:16:55 [INFO] Generate 1 csv dump tasks for non-partitioned table(without macro): t1. Rem
ain: 0
2021-09-29 21:16:55 [INFO] Generate 1 dump tasks finished
2021-09-29 21:16:55 [INFO] Start 16 record dump threads finished
2021-09-29 21:16:55 [INFO] Dump 8 rows test.t1 to "/tmp/obdumper/data/test/TABLE/t1.csv" finished
2021-09-29 21:16:56 [INFO] {dataSource-1} closing ...
2021-09-29 21:16:56 [INFO] {dataSource-1} closed
2021-09-29 21:16:56 [INFO] Close count: 1
2021-09-29 21:16:56 [INFO] {dataSource-2} closing ...
2021-09-29 21:16:56 [INFO] {dataSource-2} closed
2021-09-29 21:16:56 [INFO] Close count: 1
2021-09-29 21:16:56 [INFO] Shutdown task context finished
2021-09-29 21:16:56 [INFO]
Finished Tasks: 1       Running Tasks: 0        Progress: 100.00%
2021-09-29 21:16:56 [INFO]

All Dump Tasks Finished:

-------------------------------------------------------------------------------------------------------
----------------------
        No.#        |        Type        |             Name             |            Coun
t             |       Status
-------------------------------------------------------------------------------------------------------
----------------------
```

```
         1          |       TABLE        |              t1              |
8               |      SUCCESS
-------------------------------------------------------------------------------------------------------
----------------------

Total Count: 8          End Time: 2021-09-29 21:16:56


2021-09-29 21:16:56 [INFO] You can't merge the data files for 1 tables. --file-name is missing
2021-09-29 21:16:56 [INFO] Dump record finished. Total Elapsed: 1.548 s
2021-09-29 21:16:56 [INFO] System exit 0
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]#
```

目录结构如下:

```
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]# tree /tmp/obdumper
/tmp/obdumper
├── CHECKPOINT.bin
├── data
│   ├── MANIFEST.bin
│   └── test
│       └── TABLE
│           ├── t1.0.csv
│           └── t1-schema.sql
└── logs
    ├── ob-loader-dumper.error
    ├── ob-loader-dumper.info
    └── ob-loader-dumper.warn

4 directories, 7 files
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]#
```

导出的数据文件（CSV 格式）跟表结构文件在同一个目录。

查看导出结果文件:

```
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]# cat /tmp/obdumper/data/test/TABLE/t1-schema.sql
create table if not exists `t1` (
        `id` bigint(20) not null auto_increment,
        `c1` timestamp not null default CURRENT_TIMESTAMP,
        primary key (`id`)
)
default charset=utf8mb4
default collate=utf8mb4_general_ci;
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]# cat /tmp/obdumper/data/test/TABLE/t1.0.csv
'id','c1'
1,'2021-09-29 21:16:03'
2,'2021-09-29 21:16:05'
3,'2021-09-29 21:16:05'
4,'2021-09-29 21:16:06'
5,'2021-09-29 21:16:06'
6,'2021-09-29 21:16:18'
7,'2021-09-29 21:16:18'
8,'2021-09-29 21:16:19'
```

OBLOADER 导入 OceanBase 数据库

同样的，也建议导入 OceanBase 数据库的时候，表结构和数据分开导入。

OBLOADER 命令帮助:

```
bin/obloader -h <主机IP> -P <端口> -u <用户> -p <密码>  --sys-user <sys 租户下的 root 用户或 proxyro 用户> --sys-password <sys 租户下的账户密码> -c <集群> -t <租户> -D <Schema 库名> [--ddl] [--csv|--sql] [--all|--table '表名'] -f<数据文件或者目录>
```

跟上面一样，导入的时候 OBLOADER 需要获取表结构元数据信息，也需要连接到 sys 租户下。推荐使用用户 `proxyro`。

导入表结构

下面将前面的数据导出文件导入到租户 `obmysql` 的数据库 `test2` 下。

```
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]# bin/obloader -h 172.xx.xxx.52 -P 2883 -u loader -p ****** --sys-user=proxyro --sys-password=uY****zx -c obdemo -t obmysql -D test2 --ddl --all -f /tmp/obdumper
2021-09-29 21:26:05 [INFO] Parsed args:
-h[--host] 172.xx.xxx.52
-P[--port] 2883
-u[--user] loader
-p[--password] ******
[--sys-user] proxyro
[--sys-password] ******
-c[--cluster] obdemo
-t[--tenant] obmysql
-D[--database] test2
[--ddl]
[--all]
-f[--file-path] /tmp/obdumper

2021-09-29 21:26:05 [INFO] No control files were defined in the path: "/tmp/obdumper"
2021-09-29 21:26:05 [INFO] No control files were defined in the path: "/tmp/obdumper"
2021-09-29 21:26:05 [INFO] No mapping files were defined in the path: "/tmp/obdumper"
obproxy Druid LogFactory, userDefinedLogType=null, logInfo=public com.alipay.oceanbase.obproxy.druid.support.logging.Log4j2Impl(java.lang.String)
2021-09-29 21:26:05 [INFO] {dataSource-1} inited
2021-09-29 21:26:05 [INFO] {dataSource-2} inited
2021-09-29 21:26:05 [INFO] The manifest file: "/tmp/obdumper/data/MANIFEST.bin" has been saved
2021-09-29 21:26:05 [INFO] Init writer thread pool finished
2021-09-29 21:26:05 [INFO] No.1 sql of the file: "t1-schema.sql" exec success. Elapsed: 114.4 ms
2021-09-29 21:26:05 [INFO] Load file: "t1-schema.sql" finished
2021-09-29 21:26:06 [INFO] {dataSource-2} closing ...
2021-09-29 21:26:06 [INFO] {dataSource-2} closed
2021-09-29 21:26:06 [INFO] Close count: 0
2021-09-29 21:26:06 [INFO] {dataSource-1} closing ...
2021-09-29 21:26:06 [INFO] {dataSource-1} closed
2021-09-29 21:26:06 [INFO] Close count: 1
2021-09-29 21:26:06 [INFO] Shutdown task context finished
2021-09-29 21:26:06 [INFO]
```

```
Finished Tasks: 1       Running Tasks: 0        Progress: 100.00%
2021-09-29 21:26:06 [INFO]

All Load Tasks Finished:

-------------------------------------------------------------------------------------------------------------
----------------------
        No.#        |        Type        |             Name             |            Coun
t             |       Status
-------------------------------------------------------------------------------------------------------------
----------------------
         1          |       TABLE        |              t1              |
1               |      SUCCESS
-------------------------------------------------------------------------------------------------------------
----------------------

Total Count: 1          End Time: 2021-09-29 21:26:06


2021-09-29 21:26:06 [INFO] Load schema finished. Total Elapsed: 1.070 s
2021-09-29 21:26:06 [INFO] System exit 0
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]#
```

导入数据

```
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]# bin/obloader -h 172.xx.xxx.52 -P 2883 -u loader -p ***
*** --sys-user=proxyro --sys-password=uY****zx -c obdemo -t obmysql -D test2 --csv --all -f /tmp/obdum
per
2021-09-29 21:27:53 [INFO] Parsed args:
-h[--host] 172.xx.xxx.52
-P[--port] 2883
-u[--user] loader
-p[--password] ******
[--sys-user] proxyro
[--sys-password] ******
-c[--cluster] obdemo
-t[--tenant] obmysql
-D[--database] test2
[--csv]
[--all]
-f[--file-path] /tmp/obdumper

2021-09-29 21:27:53 [INFO] No control files were defined in the path: "/tmp/obdumper"
2021-09-29 21:27:53 [INFO] No control files were defined in the path: "/tmp/obdumper"
2021-09-29 21:27:53 [INFO] No mapping files were defined in the path: "/tmp/obdumper"
obproxy Druid LogFactory, userDefinedLogType=null, logInfo=public com.alipay.oceanbase.obproxy.druid.s
upport.logging.Log4j2Impl(java.lang.String)
2021-09-29 21:27:53 [INFO] {dataSource-1} inited
2021-09-29 21:27:53 [INFO] {dataSource-2} inited
2021-09-29 21:27:53 [INFO] The manifest file: "/tmp/obdumper/data/MANIFEST.bin" has been saved
2021-09-29 21:27:54 [WARN] File: "/tmp/obdumper/data/MANIFEST.bin" is unmatched on the suffix[.csv], i
gnore it
2021-09-29 21:27:54 [WARN] File: "/tmp/obdumper/data/test/TABLE/t1-schema.sql" is unmatched on the suf
fix[.csv], ignore it
2021-09-29 21:27:54 [INFO] Binding table: "t1" to the file: "/tmp/obdumper/data/test/TABLE/t1.0.csv" f
inished
```

```
2021-09-29 21:27:54 [INFO] File: "/tmp/obdumper/data/test/TABLE/t1.0.csv" has not been splitted. 202
< 67108864
2021-09-29 21:27:54 [INFO] Splitted 1 csv subfiles by 64.0 MB. Elapsed: 10.49 ms
2021-09-29 21:27:54 [INFO] Generate 1 subfiles finished
2021-09-29 21:27:55 [INFO] Query table entry and primary key for table: t1 finished. Remain: 0
2021-09-29 21:27:55 [INFO] Query the leader location for the table: "t1". Remain: 0
2021-09-29 21:27:55 [INFO] Calculate leader: 172.xx.xxx.52:2881 of table: "t1", part: 0. Remain: 0
2021-09-29 21:27:55 [INFO] Waiting to refresh observer load status ......
2021-09-29 21:27:55 [INFO] Refresh the observer load status success. Table: "t1". Remain: 0
2021-09-29 21:27:55 [INFO] Refresh observer load status finished. Elapsed: 94.30 ms
2021-09-29 21:27:55 [INFO] Create 16384 slots for ring buffer finished. [172.xx.xxx.52:2881]
2021-09-29 21:27:55 [INFO] Start 11 database writer threads finished. [172.xx.xxx.52:2881]
2021-09-29 21:27:55 [INFO] Start 3 csv file reader threads successed
2021-09-29 21:27:55 [INFO] File: "/tmp/obdumper/data/test/TABLE/t1.0.csv" has been parsed finished
2021-09-29 21:27:56 [INFO] Wait for the all the workers to drain of published events then halt the wor
kers
2021-09-29 21:27:56 [INFO] {dataSource-2} closing ...
2021-09-29 21:27:56 [INFO] {dataSource-2} closed
2021-09-29 21:27:56 [INFO] Close count: 5
2021-09-29 21:27:56 [INFO] {dataSource-1} closing ...
2021-09-29 21:27:56 [INFO] {dataSource-1} closed
2021-09-29 21:27:56 [INFO] Close count: 2
2021-09-29 21:27:56 [INFO] Shutdown task context finished
2021-09-29 21:27:56 [INFO]
Finished Tasks: 1       Running Tasks: 0        Progress: 100.00%
2021-09-29 21:27:56 [INFO]

All Load Tasks Finished:

-----------------------------------------------------------------------------------------------------
----------------------
        No.#        |        Type        |             Name             |            Coun
t             |       Status
-----------------------------------------------------------------------------------------------------
----------------------
         1          |       TABLE        |              t1              |            8 ->
8            |      SUCCESS
-----------------------------------------------------------------------------------------------------
----------------------

Total Count: 8          End Time: 2021-09-29 21:27:56


2021-09-29 21:27:56 [INFO] Load record finished. Total Elapsed: 2.140 s
2021-09-29 21:27:56 [INFO] System exit 0
```

检查导入的表结构和数据:

```
MySQL [test]> show create table test2.t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `c1` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) AUTO_INCREMENT = 1 DEFAULT CHARSET = utf8mb4 ROW_FORMAT = COMPACT COMPRESSION = 'zstd_1.3.8' R
1 row in set (0.012 sec)
```

```
MySQL [test]> select * from test2.t1;
+----+---------------------+
| id | c1                  |
+----+---------------------+
|  1 | 2021-09-29 21:16:03 |
|  2 | 2021-09-29 21:16:05 |
|  3 | 2021-09-29 21:16:05 |
|  4 | 2021-09-29 21:16:06 |
|  5 | 2021-09-29 21:16:06 |
|  6 | 2021-09-29 21:16:18 |
|  7 | 2021-09-29 21:16:18 |
|  8 | 2021-09-29 21:16:19 |
+----+---------------------+
8 rows in set (0.015 sec)
```

常见导出或导入问题

多次导出时没有换目录导致导入数据时出现非预期的结果

OBDUMPER 导出的时候需要指定目录。如果多次导出不同数据库或表时，使用了同一个目录，导出文件会混在一起,同名的文件会被覆盖。在用 OBLOADER 导入的时候，可能会导入非预期内的数据。OBLOADER 导入是解析这个目录结构自动获取要导入的表。

所以，如果需要多次导出，建议每次导出时，导出目录都不要重复。或者在导出之前，清空导出目录。

外部文件不符合要求导致导入失败

该问题需要分情况讨论:

- 外部文件是 SQL 文件
  - SQL 内容是 DDL，要求 SQL 里不能有注释和 SET 开关语句等。
  - SQL 内容是 DML， 要求 SQL 里面只有 INSERT 语句，并且每个语句不换行。

说明

这类文件格式很麻烦，建议不要使用 OBLOADER 执行，改为在 MySQL 命令行下直接执行。

- 外部文件如果是 CSV 文件

  CSV 文件要符合标准定义。要求有列分隔符、行分隔符。列尽量用特定字符串（通常是双引号）包括起来。同时对于被包括起来的列里面出现双引号要实现转义。

导出性能问题

导出表结构的时候，如果表很多，不能开启太多并发， `--threads` 控制在 4 以内。太高的并发没有意义，会加重

sys 租户访问内部视图的负担，进而出现导出超时报错。

导出表数据的时候，如果表很多，可以开启并发，`--threads` 默认会根据 CPU 动态调整，也可以手动指定，以控制导出对主机性能的影响。

导出或导入大量的数据时，可能瓶颈处在 OBLOADER 和 OBDUMPER 自身，这时候可以编辑这两个文件，调大里面的 Java 参数 `Xms` 和 `Xmx` 后面的值，分别表示 Java 初始内存和最大可用内存。

```
vim bin/obdumper  或 vim bin/obloader

 50 JAVA_OPTS="$JAVA_OPTS -server -Xms4G -Xmx4G -XX:MetaspaceSize=512M -XX:MaxMetaspaceSize=512M -Xss3
52K"
```

导入性能问题

导入表结构时，指定 `--threads` 并发数不要超过 2 。并发 DDL 并不会一定加快速度，在 OceanBase 数据库里，DDL 是串行执行的。每个 DDL 平均耗时 1s 。

导入大量数据时，可以开启并发，`--threads` 默认会根据 CPU 动态调整，也可以手动指定，以控制导出对主机性能的影响。

导入的性能还受表上的索引影响。对于很大的表，建议建表的时候，除了主键索引外，其他唯一索引、普通索引都留到数据导入结束后再创建。

导入的性能还受租户的增量内存写入速度限制。如果增量内存不足时会触发合并或转储。合并比较耗性能，应该尽量避免不要触发。可以开启内存的转储，并将转储的次数设置为 100 以上（通常足够坚持到 24h ）。如果增量内存使用率达到租户限速阈值时，导入性能也会下降。如果增量内存使用率满了，会出现导入报错的现象，这个容易导致数据导入前功尽弃。

所以租户限速的阈值建议不要高于 90 。转储相关参数的设置跟租户内存的大小、写入速度都有关系，需要根据实际情况调优。

更多使用问题请参考官方 产品文档。

4.9 如何使用 DataX 迁移 OceanBase 数据到 MySQL/ORACLE

OceanBase 数据同步到 MySQL

使用 OceanBase Reader 和 MySQL 的 Writer 搭配实现 OceanBase 数据同步到 MySQL。

配置文件如下:

```
{
    "job": {
        "setting": {
            "speed": {
                "channel": 16
            },
            "errorLimit": {
                "record": 0,
                "percentage": 0.1
            }
        },
        "content": [
            {
                "reader": {
                    "name": "oceanbasev10reader",
                    "parameter": {
                        "where": "",
                        "readBatchSize": 10000,
                        "column": [
                            "*"
                        ],
                        "connection": [
                            {
                                "jdbcUrl": ["||_dsc_ob10_dsc_||obdemo:oboracle||_dsc_ob10_dsc_||jdbc:o
ceanbase://127.0.0.1:2883/tpcc"],
                                "table": [
                                    "bmsql_oorder"
                                ]
                            }
                        ],
                        "username": "tpcc",
                        "password":"********"
                    }
                },
                "writer": {
                    "name": "mysqlwriter",
                    "parameter": {
                        "writeMode": "replace",
                        "username": "tpcc",
                        "password": "******",
                        "column": [
```

```
                            "*"
                        ],
                        "session": [
                         "set session sql_mode='ANSI'"
                        ],
                        "preSql": [
                            "truncate table bmsql_oorder"
                        ],
                        "batchSize": 512,
                        "connection": [
                            {
                                "jdbcUrl": "jdbc:mysql://127.0.0.1:3306/tpccdb?useUnicode=true&characte
rEncoding=utf8",
                                "table": [
                                    "bmsql_oorder"
                                ]
                            }
                        ]
                    }
                }
            }
        ]
    }
}
```

OceanBase 数据同步到 Oracle

配置文件如下:

```
{
    "job": {
        "setting": {
            "speed": {
                "channel": 16
            },
            "errorLimit": {
                "record": 0,
                "percentage": 0.1
            }
        },
        "content": [
            {
                "reader": {
                    "name": "oceanbasev10reader",
                    "parameter": {
                        "where": "",
                        "readBatchSize": 10000,
                        "column": [
                            "*"
                        ],
                        "connection": [
                            {
                                "jdbcUrl": ["||_dsc_ob10_dsc_||obdemo:oboracle||_dsc_ob10_dsc_||jdbc:o
ceanbase://127.0.0.1:2883/tpcc"],
```

```
                                "table": [
                                    "bmsql_oorder"
                                ]
                            }
                        ],
                        "username": "tpcc",
                        "password":"********"
                    }
                },
                "writer": {
                    "name": "oraclewriter",
                    "parameter": {
                        "username": "tpcc",
                        "password": "********",
                        "column": [
                            "*"
                        ],
                        "preSql": [
                            "truncate table bmsql_oorder"
                        ],
                        "batchSize": 512,
                        "connection": [
                            {
                                "jdbcUrl": "jdbc:oracle:thin:@127.0.0.1:1521:helowin",
                                "table": [
                                    "bmsql_oorder"
                                ]
                            }
                        ]
                    }
                }
            }
        ]
    }
}
```

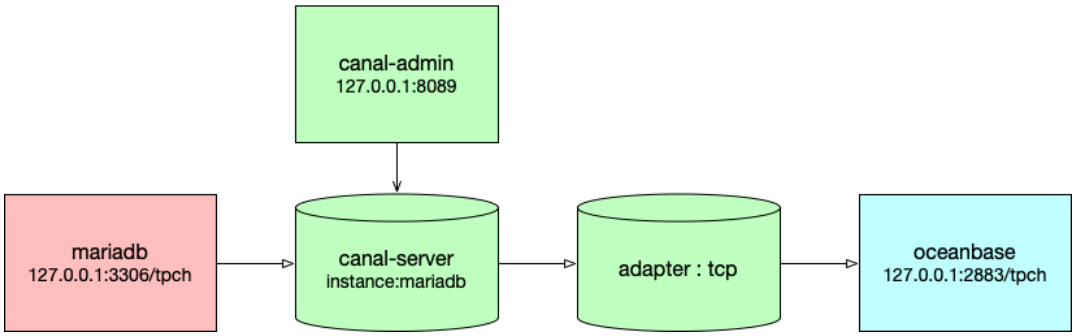
4.10 如何使用 Canal 将 MySQL 数据实时同步到 OceanBase

`Canal` 是 Alibaba 开源的一个产品，主要用途是基于 MySQL 数据库增量日志解析，提供增量数据订阅和消费。开源项目地址：https://github.com/alibaba/canal。

架构原理

`canal` 主要提供了 4 个组件：

- `canal deployer`：`canal` 的 server 端，进行 `binlog` 到 `CanalEntry` 的转换。
- `canal admin`：`canal` 的配置管理服务，提供 web 页面管理 `canal` 的 `server` 端服务。
- `canal adapter`：`canal` 的客户端适配器，解析 `CanalEntry` 并将增量变动同步到目的端。
- `canal example`：`canal` 的 `client` 端示例，用户可以基于该部分代码实现自己的消费逻辑。



图中 `canal-admin` 是部署用的（可选），`canal-server` 就是 `canal deployer` 软件，`adapter` 就是 `canal adapter` 软件。源端数据库是 `mariadb`，目标端是 OceanBase 的 mysql 租户。

Canal Deployer

`Canal Deployer` 的服务中有 `Server` 和 `Instance` 的概念，一个 `server` 代表一个 `deployer` 服务，一个 `instance` 代表一个实际的数据同步通路，在 `Canal Server` 中，一个 `server` 可以有多个 `instance`。

`Canal Instance` 由 `Spring` 在运行时创建，其配置信息在 `canal deployer` 的 `conf/canal.properties` 中指定。`Canal` 本身提供了几种可以直接使用的配置，存放在 `conf/spring` 目录下。

`Canal Instance` 在解析完日志信息后，得到的 `CanalEntry` 数据会放入内存等待消费。Canal 提供了两种消费方式供用户选择：

- TCP 模式：直接使用客户端连接 Canal 消费数据。
- MQ 模式：先将 Canal 内存中的数据写入 MQ，用户可以使用客户端连接 MQ 进行数据消费。

Canal 中有两种位点信息，一个是解析位点，即日志转化 `Entry` 的过程记录的位点，由 `LogPositionManager` 管理，另一个是客户端消费的位点，由 `MetaManager` 管理。两者同样是在 `instance` 的 `spring xml` 文件进行配置。

Canal Adapter

`Canal Adapter` 用于消费 `CanalEntry`，并写入对应的目的容器。`adapter` 与 `deployer` 一样有 `instance` 的概念，实际运行时，`adapter` 本身由 `adapter launcher` 服务启动，并根据用户配置生成 `adapter instance`，由 `instance` 执行具体的 `CanalEntry` 读取和目的端写入的工作。

Canal Admin

`Canal Admin` 就是为了简化部署操作而引入的一个管理平台服务。`Canal Deployer` 和 `Canal Instance` 都分别支持单机部署和高可用集群部署两种模式，通过 `Canal Admin`，用户可以通过 web 页面来方便地管理 `Canal Deployer` 和 `Canal Instance` 的部署，同样也是支持单机部署和高可用集群化部署。

部署示例

MySQL 准备

这里使用的是 `mariadb`。

```
yum install mariadb mariadb-server
```

对于自建 MySQL , 需要先开启 Binlog 写入功能，配置 `binlog-format` 为 ROW 模式，`/etc/my.cnf` 中配置如下:

```
[mysqld]
log-bin=mysql-bin # 开启 binlog
binlog-format=ROW # 选择 ROW 模式
server_id=1 # 配置 MySQL replaction 需要定义，不要和 canal 的 slaveId 重复
```

授权 canal 链接 MySQL 账号具有作为 `MySQL slave` 的权限, 如果已有账户可直接 `grant`。

```
CREATE USER canal IDENTIFIED BY 'c****l';
GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'canal'@'%';
-- GRANT ALL PRIVILEGES ON *.* TO 'canal'@'%' ;
FLUSH PRIVILEGES;
```

然后初始化了一个业务数据库（TPCH，后面介绍性能测试的时候再重点介绍）。

部署 Canal Admin

用 Canal Admin 部署 Canal Deployer 会方便一些，这个不是必须的。

本文使用 Canal Admin。

- 下载 Canal Admin，访问地址：https://github.com/alibaba/canal/releases

```
wget https://github.com/alibaba/canal/releases/download/canal-1.1.5/canal.admin-1.1.5.tar.gz
```

- 解压缩到指定目录

```
mkdir ~/canal-admin && tar zxvf canal.admin-1.1.5.tar.gz -C ~/canal-admin/
```

- 修改配置文件

```
cd ~/canal-admin && vim conf/application.yml
```

内容如下：

```
server:
  port: 8089
spring:
  jackson:
    date-format: yyyy-MM-dd HH:mm:ss
    time-zone: GMT+8
spring.datasource:
  address: 127.0.0.1:3306
  database: canal_manager
  username: canal
  password: c****l
  driver-class-name: com.mysql.jdbc.Driver
  url: jdbc:mysql://${spring.datasource.address}/${spring.datasource.database}?useUnicode=true&cha
racterEncoding=UTF-8&useSSL=false
  hikari:
    maximum-pool-size: 30
    minimum-idle: 1
canal:
  adminUser: admin
  adminPasswd: admin
```

初次测试时，建议不要修改上面密码 `adminPasswd`，以免后面密码修改不对导致连接不上。

- 初始化元数据库

```
mysql -h127.1 -uroot -P3306 -p
source conf/canal_manager.sql
```

脚本会自动创建相关表，如下：

```
show tables;
MariaDB [tpch]> use canal_manager;
Database changed
MariaDB [canal_manager]> show tables;
+-------------------------+
```

```
| Tables_in_canal_manager |
+-------------------------+
| canal_adapter_config    |
| canal_cluster           |
| canal_config            |
| canal_instance_config   |
| canal_node_server       |
| canal_user              |
+-------------------------+
6 rows in set (0.00 sec)
```

- 启动 web 服务

```
cd ~/canal-admin && bin/startup.sh
```

正常情况下，启动成功会监听 8089 端口。

```
[root@obce00 adapter]# netstat -ntlp |grep 15973
tcp        0      0 0.0.0.0:8089            0.0.0.0:*               LISTEN      15973/java
```

启动如果有问题，可以查看日志。

```
vim logs/admin.log +
```

- 登录 web 界面

Canal Admin 的 web 访问地址：http://127.0.0.1:8089/

登录用户名：`admin`

登录密码：`123456`

部署 Canal Deployer

- 下载 canal，访问地址：https://github.com/alibaba/canal/releases。

```
wget https://github.com/alibaba/canal/releases/download/canal-1.1.5/canal.deployer-1.1.5.tar.gz
```

- 解压缩

```
mkdir ~/canal && tar zxvf canal.deployer-1.1.5.tar.gz  -C ~/canal
```

- 修改配置

如果不使用 Canal Admin 部署，则使用默认的配置文件 `conf/canal.properties` 和 `conf/example/instance.properties`。这个是默认创建了一个 instance 叫 `example`。需要修改 `example` 的实例配置文件，修改数据库连接地址、用户名和密码。

```
vi conf/example/instance.properties

# mysql serverId
canal.instance.mysql.slaveId = 1234
#position info，需要改成自己的数据库信息
canal.instance.master.address = 127.0.0.1:3306
canal.instance.master.journal.name =
canal.instance.master.position =
canal.instance.master.timestamp =
#canal.instance.standby.address =
#canal.instance.standby.journal.name =
#canal.instance.standby.position =
#canal.instance.standby.timestamp =
#username/password，需要改成自己的数据库信息
canal.instance.dbUsername = canal
canal.instance.dbPassword = canal
canal.instance.defaultDatabaseName =
canal.instance.connectionCharset = UTF-8
#table regex
canal.instance.filter.regex = .*\\..*
```

`canal.instance.connectionCharset` 代表数据库的编码方式对应到 Java 中的编码类型，比如 `UTF-8`，`GBK`，`ISO-8859-1`。如果系统是 1 个 CPU，需要将 `canal.instance.parser.parallel` 设置为 `false`。

如果使用 Canal Admin 部署 `server` 和 `instance`，则需将配置文件 `conf/canal.properties` 中的内容替换为配置文件 `conf/canal_local.properties` 中的内容，并修改内容替换后的 `conf/canal.properties` 文件中的 `manager` 地址，其他参数值可以保持默认。

```
[root@obce00 canal]# cat conf/canal.properties
# register ip
canal.register.ip =
# canal admin config
canal.admin.manager = 127.0.0.1:8089
canal.admin.port = 11110
canal.admin.user = admin
canal.admin.passwd = 4ACFE3202A5FF5CF467898FC58AAB1D615029441
# admin auto register
canal.admin.register.auto = true
canal.admin.register.cluster =
canal.admin.register.name =
```

注意

`passwd` 后面的字符串是 `admin` 在 MySQL 里的密文。这个密码跟前面 Canal Admin 配置文件里的密码保持一致。如果前面密码改了，这里也要相应修改，密文的值可以通过 MySQL 的 `password` 方法获取。

```
MariaDB [canal_manager]> select password('admin');
+-------------------------------------------+
| password('admin')                         |
+-------------------------------------------+
| *4ACFE3202A5FF5CF467898FC58AAB1D615029441 |
+-------------------------------------------+
```

```
1 row in set (0.00 sec)
```

- 启动 Cananl Server

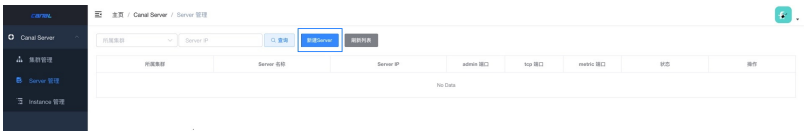
  不管是那种部署方法，配置文件修改好后，就可以启动服务。

```
sh bin/startup.sh
```

- （可选）图形化部署 Canal server 和 Canal instance

  如果使用 Canal Admin 管理 Canal server，则登录 admin 的管理界面 http://172.24.50.39:8089/#/canalServer/nodeServers，选择 **Canal Server** > **Server 管理**，单击 **新建 Server**。

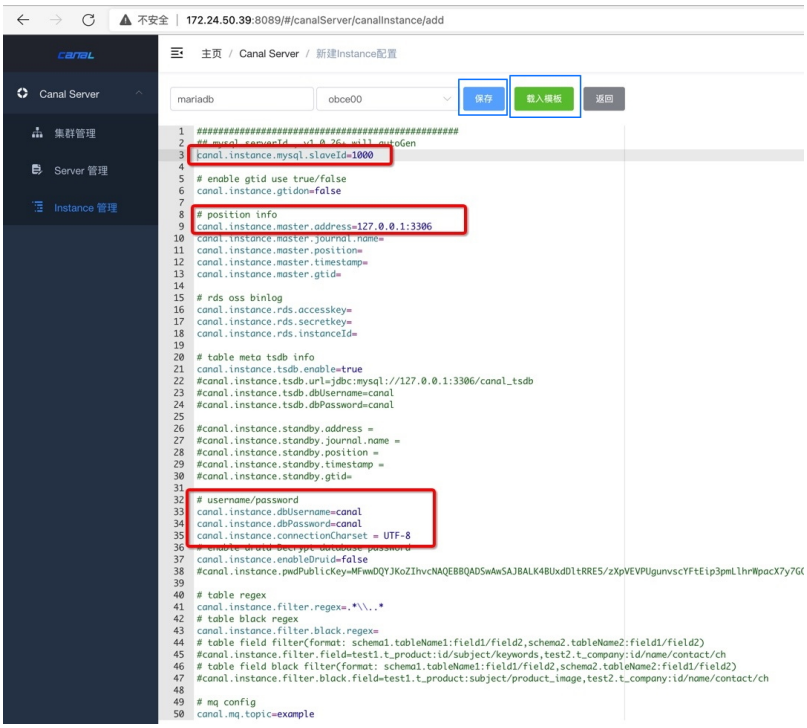
  > **说明**
  >
  > 此处链接中的 IP 为示例中配置 Canal Admin 的服务器 IP，仅供参考。您需根据实际情况将其转换为自身配置 Canal Admin 的服务器 IP。



在弹出的 **新建Server信息** 界面单击 **确定** 按钮。

之后，选择 **Canal Server** > **Instance 管理** ，单击 **新建 Instance**。

单击 **载入模板** ，显示一个配置文件，跟前面看到的类似。修改配置文件中的跟源端 `canal` 和数据库有关的信息。



单击 **保存**，命名为 `mariadb`。

保存后的 instance 状态为 **停止**，单击 **操作** > **启动** 可启动 instance，启动后，instance 状态为 **启动**。

- 查看 server 日志

  可以命令行下查看日志，或者在 `Canal Admin` 里查看 `server` 的日志。

```
vi logs/canal/canal.log
```

- 查看 instance 的日志

  可以命令行下查看日志，或者在 `Canal Admin` 里查看 `instance` 的日志。

```
tail -f logs/canal/canal.log
tail -f logs/example/example.log
tail -f logs/mariadb/mariadb.log
```

- 停止服务

```
sh bin/stop.sh
```

部署 RDB 适配器

Canal Adapter 提供了对多种目标容器的支持，对于 OceanBase 社区版来说，主要使用它的 `rdb` 模块，目的端容器为 MySQL 或社区版 OceanBase 。

Adapter 的需要手动部署。

- 下载 Canal Adapter，访问地址：https://github.com/alibaba/canal/releases

```
wget https://github.com/alibaba/canal/releases/download/canal-1.1.5/canal.adapter-1.1.5.tar.gz
```

- 解压到指定目录

```
mkdir ~/adapter && tar zxvf canal.adapter-1.1.5.tar.gz -C ~/adapter/
```

- 修改启动器配置：`conf/application.yml`，这里以 OceanBase 目标库为例。

  首先指定 adapter 源端类型，通过 `mode` 指定。这里选择 `tcp`。后面就要指定 `canal.tcp` 相关属性，包括 canal server 的 IP 和 端口，数据库的连接用户和密码。之后指定 adapter 目标端连接信息。`instance` 是源端实例名称，在 canal 部署的时候定义的。

  说明

  如果没有用 Canal Admin 部署，沿用的是 `example` 这个名称；如果用了 Canal Admin 部署 `instance`，前面命名的是 `mariadb`。

  `key` 是自定义，名字后面有用。`jdbc` 相关属性是目标端 OceanBase MySQL 的连接方式，可以使用 MySQL 自带的驱动。

```
mode: tcp #tcp kafka rocketMQ rabbitMQ
  flatMessage: true
  zookeeperHosts:
  syncBatchSize: 1000
  retries: 0
  timeout:
  accessKey:
  secretKey:
  consumerProperties:
   # canal tcp consumer
    canal.tcp.server.host: 127.0.0.1:11111
    canal.tcp.zookeeper.hosts:
    canal.tcp.batch.size: 500
    canal.tcp.username: tpch
    canal.tcp.password: D********V
  canalAdapters:
  - instance: mariadb # canal instance Name or mq topic name
    groups:
    - groupId: g1
      outerAdapters:
    - name: logger
```

```
      - name: rdb
        key: obmysql
        properties:
          jdbc.driverClassName: com.mysql.jdbc.Driver
          jdbc.url: jdbc:mysql://127.0.0.1:2883/tpch?useUnicode=true
          jdbc.username: tpch@obmysql#obdemo
          jdbc.password: D********V
```

- RDB 映射文件

  修改 `conf/rdb/mytest_user.yml` 文件。

  映射有两种：一是按表映射；二是整库映射。下面示例是整库映射。

```
[root@obce00 adapter]# cat conf/rdb/mytest_user.yml
#dataSourceKey: defaultDS
#destination: example
#groupId: g1
#outerAdapterKey: mysql1
#concurrent: true
#dbMapping:
#  database: mytest
#  table: user
#  targetTable: mytest
#  targetPk:
#    id: id
##  mapAll: true
#  targetColumns:
#    id:
#    name:
#    role_id:
#    c_time:
#    test1:
#  etlCondition: "where c_time>={}"
#  commitBatch: 3000 # 批量提交的大小
# Mirror schema synchronize config
dataSourceKey: defaultDS
destination: mariadb
groupId: g1
outerAdapterKey: obmysql
concurrent: true
dbMapping:
  mirrorDb: true
  database: tpch
  commitBatch: 1000
```

  其中，`destination` 指定的是 `canal instance` 名称；`outerAdapterKey` 是前面定义的 `key`；`mirrorDb` 指定数据库级别 DDL 和 DML 镜像同步。

  导入的类型以目标表的元类型为准, 将自动进行类型转换。

- 启动 RDB

  如果使用了 OceanBase 的驱动，则将目标库 OceanBase 驱动包放入 lib文件夹。

启动 canal-adapter 启动器。

```
bin/startup.sh
```

验证修改 `mysql mytest.user` 表的数据, 将会自动同步到 MySQL 的 `MYTEST.TB_USER` 表下面, 并会打出 DML 的 log。

- 停止 RDB

```
bin/stop.sh
```

- 查看 RDB 日志

```
tail -f logs/adapter/adapter.log
2021-12-09 09:56:04.148 [pool-6-thread-1] DEBUG c.a.o.canal.client.adapter.rdb.service.RdbSyncSe
rvice - DML: {"data":{"s_suppkey":99995,"s_name":null,"s_address":null,"s_nationkey":null,"s_phon
e":null,"s_acctbal":null,"s_comment":null},"database":"tpch","destination":"mariadb","old":null,
"table":"supplier2","type":"INSERT"}
2021-12-09 09:56:04.149 [pool-6-thread-1] DEBUG c.a.o.canal.client.adapter.rdb.service.RdbSyncSe
rvice - DML: {"data":{"s_suppkey":99998,"s_name":null,"s_address":null,"s_nationkey":null,"s_phon
e":null,"s_acctbal":null,"s_comment":null},"database":"tpch","destination":"mariadb","old":null,
"table":"supplier2","type":"INSERT"}
2021-12-09 10:13:35.915 [Thread-3] INFO  c.a.o.canal.client.adapter.rdb.monitor.RdbConfigMonito
r - Change a rdb mapping config: mytest_user.yml of canal adapter
```

同步测试

可以对源端 MySQL 数据库 `tpch` 做 DML 和 DDL 测试，都可以同步到目标端。这里就不详细展开，只介绍一些已知的功能限制。

- 同步的表必须有主键。否则，源端删除无主键表的任意一笔记录，同步到目标端会导致整个表被删除。
- DDL 支持新建表、新增列。但受 OceanBase MySQL 租户功能限制，不支持后期加主键、修改列的类型（大类型变更，如数值、字符串、日期之间类型变化）。

4.11 如何使用 Canal 将 OceanBase 数据实时同步到 MySQL

什么是 CDC

CDC 全称是 Change Data Capture，即变更数据捕获。

为什么需要 CDC 功能

CDC 能够帮助识别上次提取之后发生变化的数据。CDC 提供的数据可以做很多事情，比如：做历史库、做近实时缓存、提供给消息队列（MQ），用户消费 MQ 做分析和审计等。

OceanBase CDC 实现逻辑

oblogproxy 是 OceanBase 数据库的增量日志代理服务。基于 liboblog，以服务的形式提供实时增量链路接入和管理能力，方便应用接入 OceanBase 增量日志。能够解决在网络隔离的情况下，订阅增量日志的需求，并提供多种链路接入方式。

Canal 是开源 MySQL 数据库 Binlog 的增量订阅和消费组件，基于 MySQL 数据库的增量日志解析，可以用于数据同步。

数据链路:

```
ob_cluster -> oblogreader -> oblogmsg -> canal_server -> canal_client -> mysql
```

组件介绍:

- liboblog 是 OceanBase CDC 的基本组件，liboblog 衍生出 oblogproxy，liboblog 依赖 oblogmsg。
- liboblog 是 C++ 动态库，将从 OceanBase 集群中拉到的增量日志按事务提交顺序向外透出，透出的方式是按 LogMessage 协议创建相关对象给使用方使用。
- ObLogReader 对 liboblog 的 C++ 封装，在 liboblog 的基础上，对创建出的 LogMessage 对象适配多种序列化输出。
- logproxy 增量日志代理，并不对增量数据进行转发，而只是对新建连接请求创建对应的 oblogreader 子进程，并由该 oblogreader 子进程通过该连接直接向客户端发送按 LogMessage 序列化后的增量日志记录。LogProxy 用于对同机器上所有 ObLogReader 子进程的生命周期进行管理。
- logclient 对字节流按 LogMessage 协议反序列化后生成 LogMessage 对象给使用方调用，比如和 canal 对接。

安装 oblogproxy

- 下载 oblogproxy rpm 包

  下载地址为：https://github.com/oceanbase/oblogproxy/releases/tag/v1.0.0

- 安装 oblogproxy

```
yum install oblogproxy-1.0.0-1.el7.x86_64.rpm
```

- 确认 oblogproxy 依赖，重点是确认有 liboblog。

```
[root@172.xx.xxx.49 ~]$cd /usr/local/oblogproxy/

[root@172.xx.xxx.49 oblogproxy]$ldd ./bin/logproxy
linux-vdso.so.1 =>  (0x00007fffe68fe000)
liboblog.so.1 => /lib/liboblog.so.1 (0x00002ae0113f1000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00002ae046bf8000)
libdl.so.2 => /lib64/libdl.so.2 (0x00002ae046e14000)
librt.so.1 => /lib64/librt.so.1 (0x00002ae047018000)
libm.so.6 => /lib64/libm.so.6 (0x00002ae047220000)
libc.so.6 => /lib64/libc.so.6 (0x00002ae047522000)
/lib64/ld-linux-x86-64.so.2 (0x00002ae0111cd000)
libaio.so.1 => /lib64/libaio.so.1 (0x00002ae0478f0000)
```

启动 oblogproxy

修改 oblogproxy 参数

- 生成加密用户和对应的密码

```
[root@172.xx.xxx.49 bin]$pwd
/usr/local/oblogproxy/bin

[root@172.xx.xxx.49 bin]$./logproxy -x root@sys
EA87*****************E1556E917

[root@172.xx.xxx.49 bin]$./logproxy -x Root@2021
8852D*****************A9D8FD52
```

- 修改对应的 conf.json 参数

```
[root@172.xx.xxx.49 conf]$pwd
/usr/local/oblogproxy/conf

[root@172.xx.xxx.49 conf]$ls -l
总用量 4
-rw-r--r-- 1 root root 1081 10月 25 18:00 conf.json
```

```
# 修改conf.json的内容
"ob_sys_username": "EA87*****************E1556E917",
"ob_sys_password": "8852D*****************A9D8FD52",
```

- 启动 oblogproxy

```
[root@172.xx.xxx.49 oblogproxy]$./run.sh start
work path : /usr/local/oblogproxy
is_running : (8252)/usr/local/oblogproxy logproxy is running !
logproxy started!
```

- 启停 obproxy 和查看状态
  - 启动 oblogproxy

```
[root@172.xx.xxx.49 oblogproxy]$pwd
/usr/local/oblogproxy
[root@172.xx.xxx.49 oblogproxy]$./run.sh start
```

  - 停止 oblogproxy

```
[root@172.xx.xxx.49 oblogproxy]$pwd
/usr/local/oblogproxy
[root@172.xx.xxx.49 oblogproxy]$./run.sh stop
```

  - 查看 oblogproxy 状态

```
[root@172.xx.xxx.49 oblogproxy]$pwd
/usr/local/oblogproxy
[root@172.xx.xxx.49 oblogproxy]$./run.sh status
```

说明:

oblogproxy 启动成功后进程确认。

```
[root@172.xx.xxx.49 ~]$ps -ef | grep logproxy | grep -v grep
root     26379 26373  0 10:45 pts/1    00:00:00 ./bin/logproxy -f ./conf/conf.json
[root@172.xx.xxx.49 ~]$
```

当有一个 client 连接成功后会 fork 一个子进程。

```
[root@172.xx.xxx.49 ~]$ps -ef | grep oblogreader | grep -v grep
root     26386 26379  2 10:45 pts/1    00:00:17 oblogreader    -f ./conf/conf.json
```

canal server

下载 canal-for-ob。canal-for-ob 详细信息请参考 canal-for-ob GitHub 仓库。

修改 `/opt/canal_ob/canal.deployer-for-ob-rc1/conf/canal.properties`。

```
canal.zkServers = 172.xx.xxx.47:12181,172.xx.xxx.48:12181,172.xx.xxx.49:12181
canal.serverMode = tcp
canal.destinations = obtest2
canal.instance.global.spring.xml = classpath:spring/ob-default-instance.xml
```

canal instance

```
[root@172.xx.xxx.49 conf]$ll
总用量 28
-rwxr-xr-x 1 root root  319 10月 21 16:15 canal_local.properties
-rwxr-xr-x 1 root root 6577 10月 26 16:21 canal.properties
-rwxr-xr-x 1 root root 3592 10月 21 16:15 logback.xml
drwxrwxrwx 2 root root 4096 10月 21 16:15 metrics
drwxr-xr-x 2 root root 4096 12月  7 13:29 obtest2  -- 将 example 修改为 obtest2
drwxrwxrwx 3 root root 4096 11月  2 10:51 spring
```

```
[root@172.xx.xxx.49 obtest2]$pwd
/opt/canal_ob/canal.deployer-for-ob-rc1/conf/obtest2
[root@172.xx.xxx.49 obtest2]$ll
total 16
-rwxrwxrwx 1 root root 1147 Jan 13 17:15 ca.crt
-rwxrwxrwx 1 root root 1241 Jan 13 17:15 client.crt
-rwxrwxrwx 1 root root 1708 Jan 13 17:15 client.key
-rwxr-xr-x 1 root root 1743 Feb 13 17:15 instance.properties
-rwxr-xr-x 1 root root 1743 Feb 13 17:15 ob-instance.properties

# 删除 instance.properties，将 ob-instance.properties 重命名为 instance.properties

[root@172.xx.xxx.49 obtest2]$cat instance.properties
```

```
# oceanbase集群参数
canal.instance.oceanbase.rsList=172.xx.xxx.49:2882:2881;172.xx.xxx.47:2882:2881
canal.instance.oceanbase.username=root@tenant#obcluster
canal.instance.oceanbase.password=Ro******21
canal.instance.oceanbase.startTimestamp=0

# oceanbase logproxy参数
canal.instance.oceanbase.logproxy.address=127.0.0.1:2983
canal.instance.oceanbase.logproxy.sslEnabled=false
canal.instance.oceanbase.logproxy.serverCert=../conf/${canal.instance.destination:}/ca.crt
canal.instance.oceanbase.logproxy.clientCert=../conf/${canal.instance.destination:}/client.crt
canal.instance.oceanbase.logproxy.clientKey=../conf/${canal.instance.destination:}/client.key

# 是否要在库名中去掉租户前缀。logproxy 输出的日志中库名默认为 [tenant].[db]
canal.instance.parser.excludeTenantInDbName=true
canal.instance.oceanbase.tenant=test_tenant

# 日志过滤。格式为 [tenant].[database].[table]，支持正则
canal.instance.filter.regex=test_tenant.db1.*
```

启动 canal server

```
[root@172.xx.xxx.49 ~]$cd /opt/canal_ob
[root@172.xx.xxx.49 canal_ob]$cd canal.deployer-for-ob-rc1
[root@172.xx.xxx.49 canal.deployer-for-ob-rc1]$cd bin
[root@172.xx.xxx.49 bin]$ls -lrt
总用量 20
-rwxr-xr-x 1 root root 1244 10月 21 16:15 startup.bat
-rwxr-xr-x 1 root root 1356 10月 21 16:15 stop.sh
-rwxr-xr-x 1 root root 3167 10月 21 16:15 startup.sh
-rwxr-xr-x 1 root root  226 10月 21 16:15 restart.sh
-rw-r--r-- 1 root root    6 12月  7 10:42 canal.pid
[root@172.xx.xxx.49 bin]$./startup.sh
```

canal client

下载 canal-for-ob。更多信息，参考 canal-for-ob GitHub 仓库。

```
[root@172.xx.xxx.49 conf]$pwd
/opt/canal_ob/canal_adapter_for_ob/conf
[root@172.xx.xxx.49 conf]$ls -lrt
总用量 44
-rwxr-xr-x 1 root root 2172 10月 21 16:15 logback.xml
drwxrwxrwx 2 root root 4096 10月 21 16:15 tablestore
drwxr-xr-x 2 root root 4096 10月 21 16:15 hbase
drwxr-xr-x 2 root root 4096 10月 21 16:15 kudu
drwxrwxrwx 2 root root 4096 10月 21 16:15 META-INF
drwxr-xr-x 2 root root 4096 10月 21 16:15 es6
-rwxr-xr-x 1 root root  170 10月 21 16:15 bootstrap.yml
-rwxr-xr-x 1 root root  552 10月 21 16:15 mytest_user.yml
drwxr-xr-x 2 root root 4096 10月 21 16:15 es7
-rwxr-xr-x 1 root root 3240 10月 26 16:26 application.yml
drwxrwxrwx 2 root root 4096 12月  7 10:27 rdb
[root@172.xx.xxx.49 conf]$cat application.yml
```

```
canalAdapters:
  - instance: obtest2# canal instance Name or mq topic name
    groups:
    - groupId: g1
      outerAdapters:
      - name: logger
      - name: rdb
        key: mysql1
        properties:
          jdbc.driverClassName: com.mysql.jdbc.Driver
          jdbc.url: jdbc:mysql://172.xx.xxx.49:3367/mysql?useUnicode=true&useSSL=false
          jdbc.username: root
          jdbc.password: Ro******21
```

修改适配器库/表映射（以库映射为例）

```
[root@172.xx.xxx.49 conf]$pwd
/opt/canal_ob/canal_adapter_for_ob/conf
[root@172.xx.xxx.49 conf]$ls -l rdb
总用量 4
-rwxr-xr-x 1 root root 186 12月  6 21:11 mytest_user.yml
```

```
## Mirror schema synchronize config
dataSourceKey: defaultDS
destination: obtest2
groupId: g1
outerAdapterKey: mysql1
concurrent: true
dbMapping:
  mirrorDb: true
  database: db1
```

启动 canal client

```
[root@172.xx.xxx.49 bin]$pwd
/opt/canal_ob/canal_adapter_for_ob/bin

[root@172.xx.xxx.49 bin]$ls -lrt
总用量 16
-rwxr-xr-x 1 root root 2289 10月 21 16:15 startup.sh
-rwxr-xr-x 1 root root  793 10月 21 16:15 startup.bat
-rwxr-xr-x 1 root root  205 10月 21 16:15 restart.sh
-rwxr-xr-x 1 root root 1370 10月 21 16:15 stop.sh
[root@172.xx.xxx.49 bin]$./startup.sh
```

在 OceanBase 源端写入数据，在 MySQL 目标端查看数据同步。

4.12 如何对 OceanBase 迁移性能进行简单分析和调优

DataX 的调优建议

DataX 本质上是个数据交换平台，将源端的数据读出，写入到目标端。其数据迁移性能取决于下面几个因素:

- 源端的读性能。可以加并发，制约条件是对源库的影响、源库的性能瓶颈等。
- DataX 自身的性能。DataX 是个 Java 程序，其能起的线程数有限，受限于所在主机的 CPU 和内存大小。
- 网络传输性能。并发高的时候，网络传输要留意吞吐量是否达到网卡瓶颈。现在万兆网卡的吞吐量 10000Mb，很难达到。不过占用网络带宽对其他业务可能也会有影响。
- 目标端的写入性能。也可以加并发，制约条件就是目标库写入性能瓶颈、对目标库的影响等。如果目标端是 OceanBase 数据库，需要针对 OceanBase 数据库调优。
- 涉及到文件数据源时，关注文件所在磁盘 IO 性能。如 iops、吞吐量等。

所以 DataX 的调优就是调节 `reader` 和 `writer` 的各个并行参数，尽可能的把源数据库和目标端数据库资源能力都利用上，这样整体 DataX 的迁移效率才能达到最好。

此外，如果主机内存够大，`datax.py` 能使用的 JVM 内存也可以调大。您可编辑脚本，调大 `-Xms` 和 `-Xmx` 参数。

```
vim bin/datax.py

30 DEFAULT_JVM = "-Xms16g -Xmx16g -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=%s/log" % (DATAX_HO
ME)
```

读写并行度

DataX 的配置文件中首先有 `speed` 的设置，其中 `channel` 是总的并发数。OceanBase 的 `oceanbasev10writer` 还进一步支持多线程，通过参数 `"writerThreadCount":10` 指定每个 channel 从源端读取的数据，再分几个并发线程写入。所以 OceanBase 数据库总的写入并发数是 `channel` * `writerThreadCount`。

每个 `writer` 里还有个 batchsize，这是一个事务的记录数数量。通常建议 200-10000 都可以，尽量不要超过 1 万。在 OceanBase 数据库里事务太大太长，可能会到达事务超时时间（默认 120 秒）。

这里有趣的是，OceanBase 的 `oceanbasev10writer` 会把 batch insert 合并为一个 insert 多个 values 子句。所以 `batchSize` 也不要太大，否则 insert sql 文本太长, 高并发时也可能会报错（内存不足方面的错误）。当列非常多的时候（比如 100 列）或者值的内容有大文本时，建议 `batchSize` 控制在几百左右。

源端数据库读优化

当源端是数据库时，如果表有单列主键，并且主键列类型为数值型（如 number、bigint、integer、decimal

等），可以在源端 `reader` 里增加配置 `"splitPk": "id"`。这个时候，DataX 能先对主键进行切片，然后多个 channel 同时并发分段去读取源数据。如果没有该配置，源端就只能单并发读取数据。

OceanBase 写入的内存调优

OceanBase 数据库的数据读写模型比较特殊，增量都在内存里。当 OceanBase 机器已经是 SSD 盘时，IO 不会先成为 OceanBase 数据库的性能瓶颈，内存和 CPU 更有可能先成为瓶颈。大量数据写入时，增量对 memtable 内存的消耗会很快。OceanBase 数据库设置不当的情况下，可能会出现内存耗尽的情况，进而写入报错。其他业务写入也会跟着报错。

OceanBase 数据库的内存优化过程比较复杂。这里先给出一个初始的设置，能降低内存写入报错的概率。

```
alter system set merge_thread_count = 32;  #  增大合并的线程数。
alter system set minor_merge_concurrency = 16;  #  增大转储的线程数，期望提高转储的速度。
alter system set _mini_merge_concurrency = 8;  #  增大 mini_merge 的线程数，期望提高 mini_merge 的速度（默认值为 3）。调大为 8 以后，发现会导致压测中 CPU 使用率有时飙升至 90%，对性能有影响。
alter system set memory_limit_percentage = 90;    #  OceanBase 占系统总内存的比例，提高 OceanBase 可用的内存量。
alter system set memstore_limit_percentage = 55;  #  memstore 占租户的内存比，尽量增大 memstore 的空间（但是可能对读操作有负面影响）。
alter system set freeze_trigger_percentage = 40;  #  启动 major/minor freeze 的时机，让转储（minor freeze）尽早启动，memstore 内存尽早释放。
alter system set minor_freeze_times = 100;        #  minor freeze 的次数，尽量不在测试期间触发 major freeze。
alter system set minor_warm_up_duration_time = 0;  #  加快 minor freeze
```

OceanBase 数据库的 `oceanbasev10writer` 插件也提供参数（`memstoreThreshold`）监测增量内存的利用率，如果到达这个阈值，DataX 会自动降速。

OceanBase 数据库的增量内存使用也可以监控，关键 SQL 如下：

```
SELECT tenant_id, ip, round(active/1024/1024/1024) active_gb, round(total/1024/1024/1024) total_gb, round(freeze_trigger/1024/1024/1024) freeze_trg_gb, round(mem_limit/1024/1024/1024) mem_limit_gb
    , freeze_cnt , round((active/freeze_trigger),2) freeze_pct, round(total/mem_limit, 2) mem_usage
FROM `gv$memstore`
WHERE tenant_id =1001
ORDER BY tenant_id, ip;
```

OceanBase 数据库的监控产品也能监控增量内存的变化。

4.13 如何使用 CloudCanal 迁移和实时同步数据到 OceanBase

介绍

CloudCanal 社区版是一款由 ClouGence公司 发行的集结构迁移、数据全量迁移/校验/订正、增量实时同步为一体的免费数据迁移同步平台。产品包含完整的产品化能力，助力企业打破数据孤岛、完成数据互融互通，从而更好的使用数据。

CloudCanal 提供了完整的产品化能力，用户在可视化界面完成数据源添加和任务创建即可自动完成结构迁移、全量迁移、增量实时同步。本文将说明如何将 MySQL 数据库中的数据迁移同步到对端 OceanBase 中，其他源端例如 Oracle 或 PostgreSQL 同步到 OceanBase 也可以按照类似的方式进行。

下载安装

CloudCanal 最新版下载地址：https://www.clougence.com/

CloudCanal社区版安装部署参考文档：https://doc-cloudcanal.clougence.com/operation/install_linux

功能说明

- 当前支持的链路以及核心功能项：

数据源\功能项	结构迁移	全量数据迁移	增量实时同步
Oracle -> OceanBase	支持	支持	支持
PostgreSQL -> OceanBase	支持	支持	支持

Greenplum -> OceanBase	支持	支持	不支持
MySQL -> OceanBase	支持	支持	支持

- 产品能力矩阵

功能域	功能项	功能说明
用户体系	注册/登陆	作为平台使用，支持用户注册独立租户、资源隔离；允许账号密码登陆、手机验证码登陆
	密码找回	支持密码忘记找回
数据源管理	添加阿里云数据源	用户配置有权限的ak/sk信息后，通过可视化界面可以轻松导入CloudCanal支持的阿里云数据源
	添加自建数据源	支持自建数据源添加
	数据源筛选	提供强大的筛选能力方便用户更好的检索数据源。支持按照数据源类型、描述、部署类型等多维度搜索
	数据源信息可视化展示	可视化分页展示数据源明细，包含数据库类型、部署类型、版本号等多维度信息
	数据源信息修改	支持修改内外网地址、同步使用的账号、关联的阿里云AK/SK信息等
集群管理	集群筛选	支持集群按照名称、类型进行多为筛选
	集群列表可视化展示	支持列表展示集群信息，包括集群内可用机器数量、创建人等信息
	集群新增、删除	支持添加集群、删除集群
机器管理	机器列表可视化展示	可视化展示机器多维信息，包括机器核心监控指标、描述信息等
	机器、任务绑定关系查看	支持查看每台机器上绑定的任务
	机器日志白屏化查看	支持直接在控制台上查看机器白屏化日志
	任务手动调度	支持手动调度任务到其他机器执行
	机器运维	支持机器离线、上线、删除等生命周期管理；支持机器存活性检测、报警开关
	添加阿里云机器白名单	支持一键添加机器白名单到任务关联的阿里云数据库
监控管理	任务大盘	支持查看全局任务统计信息和实时监控信息
	机器大盘	支持查看全局集群、机器统计信息

	异常监控	支持全局异常白屏化展示，可以实时查看管控、机器、任务的异常信息便于自助定位问题
	告警日志	告警发送日志，监测报警发送情况
异步任务管理	异步任务可视化展示	诸如任务创建、任务重跑、添加白名单等能力都为多阶段的异步任务，此处可以展示其运行情况；明细页也可以看到其具体的执行的多个子阶段
	异步任务运维	支持失败的异步任务子阶段进行重试
操作审计	可视化展示审计记录	所有操作均按照安全等级划分了审计日志
数据迁移与同步内核能力	结构迁移	支持异构数据源之间迁移表结构，支持类型自动映射、类型降级等能力，方便用户轻松迁移表结构
	全量数据迁移	按照用户订阅的配置信息，将订阅的库表数据迁移到对端数据源
	增量数据迁移	通过订阅解析数据源的变更日志，实时地将源端的INSERT/UPDATE/DELETE同步到对端数据源
	增量期间DDL同步	支持源端数据库DDL改写，同步到对端
	自动化流程	可以自动化完成结构迁移、全量、增量，无需人工介入，并且保证数据一致性
	两种并发模型	支持表级并行、PK HASH两种并发模型
任务创建	可视化创建步骤	通过五个步骤即可完成结构迁移、全量迁移、增量同步一体化的任务
	树形订阅/默认订阅模式	树形订阅模式支持直接将库表展开进行批量处理与映射；默认模式支持手动处理库表列映射细节
	整库迁移	支持流程简化的整库迁移
	任务类型选择	支持结构迁移、全量、增量、数据校验多种组合类型
	任务规格	支持选择不同的任务规格，精细化控制任务资源消耗
	禁止自动启动	可以关闭自动启动，手动启动任务
	库、表、列映射	如果映射对端已有的库表，支持库表列维度的对象映射
	行为过滤	支持细粒度订阅INSERT/UPDATE/DELETE/ALTER/RENAME等行为
	虚拟列	支持同步的时候新增一个虚拟列，一般用于数据汇聚场景
	自定义主键	允许用户设置自定义主键，一般用于数据汇聚场景
	where条件	允许用户设置过滤的where条件，按照where条件过滤数据
	批量操作	支持where条件、自定义主键、列裁剪的批量操作

	多维筛选	表映射时可以按照有无主键、主键是否对齐等多种维度进行筛选，细粒度操作映射
	自定义代码	创建任务时可以上传自定义代码，在全量、增量期间根据自定义代码做自定义的数据实时加工
	创建预检	任务创建时候支持必要前提的预检，避免创建不符合要求的同步任务
任务管理	任务可视化展示	支持表格、卡片两种分页展示模式；可视化呈现多维信息，并且能够按照状态、同步进度进行排序
	任务筛选	可以按照任务描述、源对端数据源ID等多维条件进行任务筛选
	任务生命周期管理	可以可视化查看任务生命周期和阶段；支持任务启停控制、删除
	订阅明细查看	可以查看库表订阅详情、支持订阅库表搜索
	进度明细查看	可以查看各个阶段表的迁移、同步进度，并且支持搜索和维度筛选
	重启历史记录	支持查看任务的调度历史、重启历史查看
	任务告警配置	支持钉钉、短信两种报警模式；支持异常报警、延迟报警；支持报警自动发送给管理员
	白屏化日志查看	支持在管控界面直接查看任务核心日志
	自定义代码包管理	支持上传、下载、激活自定义代码包
	创建相似任务	支持按照当前任务的配置创建一个相似任务
	参数修改	支持任务参数控制
	监控图表	支持性能、资源等指标的监控
	位点回溯/重置	允许重置增量消费的位点，对已经消费过的增量重新进行消费
	规格升降配	支持对任务规格进行升降配
	源对端限流	支持源对端进行限流
	任务重跑	支持重跑全量
日志能力	COMMIT日志	提交日志记录所有CloudCanal收到并且正确同步的日志
	位点日志	详细记录位点提交的日志信息
场景化能力	ZeroDate处理	MySQL源端有0000这种特殊时间，支持设置DefaultZeroDate自动处理零时间
	异常跳过	允许用户自动跳过异常记录，避免阻塞同步。跳过的异常记录信息会记录的日志中
	MySQL RDS OSS binlog支持	支持阿里云RDS for MySQL的OSS binlog， CloudCanal如果找不到MySQL本地日志，会自动下载OSS binlog消费

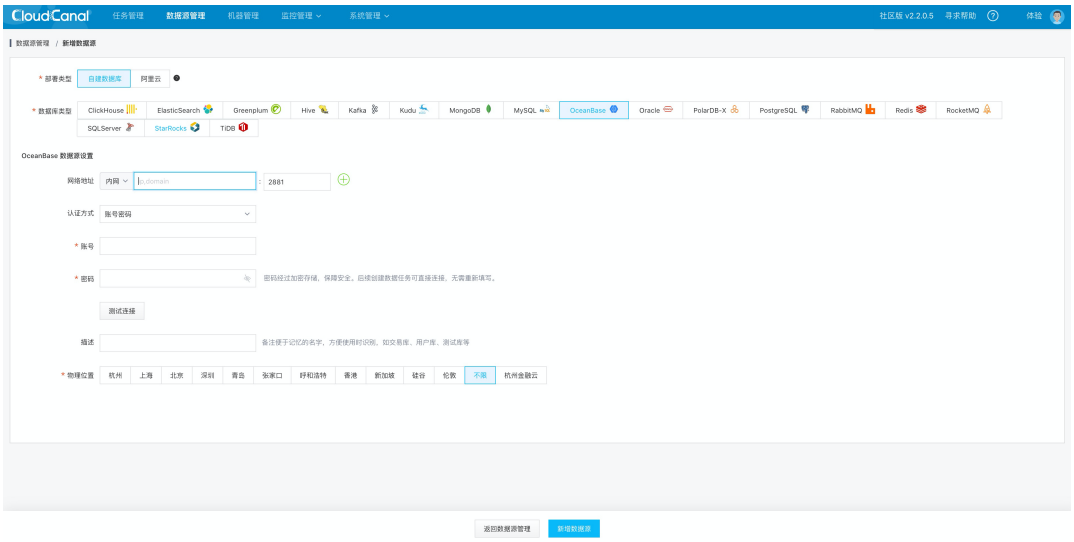
	双向同步	MySQL->MySQL链路支持双向同步
	数据校验	允许比对源对端差异、丢失的数据
	数据订正	支持根据数据校验的结果进行数据订正
	时区处理	部分链路支持时区自动转换
	DEBUG日志	支持开启debug模式，允许查看同步的记录明细
	UK冲突感知的写入模式	UK列冲突的情况，并行写入可能产生死锁，该写入模式可避免死锁
	json特殊字符转义	针对json内容，可以对特殊字符转义，避免写入异常
	任务摘除	增量任务会预占用内存，通过任务摘除可以解除内存预占用
	心跳机制	源端实例没有任何变更时会导致延迟增加，支持自动化心跳
高可用	自动迁移	一台机器下线后，上面的任务可以自动迁移到别的正常的机器
	自动重试	默认情况不允许跳过数据，如果因为对端数据库问题或者网络问题导致写入失败，数据会自动尝试写入
数据准确性	位点机制、断点续传	只有成功写入对端的记录才会对位点做持久化，相当于记录一个checkpoint，后续任务重启都会基于上一次的位点进行重新同步，确保数据不丢

前置条件

首先参考 CloudCanal安装部署参考文档 完成 CloudCanal 社区版的安装和部署。

添加数据源

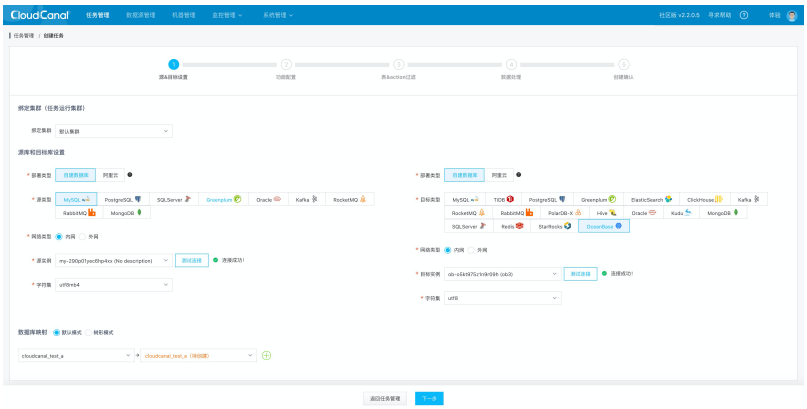
- 登录 CloudCanal 平台
- 数据源管理 -> 新增数据源
- 选择自建数据库中 OceanBase



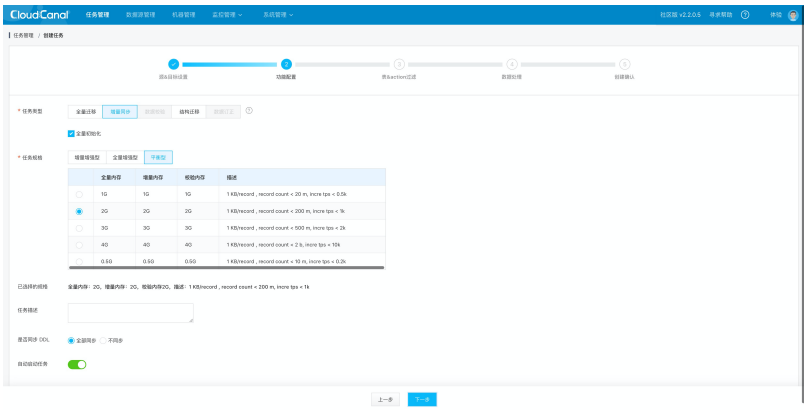
任务创建

添加好数据源之后可以按照如下步骤进行数据迁移、同步任务的创建。

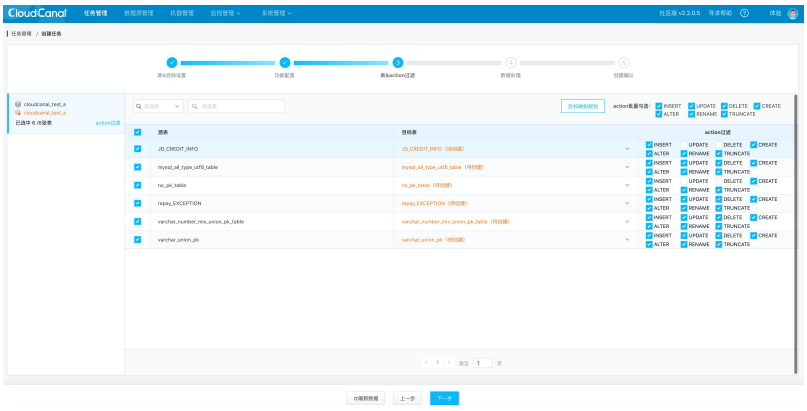
- **任务管理->任务创建**
- 选择 **源** 和 **目标** 数据库
- 点击 下一步



- 选择 **增量同步**，并且启用 **全量数据初始化**
- 点击下一步



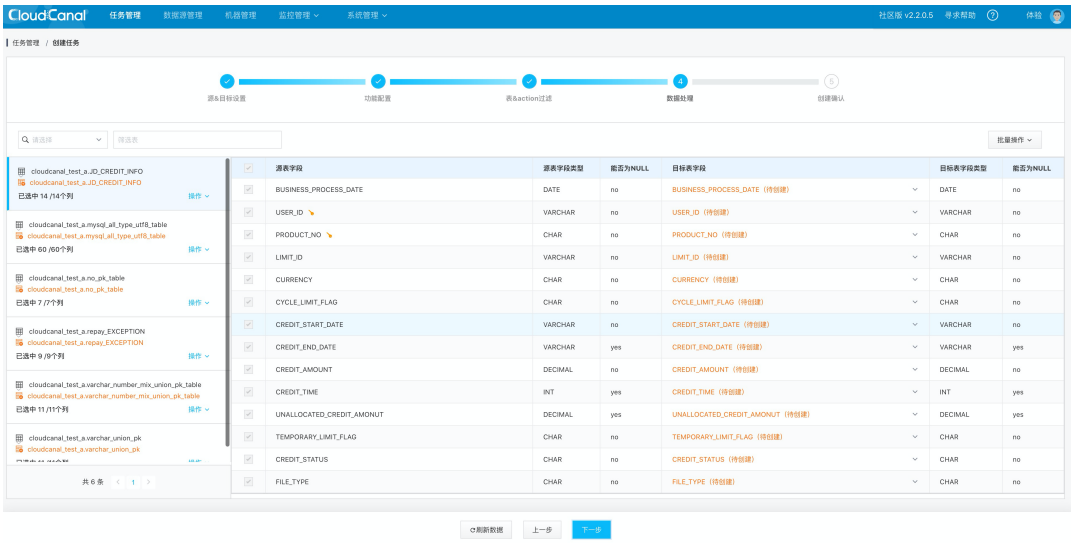
- 选择订阅的表，结构迁移自动创建的表会按照默认类型映射进行处理。对端表如果已经提前建好，这里也可以直接映射对端已经存在的表
- 点击下一步



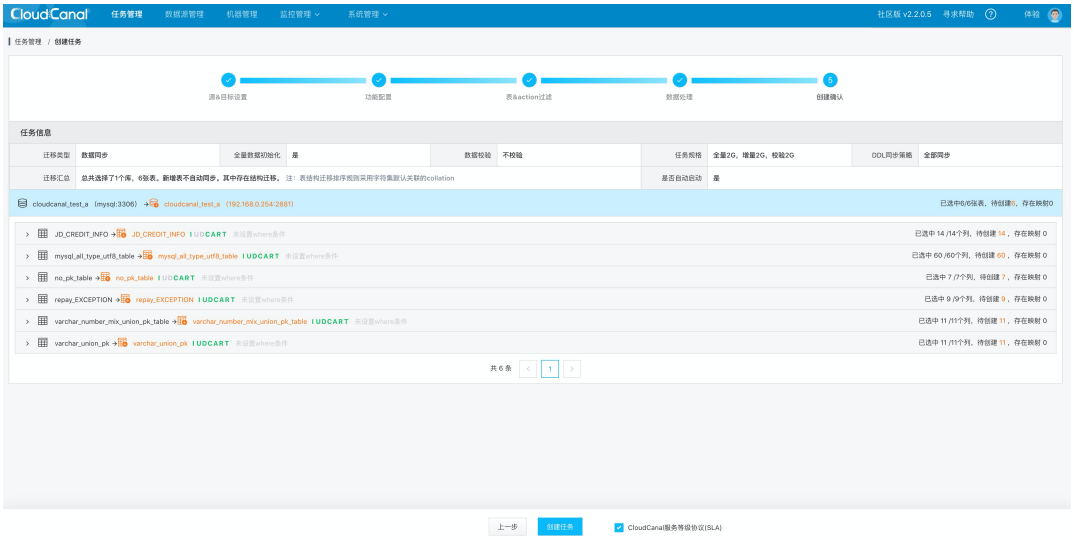
- 配置列映射
- 点击下一步

说明

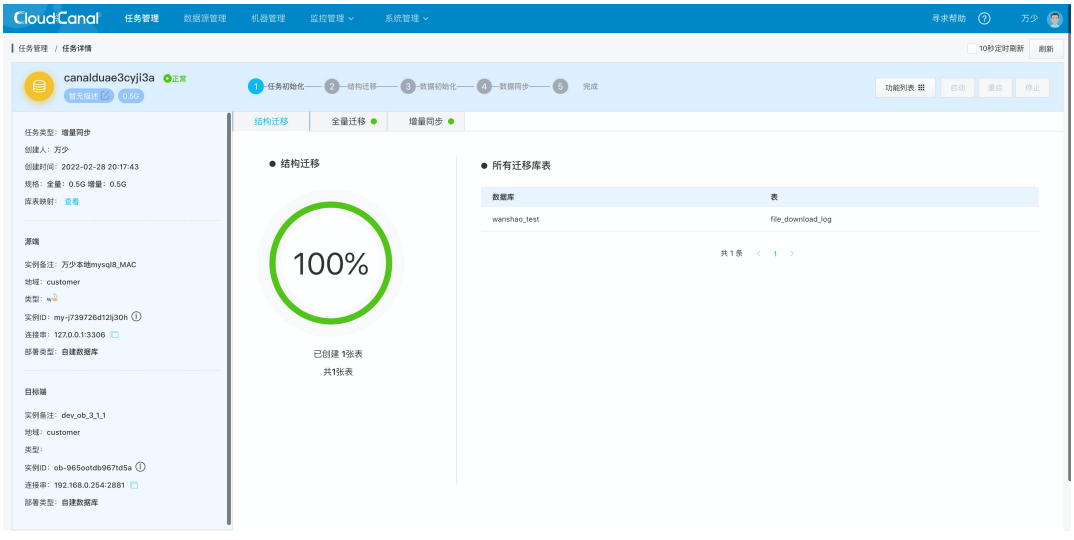
如果是通过 CloudCanal 结构迁移自动建表，这边不允许重命名、裁剪以及列映射；如果映射的是对端已经提前建好的表，这边支持列的裁剪和映射。



- 创建任务



- 查看任务状态。任务创建后，会自动完成结构迁移、全量、增量阶段。



参考资料

更多关于CloudCanal同步OceanBase的资料，可以查看如下资料：

- 5分钟搞定 MySQL/PostgreSQL/Oracle 到 OceanBase 数据迁移同步-CloudCanal 实战：https://www.askcug.com/topic/328
- CloudCanal 官方社区：https://www.askcug.com/

如在使用过程中，遇到问题请与 CloudCanal 官方社区联系。

第 5 章：运维 OceanBase 数据库

OceanBase 运维旨在维护集群稳定、管理集群资源和保证集群性能最优。OceanBase 运维工作包含:

- 集群扩容/缩容/替换/搬迁/重启/备份/监控/升级
- 租户扩容/缩容/性能调优
- 故障处理

OceanBase 自动化运维工具 OBD 能够方便集群部署，也提供部分扩容和升级能力，但 OBD 不是运维平台。OceanBase 的官方运维产品 OCP 后期会支持社区版。OCP 有产品文档手册，这里就不再重复。

本章主要介绍 OceanBase 运维操作的具体原理和步骤。自动化产品 OBD 只是将这些基本操作做了封装。第三方运维平台也可以参考这里的原理和步骤进行开发，将 OceanBase 纳入自己的产品支持列表中。

本章目录

5.1 如何管理 OceanBase 集群

OceanBase 集群架构简述

进程简述

OceanBase 数据库是单进程软件，进程名为：`observer`。通常每个机器启动一个进程就是一个节点。在第 2 章部署中介绍了 OceanBase 数据库的目录架构。以下是 OceanBase 数据库和 OBProxy 安装后的目录。

```
[admin@obce02 ~]$ pwd
/home/admin
[admin@obce02 ~]$ tree -L 3 --filelimit 10 oceanbase/ obproxy/ /redo /data/
oceanbase/
├── bin
│   ├── import_time_zone_info.py
│   └── observer
├── etc
│   ├── observer.config.bin
│   ├── observer.config.bin.history
│   └── timezone_V1.log
├── lib
│   ├── libaio.so -> libaio.so.1.0.1
│   ├── libaio.so.1 -> libaio.so.1.0.1
│   ├── libaio.so.1.0.1
│   ├── libmariadb.so -> libmariadb.so.3
│   └── libmariadb.so.3
├── log [588 entries exceeds filelimit, not opening dir]
├── run
│   ├── mysql.sock
│   └── observer.pid
└── store
    └── obdemo
        ├── clog -> /redo/obdemo/clog
        ├── etc2 -> /redo/obdemo/etc2
        ├── etc3 -> /data/obdemo/etc3
        ├── ilog -> /redo/obdemo/ilog
        ├── slog -> /redo/obdemo/slog
        └── sstable -> /data/obdemo/sstable
obproxy/
/redo
└── obdemo
    ├── clog [160 entries exceeds filelimit, not opening dir]
    ├── etc2
    ├── ilog [17 entries exceeds filelimit, not opening dir]
    └── slog
        ├── 5
        └── 6
/data/
└── obdemo
    ├── etc3
    └── sstable
```

```
        └── block_file

21 directories, 15 files
```

OceanBase 数据库目录中需要关注以下几个目录:

- 软件安装目录

  若您是使用 RPM 包手动安装 OceanBase 数据库的，默认会安装在用户 admin 的 HOME 目录下（`~/`）。如果您是使用 OBD 自动部署的，RPM 将会解压缩到用户的 HOME 目录下隐藏文件夹里（`~/.obd`）。

- 工作目录

  本教程里进程 observer 的工作目录是 `~/oceanbase`，进程 obproxy 的启动目录是 `~/obproxy`。

  进程 observer 和 obproxy 启动时都会在工作目录下寻找 `etc` 子目录，读取默认配置文件。如果目录不存在则创建一个文件夹 `etc`，同时还会创建日志目录 `log` 等。所以需要了解进程 observer 和 obproxy 的启动目录，以便进程故障时，能够快速拉起进程。

  此外，启动用户也需要保持一致。本教程使用的是用户 admin。使用 OBD 部署 OceanBase 数据库和 OBProxy 时，可以指定工作目录。

- 参数目录

  参数文件目录默认在工作目录下的 `etc` 目录中，此外还可以通过参数 `config_additional_dir` 设定冗余的参数目录。参数文件是二进制文件，不能直接编辑和读取，您可通过 `strings` 命令读取。

- 日志目录

  日志目录默认放在工作目录下的 `log` 目录。如果工作目录下无 `log` 目录，将会自动生成。日志目录里有三类日志：`observer.log`、`rootservice.log` 和 `election.log`。每个日志又有一个完整的目录和一个日志级别在 WARN 级别及以上的日志。后者可以通过参数关闭。

  `observer.log` 通过参数设置可以滚动生成，保留指定日志数量。当数据库访问量非常大时，进程 observer 的日志输出量也会比较大。所以这个目录在生产环境中建议可用空间不少于 100G。并且在集群的参数里还会对日志输出限流。

- 总数据目录

  总数居目录默认放在工作目录下的 `store` 文件夹。当然进程 observer 也可以通过参数 `-d` 指定特定目录。

  本教程里手动部署 OceanBase 数据库的时候，建议在 `store` 目录下设置集群名作为总数据目录，然后通过参数 `-d` 指定到具体目录（如 `-d ~/oceanbase/store/obdemo`）。该目录下会有实际的数据文件目录（`sstable`）和事务日志目录（`slog`、`ilog`、`clog`）。

  通常建议数据文件目录和事务日志目录用独立的盘存放，所以该目录下的数据文件目录和事务日志目录是一个软链接。

- 数据文件目录

数据文件目录默认文件夹名为 `sstable`，目录下只有一个大文件 `block_file`。在进程 observer 启动时初始化，初始化默认大小是磁盘可用空间的 `80%`。`block_file` 的大小可以通过参数控制。

数据文件目录不能和事务日志目录共用一块盘存储。如果节点要重置，该目录下的内容必须清理。

- 事务日志目录

  事务日志目录包含三个文件夹:

  - `slog`：存储静态数据写入的事务日志。
  - `clog`：存储动态数据写入的事务日志。
  - `ilog`：存储日志目录。

  这些日志 OceanBase 数据库会自动管理，不要手动删除。这三个目录通常建议独立一块盘存储。如果节点要重置，这三个目录下的内容必须清理。

进程监听端口

下面分别介绍进程 observer 和 obproxy 的监听端口。

进程 observer 第一次启动时，需要通过参数 `-p` 指定连接端口，通过参数 `-P` 指定节点 RPC 通信端口。默认是 `-p 2881 -P 2882`。进程启动成功后，几秒就可以运行到监听环节。

查看方法如下:

```
[admin@obce02 oceanbase]$ pidof observer
56779

[admin@obce02 oceanbase]$ ps -ef | grep observer | grep -v grep
admin      56779       1 99 16:26 ?        00:00:49 bin/observer

[admin@obce02 oceanbase]$
[admin@obce02 oceanbase]$ netstat -ntlp | grep -i observer
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:2881            0.0.0.0:*               LISTEN      56779/bin/observer
tcp        0      0 0.0.0.0:2882            0.0.0.0:*               LISTEN      56779/bin/observer
[admin@obce02 oceanbase]$
```

最后一列中数字是进程 ID。 排查问题时，建议查看节点之间的 TCP 连接状态，或者查看哪个 IP 和 PORT 流量最大。

```
# 查看 TCP 连接
netstat -np |grep -i observer

# 查看 TCP 连接汇总
netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'

# 查看网卡 IP 流量
iftop -i eth0 -nNB
```

```
# 按 L 、T 、3 、t 、 B 、l 、 p  找出具体哪个 IP 和 PORT 流量最大
```

集群节点组织架构

OceanBase 集群由多个节点的进程组合而成，集群最小规模是 1 个节点，这是一个单副本集群，没有高可用能力。

在生产环境中，OceanBase 集群最小规模是三节点。会分为三个 `Zone`，每个 `Zone` 最少一个节点。实际上集群里节点组织关系如下:

集群	ZONE	REGION	IDC	IP
obdemo	ZONE1	region1	idc1	192.168.249.52
obdemo	ZONE2	region1	idc1	192.168.249.49
obdemo	ZONE3	region1	idc2	192.168.249.51
obdemo	ZONE4	region1	idc2	192.168.249.54
obdemo	ZONE5	region2	idc3	192.168.249.55

节点的 Zone 在进程启动的时候指定。节点的 region 和 idc 是和 Zone 绑定的，可以在集群初始化完成后查看和修改。 下面只是示例，并不表示一定需要修改。

```
alter system change zone 'zone1' region 'region1';
alter system change zone 'zone2' region 'region1';
alter system change zone 'zone3' region 'region2';

alter system modify zone zone1 idc='idc1';
alter system modify zone zone2 idc='idc2';
alter system modify zone zone3 idc='idc3';

select * from __all_zone where name in ('region','idc','status','zone_type','cluster') order by
+----------------------------+----------------------------+-------+-----------+-------+---------
| gmt_create                 | gmt_modified               | zone  | name      | value | info
+----------------------------+----------------------------+-------+-----------+-------+---------
| 2021-09-12 14:49:43.533839 | 2021-09-12 14:49:43.533839 |       | cluster   |     0 | obdemo
| 2021-09-12 14:49:43.536334 | 2021-09-20 17:08:13.727095 | zone1 | idc       |     0 | idc1
| 2021-09-12 14:49:43.537407 | 2021-09-20 17:08:20.052725 | zone2 | idc       |     0 | idc2
| 2021-09-12 14:49:43.538453 | 2021-09-20 17:08:25.589423 | zone3 | idc       |     0 | idc3
| 2021-09-12 14:49:43.536334 | 2021-09-20 17:07:44.025486 | zone1 | region    |     0 | region1
| 2021-09-12 14:49:43.537407 | 2021-09-20 17:07:48.804019 | zone2 | region    |     0 | region1
| 2021-09-12 14:49:43.538453 | 2021-09-20 17:07:57.702095 | zone3 | region    |     0 | region2
| 2021-09-12 14:49:43.535302 | 2021-09-14 10:25:55.748709 | zone1 | status    |     2 | ACTIVE
| 2021-09-12 14:49:43.536334 | 2021-09-12 14:49:43.536334 | zone2 | status    |     2 | ACTIVE
| 2021-09-12 14:49:43.537407 | 2021-09-12 14:49:43.537407 | zone3 | status    |     2 | ACTIVE
| 2021-09-12 14:49:43.536334 | 2021-09-12 14:49:43.536334 | zone1 | zone_type |     0 | ReadWrit
| 2021-09-12 14:49:43.537407 | 2021-09-12 14:49:43.537407 | zone2 | zone_type |     0 | ReadWrit
| 2021-09-12 14:49:43.538453 | 2021-09-12 14:49:43.538453 | zone3 | zone_type |     0 | ReadWrit
+----------------------------+----------------------------+-------+-----------+-------+---------
13 rows in set (0.003 sec)
```

当集群中的所有节点都在同一机房时，region 和 idc 设置为同一个值即可。只有在三机房部署时，才需要对 region 和 idc 进行设置，以方便集群正确管理各个节点。比如说避免把一个机房的节点错划到另外一个机房的 Zone 中。

这些元数据靠 DBA 维护，集群部署时，节点只需连通性、时间延时和网络延时满足要求即可，并不要求机器一定在同一个机房。

在后期对集群做读写分离规划时，region 和 idc 对 OBProxy 的 SQL 路由策略有很大影响。所以，请您尽量正确设置。

如何管理 OceanBase 集群参数

OceanBase 集群支持参数定制，以适应不同的场景需求。参数的查看和修改主要在集群内部租户 sys 中进行。

参数简介

部分参数的生效范围是集群内所有实例，可以通过命令控制在哪些节点生效。少数参数是在租户范围内生效，大部分参数的变更是立即生效，极少数参数的变更需要重启节点方可生效。

参数的值、描述、生效范围、是否要重启节点等，都可以通过命令 `show parameters [ like '%参数名特征%' ;` 查看。 您也可以通过以下命令对参数进行修改。

```
alter system set 参数名 = 参数值（如果是字符串，用两个单引号引起来）[ scope = 'xxx.xxx.xxx.xxx:2882' ] ;
```

除了集群参数可以定制集群行为外，租户也可以定制集群行为，可通过变量（variable）进行设置。

变量跟参数是同一类概念，参数的定位和风格取自 Oracle 数据库，变量的定位和风格取自 MySQL 数据库。集群 sys 租户中同样可以设置变量值（sys 租户变量通常不用调整），不同租户的变量设置彼此独立，互不影响。

大部分变量的设置是立即生效，生效范围根据设置方法不同有实例全局层面生效和会话层面生效两种情况。极少数变量（类似 MySQL 的初始化变量）不能在租户中后期修改，只能在 sys 租户中新建业务租户的时候设置。当然也能后期在 sys 租户进行修改，但是必须充分评估修改的影响。

变量的值、描述都可以通过以下命令查看或修改。

```
show global | [session] variables [ like '%变量名特征%' ] ;
# 或
show global | [session] variables where variable_name in ('参数1' [, '参数2']) ;
# 您可以通过下列命令修改变量的值
set global | [session] 变量名 = 变量值（如果是字符串，用两个单引号引起来）;
```

其中，`global` 指全局生效，对新建会话生效，当前会话不生效。 `session` 或者没有限定符，都表示对当前会话生效。会话断开重连后，又会取全局层面的默认值。

部分变量还可以通过 SQL HINT 设置语句级别变量，影响范围是所修饰的语句。语句级的变量名会去掉前面的 `ob_` 部分。

下面是常用的几个语句级别的变量命名映射关系。

全局/会话变量名	语句级变量名	描述
ob_query_timeout	query_timeout	指定语句执行超时的阈值
ob_read_consistency	read_consistency	指定读一致性级别，默认是强一致性读，可以修改为弱一致性读

常用集群参数作用的分类如下:

- 影响节点可用性的判断。如 `server_permanent_offline_time` 和 `server_temporary_offline_time`。
- 影响集群日志的特点。如 `enable_syslog_recycle`、`enable_syslog_wf`、`max_syslog_file_count`、`syslog_level` 和 `syslog_io_bandwidth_limit`。
- 影响集群冻结和合并行为。如 `minor_freeze_times`、`major_freeze_duty_time`、`enable_major_freeze`、`freeze_trigger_percentage`、`micro_block_merge_verify_level`、`minor_merge_concurrency`、`merge_thread_count`、`zone_merge_timeout`、`enable_merge_by_turn`、`zone_merge_order`、`zone_merge_concurrency` 和 `enable_manual_merge`。
- 影响集群节点内存分配的行为。如 `memstore_limit_percentage`、`memory_limit_percentage`、`sql_audit_memory_limit`、`system_memory`、`memory_limit`。
- 影响集群负载均衡行为。如 `server_balance_cpu_mem_tolerance_percent`、`server_balance_disk_tolerance_percent`、`balancer_tolerance_percentage`、`data_copy_concurrency`、`server_data_copy_out_concurrency`、`server_data_copy_in_concurrency`。

以上参数的作用可以在命令里查看，也可以查询官方文档。之后讲解运维、性能调优时会用到这些参数，届时将详细介绍。

参数文件

除了通过登录集群 sys 实例查看集群的参数外，您还可以通过参数文件查看。

OceanBase 集群参数文件名称为 `observer.config.bin`，位于进程工作目录下的 `etc/` 目录下。如果您是使用 OBD 部署的 OceanBase 集群，配置文件里的 `home_path` 就是集群的工作目录。

您也可以通过查看进程的运行环境确定进程的工作目录。

```
[admin@obce02 oceanbase]$ sudo ls -lrth /proc/`pidof observer`/cwd
lrwxrwxrwx 1 admin admin 0 Sep 21 13:10 /proc/56779/cwd -> /home/admin/oceanbase
```

进程目录下的 `cwd` 为指向实际工作目录的软链接。集群的参数文件是二进制文件，不能直接编辑和查看，您可使用

命令 `strings` 进行查看。

```
[admin@obce02 oceanbase]$ strings /home/admin/oceanbase/etc/observer.config.bin | grep config_addition
config_additional_dir=/data/obdemo/etc3;/redo/obdemo/etc2
```

通常集群的参数修改成功后都会持久化到参数文件内。

进程启动参数

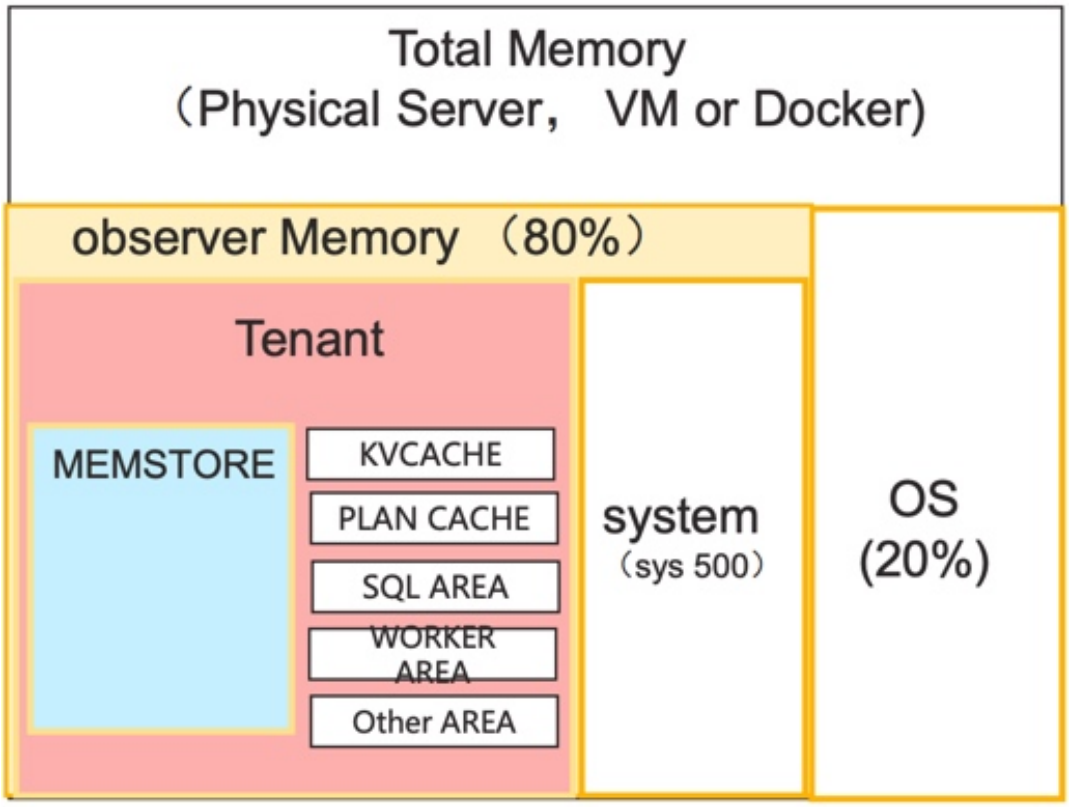
如果参数文件设置不合理，可能会导致进程启动失败。但进程启动参数文件不能直接修改，此时您可在启动进程 `observer` 的时候指定启动参数。如：

```
cd /home/admin/oceanbase && bin/observer -o "memory_limit=50G, system_memory=10G"
```

这种方法适合调整少数参数，让进程启动起来。启动成功后，该参数也会持久化到进程的参数文件里。

如何管理 OceanBase 数据库内存

OceanBase 数据库的内存分配可以参考内存分配图。



下面对图中提到的内容逐个进行详细介绍。

OceanBase observer 进程启动后会从主机拿到绝大部分资源（CPU/内存/空间），具体资源大小则由集群参数控制。其中集群节点进程启动需要多少内存是由参数 `memory_limit` 和 `memory_limit_percentage` 共同控制的。

生产环境中通常不设置 `memory_limit` 值（默认值为 `0`），只设置 `memory_limit_percentage` 值（默认值为 `80%`）。这样每个节点进程获取的总内存是主机可用内存的 80% 。当主机内存扩容后（例如虚拟机），OceanBase 节点能获得的内存也相应增加。

如果设置了 `memory_limit` 的值，则进程获取的总内存就是指定的具体的值，该值的最小值为 8G。该参数通常在测试环境中运行 OceanBase 数据库使用，或者单机启动多个 OceanBase 进程。

> **注意**
>
> 这两个参数可以在线调整，但调整参数时最大不能超过主机可用内存，最小不能少于 8G。通常调大是安全的，调小可能有风险。

系统内部内存

OceanBase 数据库内部租户除了 sys 租户外，还有 500 租户（租户 ID 为 500），500 租户的内存由 system_memory 配置项确定。

不同版本的 OceanBase 数据库中该参数默认值不同。生产环境中，主机有 386GB 内存以上时，这个参数默认值是 50GB。如果主机内存只有 256GB，这个参数内存会小一些，比如 30GB 左右。

在第 2 章中为了在 10GB 内存的机器上运行 OceanBase 数据库，`system_memory` 参数的取值范围为 3GB ~ 5GB。

该内存在内部租户 ID 500 的租户下，供业务租户某些内部操作使用。

sys 租户内存

sys 租户在集群初始化时自动创建。随着可用内存大小、版本的变化，sys 租户的资源规格会有一些细微差别。sys 租户的内存大小是由默认资源单元 `sys_unit_config` 的规格定义的。

```
select unit_config_id, name, round(max_memory/1024/1024) max_mem_MB, round(min_memory/1024/1024) min_m
em_MB from __all_unit_config where unit_config_id = 1;
+----------------+-----------------+------------+------------+
| unit_config_id | name            | max_mem_MB | min_mem_MB |
+----------------+-----------------+------------+------------+
|              1 | sys_unit_config |       1536 |       1024 |
+----------------+-----------------+------------+------------+
1 row in set (0.009 sec)
```

示例中 `sys_unit_config` 默认的内存规格最小值和最大值不一样，为了更准确的计算集群可用资源，建议把内存最小值和最大值更新为同一个值（即只能变大，不能小于默认最大值）。后续创建业务租户时，建议 CPU 和内存的最小值和最大值保持一致。

```
alter resource unit sys_unit_config min_memory = '1536M';
```

如果集群节点规模在 10 台以内，sys 租户资源通常不需要调大。

如果集群规模有几十台，需观察 sys 租户的内存和 CPU 利用率。如果出现资源瓶颈，则适当调整 sys 租户资源规

格，最大时可以独占一台机器资源。sys 租户最多只能独占 3 台机器。

MemStore 内存

每个租户分得的内存，默认有一半用于存放租户的增量数据，也叫 MemStore。该内存大小通过参数 `memstore_limit_percentage` 控制，默认值是 `50%`。

当写多读少时，可以将这个参数值调高，比如在大批量导数据且没有读的时候，可以临时将参数值调整到 `80%`。当然该参数不能随意调整。

租户的内存除去增量数据占用的内存外，还有其他用处，比如存放静态的数据（`KVCache`）、SQL 执行计划缓存（`PLAN CACHE`）、SQL 运算的临时内存（`SQLAREA`）等。如果这些内存资源过少，则会报错，或者性能减慢。

> **说明**
>
> 目前 OceanBase 数据库对内存的管理以手动分配为主，将来或许可以做到动态管理。

OceanBase 数据库的读写模型是 LSM-Tree，写操作是追加模式。

数据表是有序表格，又叫 SSTABLE，全称为 SORTED STRING TABLE。数据表的写操作不是在原来的数据块上修改，而是开辟新的内存记录变化的增量，原有的数据保存在 KVCache 不变，称之为基线数据。

绝大部分业务一天的数据变化量相对于数据总量来说比重很小，OceanBase 数据库的这种增量写很节省内存。所以，通常说 OceanBase 数据库的增量数据都是在内存里不落盘的，但严格来说，这种说法并不严谨，OceanBase 数据库的增量数据在内存里应该是延迟落盘。

如何对 OceanBase 集群进行合并

OceanBase 数据库每天会定时将增量数据和基线数据在内存中合并成新的数据块，然后以追加方式写回到磁盘数据文件中。这个合并操作叫 `major freeze`，合并的时间调度通过参数 `major_freeze_duty_time` 设定，默认值是 `02:00`，每天调度一次。

合并操作也可以手动发起。合并时会消耗一定资源，会对性能产生一些影响。不过合并的并发也可以通过参数 `merge_thread_count` 控制，并且通过参数 `merger_warm_up_duration_time` 调整合并线程启动时间。调整这两个参数可以在合并的速度和稳定性之间取一个最佳平衡。

当租户内存不是很大，或者业务写量非常大的时候，该租户的增量内存可能会写满，会出现业务写不进去，报错 `NO MEMORY` 的情况。这种情况是 OceanBase 运维要极力避免的。

您可运行 `alter system major freeze;` 命令合并集群，之后通过以下命令查看当前的合并进度。

```
SELECT ZONE,svr_ip,major_version,ss_store_count,merged_ss_store_count,modified_ss_store_count,merge_st
art_time,merge_finish_time,merge_process
FROM __all_virtual_partition_sstable_image_info
order by major_version desc ;
```

最终合并结果确认时查看合并状态。

```
select * from __all_zone where name in ('merge_status','all_merged_version','broadcast_version');
```

正常情况都是 `IDLE`，且合并版本和广播版本保持一致。

如何对 OceanBase 集群进行限速和转储

首先，OceanBase 数据库租户可以设置一个增量内存写限速的设计，触发限速有以下两种情况:

- 增量内存利用率到达参数 `writing_throttling_trigger_percentage` 设置。
- 预估剩余增量内存持续可用时间小于或等于参数 `writing_throttling_maximum_duration` 定义。

默认参数 `writing_throttling_trigger_percentage` 的值是 `100%`，也就是不限速。实际业务建议最大不超过 `90%`。

注意

当写入量非常大时，参数 `writing_throttling_trigger_percentage` 的值需要适当调低。

当业务写被限速的时候，业务表现为普通的 DML SQL 性能变慢（延时变大），等待事件是 `mem allocation` 等。限速很影响性能，所以该情况需要极力避免。

OceanBase 数据库除了 `major freeze` 操作可以释放内存外，`minor freeze` 操作也可以释放内存，该操作通常被称为转储。

转储是直接将内存中的增量数据以 SSTABLE 格式写到磁盘数据文件中，从而释放增量内存。转储与合并相比，对资源的占用更少，对性能的影响也很小。

转储的触发有两个机制。

- 手动触发

  手动触发的命令是 `alter system minor freeze ;`。命令发生时内存中的增量块会被冻结禁止写入，新的写入会生成新的增量内存块去接受写入。

- 自动触发

  当增量内存部分的利用率达到参数 `freeze_trigger_percentage` 指定的值（默认是 `70`，表示 `70%`），自动触发 `minor freeze` 操作。

转储可以有很多轮，当转储的次数达到参数 `minor_freeze_times` 指定的值，会自动触发合并（major freeze）操作。

转储也支持调优，通过调整转储的并发参数（`minor_merge_concurrency`），可以在转储的速度和稳定性之间取得一个平衡。

经验表明，当租户内存规模在 50GB 以上时，通过调整转储参数，绝大部分时候是可以避免增量内存写入限速或内存不足的错误。如果业务增量写非常大，则说明租户内存有瓶颈，则需要对内存资源扩容。

注意

如果是临时大批量导入数据，可以临时对租户内存进行扩容。等导入完毕，一轮合并结束后，就可以对租户内存进行缩容。在线扩容和缩容，也是 OceanBase 数据库最核心和实用的能力。

如何重启 OceanBase 节点和集群

(建议) OceanBase 集群合并

OceanBase 集群生产环境默认是有三个副本，任意一个副本不可用，OceanBase 数据库的内部都会发生一次故障切换。此时部分数据访问会中断，恢复时间在 30s 左右，数据能保证绝对不丢。这个故障切换过程不需要运维介入，可靠性很高。

如果是计划中的节点重启，OceanBase 集群内部也会自动发起切换，此时大部分读写感知不到切换，少数大事务会中断需要重试。

所以，整体上 OceanBase 集群计划重启可以做到业务不停机，数据库服务不中断。您只需要按机器所属的 ZONE 分批次重启 OceanBase 机器。

由于 OceanBase 节点重启后会有一个恢复的过程，恢复的时间取决于从最近的基线版本到当前时间需要恢复的数据量。OceanBase 数据库可能要应用很多 `clog` 去恢复数据。如果需恢复的 `clog` 数据量实在太大，OceanBase 节点可能会直接从主副本那里拉取最新的数据。

如果条件允许，在重启节点之前，可先对整个集群发起一次合并。待合并结束后，每个节点内存中就只有最近一段时间的增量数据，那么节点重启后的恢复时间可做到最短。

发起合并操作的命令为 `alter system major freeze;`。合并操作详情，参考：如何对 OceanBase 集群进行合并。

如何重启 OceanBase 节点

1. 节点停服

   OceanBase 数据库并没有提供重启节点的命令，但是提供了节点停服的命令:

   ```
   alter system stop server '节点ip:2882' ;
   ```

   节点停服后，节点上如果有主副本，会自动切换为备副本。节点的备副本依然参与投票，但不会当选为主副本。

   OceanBase 节点停服和 OceanBase 宕机性质不同，节点停服时间可以超出参数永久下线时间（`server_permanent_offline_time`) 而不会导致节点真的下线。

   节点停服后，大概 1~2 秒后就可以观察到有主备副本切换事件。 确认 SQL 如下:

   ```
   SELECT DATE_FORMAT(gmt_create, '%b%d %H:%i:%s') gmt_create_ , module, event, name1, value1, name
   2, value2, rs_svr_ip,name3,value3,name4,value4
   FROM __all_rootservice_event_history
   ```

```
WHERE 1 = 1
   AND module IN ('leader_coordinator', 'balancer' )
ORDER BY gmt_create DESC
LIMIT 20;
```

所有分区副本的主备切换通常需要数秒。

2. “杀”进程

“杀”节点进程 `observer` 命令如下:

```
kill `pidof observer`
```

如果是在测试环境中，比较着急的情况下可以不用发起节点停服命令，直接强行"杀"进程。

```
kill -9 `pidof observer`
```

3. 启动进程

启动节点进程的关键点在于上一次工作目录启动。所以建议第一次启动时就把工作目录跟安装目录保持一致，防止出错。

```
cd /home/admin/oceanbase && bin/observer
```

通常启动的时候不需要再带启动参数。但如果这次启动节点就是为了修改某个参数，则需要使用 `-o` 带上具体的参数。示例如下:

```
cd /home/admin/oceanbase && bin/observer -o "datafile_size=80G,clog_disk_usage_limit_percentage=
96 "
```

进程启动后等 5~10 秒后，确认进程是否启动成功。

```
#  确认进程还在
ps -ef | grep observer | grep -v grep

#  确认端口监听成功

netstat -ntlp | grep `pidof observer`
```

4. 启动服务并确认节点服务状态

进程监听成功还不够，节点还需要一个数据恢复过程，即应用 CLOG 的过程。

如果此前对节点停服了，先把节点服务启动。

```
alter system start server '节点IP:2882' ;
```

确认节点的服务状态。

```
select a.zone,concat(a.svr_ip,':',a.svr_port) observer, cpu_total, (cpu_total-cpu_assigned) cpu_
free, round(mem_total/1024/1024/1024) mem_total_gb, round((mem_total-mem_assigned)/1024/1024/102
4) mem_free_gb, usec_to_time(b.last_offline_time) last_offline_time, usec_to_time(b.start_servic
e_time) start_service_time, b.status, usec_to_time(b.stop_time) stop_time, b.build_version
from __all_virtual_server_stat a join __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_
port)
order by a.zone, a.svr_ip;
```

您需重点关注的内容如下:

- 节点状态 `status`：升级前没有 `inactive` 值，升级过程中会有。
- 节点服务时间 `start_service_time` 是否是默认值（`1970-01-01 08:00:00.000000`）。如果是，则表示节点还没有恢复结束。
- 节点停止时间 `stop_time` 是否是默认值（`1970-01-01 08:00:00.000000`）。如果不是，则表示节点被停服（`stop server`）了，需要先启动服务（`start server`）。

注意

节点的 `start_service_time` 状态正常之后方可重启其他 Zone 对应的其他副本所在的机器。同一个 Zone 的多台机器可以并行重启。

如何重置 OceanBase 节点

重置 OceanBase 节点属于应急手段之一，是高危操作，仅用在如下特殊的场景:

- 节点磁盘损坏后修复，怀疑数据有丢失的。此时需要重做副本。
- 想对节点数据文件大小进行缩容，重新初始化进程 `observer`，相当于重做副本。

重置 OceanBase 节点的步骤如下。

停掉节点进程

因为是要重置节点，所以停进程的方法就不用像重启节点那么繁琐，可以直接杀进程。

```
kill -9 `pidof observer`
```

如果进程已经停止，则跳过这步操作。 杀进程之前，需要先知道此前的进程启动时工作目录。使用如下命令:

```
ll /proc/`pidof observer`/cwd
lrwxrwxrwx 1 admin admin 0 Sep 25 08:18 /proc/15495/cwd -> /home/admin/oceanbase-ce
```

确认节点永久下线

通常情况下，节点进程异常，相当于节点掉线。掉线时间超过节点掉线时间超过参数

`server_temporary_offline_time` 值（默认 `60s`）后，状态会进入临时下线状态。此后，如果节点进程能重新正常起来，节点还是集群的成员，会自动同步落后的数据。

示例:

```
select a.zone,concat(a.svr_ip,':',a.svr_port) observer, usec_to_time(b.last_offline_time) last_offline
_time, usec_to_time(b.start_service_time) start_service_time, b.status, usec_to_time(b.stop_time) stop
_time
from __all_virtual_server_stat a join __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_port)
order by a.zone, a.svr_ip;
```

输出:

```
+-------+-------------------+----------------------------+----------------------------+----------+---
-------------------------+
| zone  | observer          | last_offline_time          | start_service_time         | status   | st
op_time                  |
+-------+-------------------+----------------------------+----------------------------+----------+---
-------------------------+
| zone1 | 172.20.249.52:2882 | 1970-01-01 08:00:00.000000 | 2021-09-25 08:19:06.622351 | active   | 19
70-01-01 08:00:00.000000 |
| zone2 | 172.20.249.49:2882 | 1970-01-01 08:00:00.000000 | 2021-09-25 08:19:07.392669 | active   | 19
70-01-01 08:00:00.000000 |
| zone3 | 172.20.249.51:2882 | 1970-01-01 08:00:00.000000 | 1970-01-01 08:00:00.000000 | inactive | 19
70-01-01 08:00:00.000000 |
+-------+-------------------+----------------------------+----------------------------+----------+---
-------------------------+
3 rows in set (0.008 sec)
```

节点掉线后进入临时下线状态时，上面节点视图的 `status` 列会变为 `inactive`。同时，在 OceanBase 事件日志视图里也会有一条“临时下线”记录。

```
SELECT DATE_FORMAT(gmt_create, '%b%d %H:%i:%s') gmt_create_ , module, event, name1, value1, name2, val
ue2, rs_svr_ip
FROM __all_rootservice_event_history
WHERE 1 = 1
   AND module IN ('server','root_service')
   and gmt_create > SUBDATE(now(),interval  1800 second)
ORDER BY gmt_create DESC
LIMIT 10;
```

输出:

```
+----------------+--------------+-----------------+--------+----------------------+-------+----------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------+--------------
+
| gmt_create_    | module       | event           | name1  | value1               | name2 | value
2                                                                                                    
                                                                                              | rs_svr_i
p     |
+----------------+--------------+-----------------+--------+----------------------+-------+----------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------+--------------
```

```
+
| Sep26 12:39:32 | root_service | admin_set_config | ret    | 0                    | arg   | {items:[{n
ame:"rootservice_list", value:"172.20.249.52:2882:2881;172.20.249.49:2882:2881", comment:"", zone:"",
server:"0.0.0.0", tenant_name:"", exec_tenant_id:1, tenant_ids:[]}], is_inner:false} | 172.20.249.52 |
| Sep26 12:39:32 | server       | lease_expire     | server | "172.20.249.51:2882" |
|
                                                                                                  | 172.
20.249.52 |
+----------------+--------------+------------------+--------+----------------------+-------+----------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------+---------------
+
2 rows in set (0.001 sec)
```

节点掉线首先会有个 `lease_expire` 事件。节点掉线原因可以有很多，如进程宕掉、网络超时或延时过大、时间误差过大等。 此外，由于这个示例集群是三节点，所以一个节点的掉线对总控服务成员也有影响，所以参数 `rootservice_list` 会自动变化，踢掉了故障节点。

如果节点掉线时间超过参数 `server_permanent_offline_time` 值（默认是 3600s ），节点会进入永久下线状态。此时，集群会清空该节点上的数据副本，并自动在同 Zone 其他节点寻求资源补足被清空的数据副本。如果没有可用资源，则这个副本对应的分区就只剩下两个副本（或者四个副本）。此时依然是多数派副本可用，所以数据读写是正常的，但如果再次宕机集群可能不可用。

```
SELECT DATE_FORMAT(gmt_create, '%b%d %H:%i:%s') gmt_create_ , module, event, name1, value1, name2, val
ue2, rs_svr_ip FROM __all_rootservice_event_history WHERE 1 = 1       AND module IN ('server','root_se
rvice')    and gmt_create > SUBDATE(now(),interval  7200 second) ORDER BY gmt_create DESC LIMIT 10;
```

输出:

```
+----------------+--------------+----------------------+--------+----------------------+-------+------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------+-----------
----+
| gmt_create_    | module       | event                | name1  | value1               | name2 | value
2
                                                                                     | rs_svr_i
p     |
+----------------+--------------+----------------------+--------+----------------------+-------+------
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------+-----------
----+
| Sep26 13:39:48 | server       | clear_with_partition | server | "172.20.249.51:2882" |
|
                                                                                                  | 172.
20.249.52 |
| Sep26 13:39:22 | server       | permanent_offline    | server | "172.20.249.51:2882" |
|
                                                                                                  | 172.
20.249.52 |
| Sep26 12:39:32 | root_service | admin_set_config     | ret    | 0                    | arg   | {items
:[{name:"rootservice_list", value:"172.20.249.52:2882:2881;172.20.249.49:2882:2881", comment:"", zone:"
", server:"0.0.0.0", tenant_name:"", exec_tenant_id:1, tenant_ids:[]}], is_inner:false} | 172.20.249.5
2 |
| Sep26 12:39:32 | server       | lease_expire         | server | "172.20.249.51:2882" |
|
                                                                                                  | 172.
```

```
20.249.52 |
+---------------+-------------+---------------------+--------+---------------------+-------+------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+-----------
----+
4 rows in set (0.004 sec)
```

从上面可以看到，节点被永久下线时，会发生事件 `permanent_offline`，节点的数据也会随后被清空，产生事件 `clear_with_partition`。

从临时下线到永久下线时间可能有点长，默认 1 小时。如果您时间紧张，可以临时把这个节点的参数 `server_permanent_offline_time` 调小。等节点永久下线后再重新上线时，把参数再改回来。

```
alter system set server_permanent_offline_time='360s';
```

（可选）清理数据库相关文件

如果您希望重建副本，则这一步并不是必须的。如果您想借重建副本操作重新定义数据文件的大小，则可以操作此步骤。

清理的文件包括:

- 运行日志。包括 `log` 目录下的 `observer.log`、`rootservice.log`、`election.log`。
- 数据文件。包括 `sstable` 下的 `block_file` 文件。
- 事务日志文件。包括目录 {`ilog`、`clog`、`slog` } 下的文件。

注意

为方便再次启动节点进程，不要删除参数文件 `etc` 。在启动参数 `-o` 里指定需要修改的参数即可。

下面删除只是示例，请注意目录结构:

```
[admin@obce03 store]$ cd /home/admin/oceanbase-ce/store
[admin@obce03 store]$ ls
clog  ilog  slog  sstable
[admin@obce03 store]$ pwd
/home/admin/oceanbase-ce/store
[admin@obce03 store]$ /bin/rm -rf */*
```

启动进程

如果没有清理参数文件，则可以直接启动进程 `observer`，可以通过 `-o` 修改参数。

```
cd /home/admin/oceanbase-ce && bin/observer

# 或
```

```
cd /home/admin/oceanbase-ce && bin/observer  -o "datafile_size=100G"
```

启动后确认端口监听正常，节点状态正常，这个前面已经阐述，此处不再重复了。

如何为 OceanBase 集群扩容/缩容/替换机器

OceanBase 集群扩容或缩容主要是调整集群机器资源池。

扩容就是往集群里加机器，操作相对简单，复杂的是缩容。

缩容的前提是剩余的机器能容纳所有租户的资源需求。如果资源不足，通常就要先对集群里的租户进行缩容，所以集群的缩容步骤将留到介绍租户缩容时再补充。

集群机器替换则是先扩容后缩容，集群机器资源池并没有发生变化，所以操作也很简单。

集群扩容节点

OceanBase 集群的部署分为标准部署模式和非标准部署模式。

- 标准部署模式是有三个 ZONE，每个 ZONE 的机器数量相等，每个 ZONE 里可以存在不同配置的机器，但是不同 ZONE 之间的机器配置是对等的，机器资源总量也是对等的。
- 非标准部署模式则是 ZONE 之间的机器配置不对等、数量也不对等。技术上这也是可以运行的，只是资源最少的机器或 ZONE 会成为集群资源的瓶颈。

OceanBase 集群扩容的标准形式就是向每个 ZONE 里添加同等配置的机器。非标准做法就是只向一个 ZONE 里添加机器。

操作步骤如下:

1. 环境初始化

   初始化新机器环境的具体操作步骤，请参见 2.4 如何初始化服务器环境。

2. 启动进程

   observer 进程第一次启动时需要指定一些参数，具体操作步骤请参见 2.11 （高级）如何手动部署 OceanBase 集群。

   ```
   cd ~/oceanbase && bin/observer -i eth0 -p 2881 -P 2882 -z zone1 -d ~/oceanbase/store/obdemo -r '
   172.20.249.52:2882:2881;172.20.249.49:2882:2881;172.20.249.51:2882:2881' -c 20210912 -n obdemo -
   o "memory_limit=8G,cache_wash_threshold=1G,__min_full_resource_pool_memory=268435456,system_memo
   ry=3G,memory_chunk_cache_size=128M,cpu_count=16,net_thread_count=4,datafile_size=50G,stack_size=
   1536K,config_additional_dir=/data/obdemo/etc3;/redo/obdemo/etc2" -d ~/oceanbase/store/obdemo
   ```

   参数说明:

   - `-p`: 连接端口。默认是 2881，可以根据实际情况修改。

- `-P`：RPC 端口，默认是 2882，可以根据实际情况修改。
- `-r`：指定 rootservice list，跟集群当前的 rootservice list 保持一致。
- `-z`：指定节点所属 ZONE。
- `-c`：cluster_id。
- `-n`：指定节点所属的集群名。

3. 添加节点

   在集群 sys 租户中运行以下命令添加新节点。

```
ALTER SYSTEM ADD SERVER '节点IP:RPC端口' ZONE '节点所属ZONE';
# 例如:
alter system add server '11.166.87.5:2882' zone 'zone1';
```

4. 查看状态

   添加节点之后查看节点状态，此时新节点的状态是 `active`。

```
select a.zone,concat(a.svr_ip,':',a.svr_port) observer, cpu_total, (cpu_total-cpu_assigned) cpu_
free, round(mem_total/1024/1024/1024) mem_total_gb, round((mem_total-mem_assigned)/1024/1024/102
4) mem_free_gb, usec_to_time(b.last_offline_time) last_offline_time, usec_to_time(b.start_servic
e_time) start_service_time, b.status, usec_to_time(b.stop_time) stop_time
from __all_virtual_server_stat a join __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_
port)
order by a.zone, a.svr_ip;
```

以上步骤全部完成之后，需按相同步骤向下一个 ZONE 中继续新增机器。当三个 ZONE 都扩容完毕，集群扩容操作就完成了。

不过，这里只代表着运维的工作结束。集群扩容后，新的节点不会马上被集群里的租户使用到。集群资源池扩大后，新节点的利用率是 0，约 1 分钟后，集群会发现资源利用不均衡，之后会启动负载均衡逻辑，尝试将其他租户的资源单元迁移到新的节点上。

集群缩容也会触发负载均衡。有关负载均衡的概念后文会介绍到。

为集群中的每个 ZONE 添加节点后，集群整体的资源池能力新增，可用的资源也相对增加，但是租户的资源并没有增加，租户不一定能利用上新机器，所以之后还需要进行租户扩容。

集群缩容节点

OceanBase 集群的标准部署模式是有三个 ZONE，每个 ZONE 的机器数量都对等。集群缩容节点就是减小某个 ZONE 的节点数。通常缩容节点时建议每个 ZONE 都减少相同的节点，但实际操作中没有这个限制，都是按 ZONE 操作。

集群缩容节点时，会触发租户资源的负载均衡。剩余的节点必须能容纳租户资源需求，否则这个节点缩容命令会报错。所以，集群节点缩容之前，需要计算集群可用资源和了解各个节点的资源池分布。

集群缩容节点命令就是将某个节点从某个 ZONE 里删除，删除时需确保节点没有资源被分配出去。

示例 SQL:

```
alter system delete server '11.166.87.5:2882' zone 'zone1';
```

集群缩容节点会触发租户资源负载均衡，之后会触发数据迁移。建议关注数据迁移的进度和影响。查看集群事件日志视图。

```
SELECT DATE_FORMAT(gmt_create, '%b%d %H:%i:%s') gmt_create_ , module, event, name1, value1, name2, val
ue2, rs_svr_ip
FROM __all_rootservice_event_history
WHERE 1 = 1
   AND gmt_create > SUBDATE(now(),interval  1 hour)
ORDER BY gmt_create DESC
LIMIT 20;
```

如何给 OceanBase 集群扩容/缩容副本

三副本扩容到五副本

生产环境 OceanBase 集群通常有三个 ZONE，分布在一个机房或者同城三个机房，数据也有三份（简称三副本）。当建设异地容灾机房时，可能会选择从三个 ZONE 扩容到 五个 ZONE（即五副本）。另外一种就是机房搬迁，将集群从一个机房在线搬迁到另外一个机房。这其中都涉及到三副本扩容到五副本。

下面介绍如何从三副本扩容到五副本。

1. 机器节点进程初始化

   主要是对 ZONE4 和 ZONE5 的机器进行初始化，注意启动参数中所属的 ZONE 要正确。

   具体步骤请参见：2.4 如何初始化服务器环境 和 2.11 （高级）如何手动部署 OceanBase 集群。

2. 新增 ZONE

   ```
   alter system add zone 'zone4';
   alter system add zone 'zone5';
   ```

3. 启动 ZONE

   ```
   alter system start zone zone4;
   alter system start zone zone5;
   ```

4. 给新增的 ZONE 添加节点

   ```
   alter system add server '11.166.87.5:2882' zone 'zone4';
   alter system add server '11.166.87.6:2882' zone 'zone5';
   ```

5. 为租户在新增节点上分配资源

```
create resource pool test_pool_z4 unit='unit_4c16g' , unit_num=1, zone_list=('ZONE_4');
create resource pool test_pool_z5 unit='unit_4c16g' , unit_num=1, zone_list=('ZONE_5');
```

该命令立即返回。

6. 修改租户的 LOCALITY ，给租户增加新增的资源池

注意

每次只能给租户新增一个资源池。

```
alter tenant test227 resource_pool_list=('test_pool_z1','test_pool_z2','test_pool_z3','test_pool
_z4');

alter tenant test227 resource_pool_list=('test_pool_z1','test_pool_z2','test_pool_z3','test_pool
_z4','test_pool_z5');
```

租户在新增 ZONE 中补数据副本是异步进行的，租户分区数越多，数据量越大，该命令的运行时间越长。

7. (可选）确认复制表的副本是否扩容

复制表的 LOCALITY 可以指定在哪些 ZONE 里配置复制表副本（也可以不指定），默认是整个集群范围内。如果原本指定 LOCALITY，这里需要手动添加新的 ZONE（这个待验证）。

```
alter table BMSQL_ITEM locality='F,R{all_server}@ZONE1, F,R{all_server}@ZONE2, F,R{all_server}@ZON
E3, F,R{all_server}@ZONE4,F,R{all_server}@ZONE_5';
```

8. 确认集群和租户副本扩容结果

```
select a.zone,concat(a.svr_ip,':',a.svr_port) observer, cpu_total, (cpu_total-cpu_assigned) cpu_
from __all_virtual_server_stat a join __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_
order by a.zone, a.svr_ip;

select t1.name resource_pool_name, t2.`name` unit_config_name, t2.max_cpu, t2.min_cpu, t2.max_me
from __all_resource_pool t1 join __all_unit_config t2 on (t1.unit_config_id=t2.unit_config_id)
    join __all_unit t3 on (t1.`resource_pool_id` = t3.`resource_pool_id`)
    left join __all_tenant t4 on (t1.tenant_id=t4.tenant_id)
order by t1.`resource_pool_id`, t2.`unit_config_id`, t3.unit_id;

select * from __all_tenant;
```

五副本缩容到三副本

五副本缩容到三副本时，将上面步骤反过来执行即可。

1. （可选）调整复制表的 LOCALITY ，缩减副本数。

2. 修改租户的 LOCALITY，缩减副本数。

```
alter tenant sys resource_pool_list = ('sys_pool','sys_pool_1');
alter tenant sys resource_pool_list = ('sys_pool');
alter tenant test227 resource_pool_list=('test_pool_z1','test_pool_z2','test_pool_z3','test_pool
_z4');
alter tenant test227 resource_pool_list=('test_pool_z1','test_pool_z2','test_pool_z3');
```

注意，sys 租户也需要缩减副本数。这一步命令立即返回，但是删除副本会异步进行，速度也很快，分区数可以多一些，基本上在分钟级别完成。

3. 删除多余的资源池

```
drop resource pool sys_pool_1 ;
drop resource pool sys_pool_2 ;
drop resource pool test_pool_z4 ;
drop resource pool test_pool_z5 ;
```

如何升级 OceanBase 集群

OceanBase 集群升级就是将集群所有节点进程 observe 的版本升级。通常小版本的升级直接替换可执行文件 `observer` 然后将进程重启即可。但有时候升级还会涉及到元数据的变更，这就需要参考新版本发布时具体的 RELEASE NOTE。

确认集群当前状态正常

OceanBase 集群可以在线不停服升级，具体就是按 ZONE 滚动升级。升级前首先要确认集群和各个 ZONE 状态都正常。避免停止某个 ZONE 时出现多数派故障，数据库访问不可用问题等。

- 确保集群节点状态正常

```
select a.zone,concat(a.svr_ip,':',a.svr_port) observer, cpu_total, (cpu_total-cpu_assigned) cpu_
free, round(mem_total/1024/1024/1024) mem_total_gb, round((mem_total-mem_assigned)/1024/1024/102
4) mem_free_gb, usec_to_time(b.last_offline_time) last_offline_time, usec_to_time(b.start_servic
e_time) start_service_time, b.status, usec_to_time(b.stop_time) stop_time, b.build_version
from __all_virtual_server_stat a join __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_
port)
order by a.zone, a.svr_ip;
```

您需关注的有如下几点:

- 节点状态 `status`：升级前没有 `inactive` 值，升级过程中会有。
- 节点服务时间 `start_service_time`：时间是否为默认值（`1970-01-01 08:00:00.000000`），如果是，则表示节点异常。
- 节点停止时间 `stop_time`：时间是否为默认值（`1970-01-01 08:00:00.000000`），如果不是，则表示节点

被 `stop server` 了。

- 节点版本 `b.build_version`：升级重启后会变为新版本。

- 观察集群最近一段时间的事件，确保无异常事件

```
SELECT DATE_FORMAT(gmt_create, '%b%d %H:%i:%s') gmt_create_ , module, event, name1, value1, name
2, value2, rs_svr_ip
FROM __all_rootservice_event_history
WHERE 1 = 1
   AND module IN ('server','root_service','daily_merge')
   and gmt_create > SUBDATE(now(),interval  1 day)
ORDER BY gmt_create DESC
LIMIT 100;
```

如果有大量正常的事件影响查看，就过滤掉。 留意报错的事件是否有影响，下列异常事件一定要先解决。如：合并异常、副本创建/搬迁异常等。

- （可选）发起合并（`MAJOR FREEZE`）

由于集群升级需要重启节点，为了减少节点重启后的恢复时间，建议升级之前发起一次合并。具体合并方法和确认方法跟 如何重启 OceanBase 节点 中方法相同。

```
alter system major freeze;
```

停止 ZONE 服务

- OceanBase 集群的升级，按 ZONE 滚动升级。首先选中一个 ZONE，将该 ZONE 停服。

```
alter system stop zone 'zone1';

SELECT * FROM __all_zone where name in ('status','merge_status') and zone = 'zone1';

# 输出:
+----------------------------+----------------------------+-------+--------------+-------+------
| gmt_create                 | gmt_modified               | zone  | name         | value | info
+----------------------------+----------------------------+-------+--------------+-------+------
| 2021-09-25 08:19:05.067944 | 2021-09-29 15:44:27.162171 | zone1 | merge_status |     0 | IDLE
| 2021-09-25 08:19:05.066915 | 2021-09-29 19:36:13.108063 | zone1 | status       |     1 | INACT
+----------------------------+----------------------------+-------+--------------+-------+------
2 rows in set (0.028 sec)
```

- 查看事件日志确认

```
SELECT DATE_FORMAT(gmt_create, '%b%d %H:%i:%s') gmt_create_ , module, event, name1, value1, name
2, value2, rs_svr_ip
FROM __all_rootservice_event_history
WHERE 1 = 1
   AND module IN ('server','root_service','leader_coordinator')
   AND gmt_create > SUBDATE(now(),interval  1 hour)
ORDER BY gmt_create DESC
```

```
LIMIT 20;
```

输出:

```
+----------------+-------------------+--------------------------+------------+-----------------
| gmt_create_    | module            | event                    | name1      | value1
+----------------+-------------------+--------------------------+------------+-----------------
| Sep29 19:36:40 | root_service      | full_rootservice         | result     | 0
| Sep29 19:36:39 | server            | load_servers             | ret        | 0
| Sep29 19:36:39 | server            | online                   | server     | "172.20.249.52:2
| Sep29 19:36:39 | server            | online                   | server     | "172.20.249.49:2
| Sep29 19:36:39 | server            | load_servers             | ret        | 0
| Sep29 19:36:39 | root_service      | finish_start_rootservice | result     | 0
| Sep29 19:36:39 | root_service      | start_rootservice        | self_addr  | "172.20.249.51:2
| Sep29 19:36:39 | root_service      | finish_wait_stop         | cost       | 350586
| Sep29 19:36:38 | root_service      | finish_stop_thread       | ret        | 0
| Sep29 19:36:38 | root_service      | stop_rootservice         | self_addr  | "172.20.249.52:2
| Sep29 19:36:38 | leader_coordinator | switch_leader           | current_rs | "172.20.249.52:2
| Sep29 19:36:22 | leader_coordinator | switch_leader           | current_rs | "172.20.249.52:2
| Sep29 19:36:13 | leader_coordinator | switch_leader           | current_rs | "172.20.249.52:2
| Sep29 19:36:13 | leader_coordinator | switch_leader           | current_rs | "172.20.249.52:2
| Sep29 19:36:13 | root_service      | stop_zone                | ret        | 0
+----------------+-------------------+--------------------------+------------+-----------------
15 rows in set (0.026 sec)
```

从事件日志看出，当停掉 ZONE 的时候，当总控服务（`rootservice`）的主副本也在该 ZONE 时，总控服务会发生切换（停掉老的服务，在新的节点上启动新的服务，其他 ZONE 所有节点重新上线），sys 租户和业务租户都发生切换。

- 确认节点状态

```
select a.zone,concat(a.svr_ip,':',a.svr_port) observer, usec_to_time(b.last_offline_time) last_o
ffline_time, usec_to_time(b.start_service_time) start_service_time, b.status, usec_to_time(b.sto
p_time) stop_time
from __all_virtual_server_stat a join __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_
port)
order by a.zone, a.svr_ip;
```

输出:

```
+-------+-------------------+----------------------------+----------------------------+--------
+----------------------------+
| zone  | observer          | last_offline_time          | start_service_time         | status
| stop_time                  |
+-------+-------------------+----------------------------+----------------------------+--------
+----------------------------+
| zone1 | 172.20.249.52:2882 | 1970-01-01 08:00:00.000000 | 2021-09-25 08:19:06.622351 | active
| 1970-01-01 08:00:00.000000 |
| zone2 | 172.20.249.49:2882 | 1970-01-01 08:00:00.000000 | 2021-09-25 08:19:07.392669 | active
| 1970-01-01 08:00:00.000000 |
| zone3 | 172.20.249.51:2882 | 1970-01-01 08:00:00.000000 | 2021-09-26 14:05:58.641570 | active
| 1970-01-01 08:00:00.000000 |
+-------+-------------------+----------------------------+----------------------------+--------
+----------------------------+
```

```
3 rows in set (0.017 sec)
```

更新节点的 OBSERVER 软件包

通常使用 RPM 包更新节点 OBSERVER 软件包。运行命令为 `rpm -uvh oceanbase-xxx.rpm`。此命令会自动覆盖可执行文件。但是由 LINUX 系统的设计可知，运行中的 observer 进程依然持有旧的文件句柄，所以不会释放文件。只有在 OceanBase 节点进程退出后才释放。

如果使用 OBD 管理 OceanBase 集群，软件包还需要维护到 OBD 本地仓库或远程仓库中。

5.2 如何管理 OceanBase 租户

OceanBase 租户原理

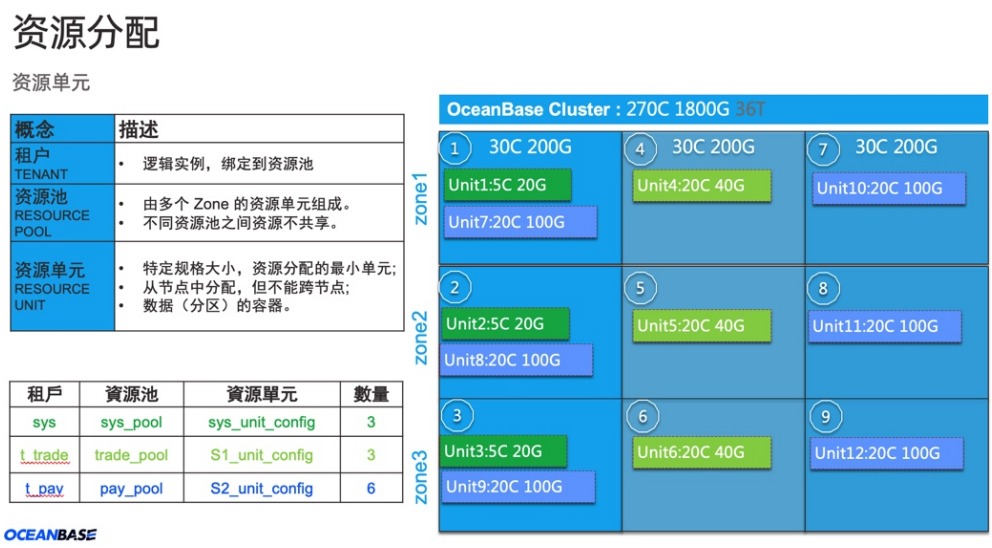
生产环境中 OceanBase 集群通常是三副本架构，有三个 ZONE，每个 ZONE 至少需要一台主机，集群架构是 N-N-N（N>=1）。

OceanBase 集群将所有节点的资源（主要是 CPU 、内存和磁盘空间）聚合成大的资源池之后，再二次分配给租户。

租户就是逻辑实例，每个租户对应一部分资源（主要是 CPU、内存）。租户的资源也被称为资源池，由每个 ZONE 中一个小的资源池组成，通常三个 ZONE 的资源池大小规格是一样的（可以不一样）。资源单元规格 * 资源单元数就是租户的可以利用的最大资源。

说明

每个 ZONE 中的资源池至少需包含一个资源单元。创建租户资源池时，每个资源单元只能从一个节点上分配资源。

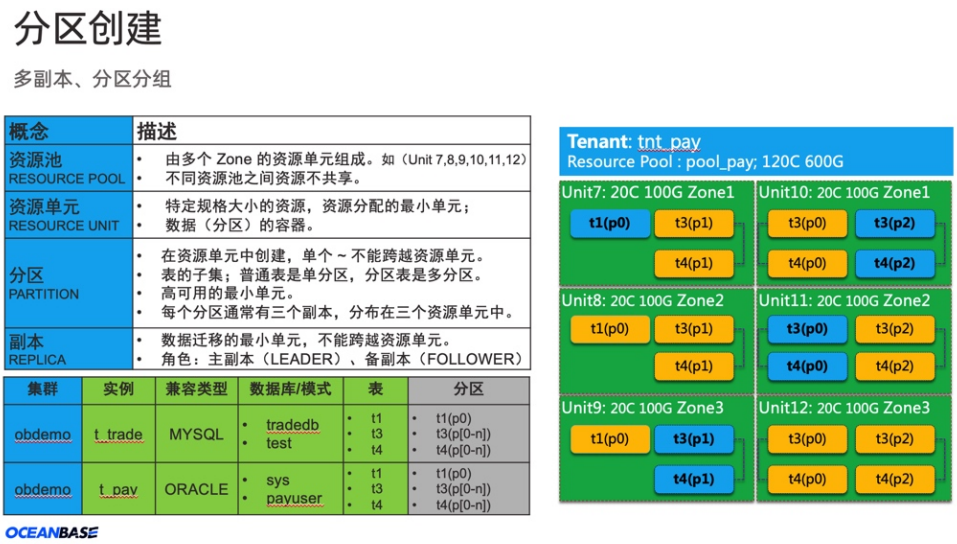


如上图:

- 集群的架构是 3-3-3。每个节点总资源是 30C 200G 内存，集群总资源是 270C 1800G 内存。
- 默认租户 sys 的架构是 1-1-1，每个 ZONE 里的资源单元大小是 5C 20G。
- 业务租户 t_trade 的架构是 1-1-1，每个 ZONE 里的资源单元大小是 20C 40G。
- 业务租户 t_pay 的架构是 2-2-2，每个 ZONE 里的资源单元大小是 20C 100G，每个 ZONE 有两个资源单元，分布在 2 个节点上。

每个租户的数据都从租户资源池对应的资源单元中分配。租户每个 ZONE 的资源单元数组成租户的架构：n-n-n（1<=n<=N）。

租户的数据管理单位是分区，分区是表的子集。普通表就是一个分区，分区表有多个分区。每个分区有三个副本，分布在三个 ZONE 里，具体就在租户的某三个资源单元中。



说明:

- 表 `t1` 是普通表，只有一个分区，标识为 `t1(p0)`，有三个副本，分别在资源单元 UNIT7、UNIT8、UNIT9 上。
- 表 `t3` 是分区表，有三个分区，标识为 `t3(p0)`、`t3(p1)` 和 `t3(p2)`，每个分区有三个副本，每个分区的三副本分别在三个资源单元上。
- 单看一个 ZONE，表 `t3` 的三个分区分散在两个节点上。如 `t3(p1)` 一个节点，`t3(p0)` 和 `t3(p2)` 是一个节点。

一个表有多个分区，这个技术类似存储的条带技术（Striping），每个分区有三个副本，类似存储的镜像技术（Mirroring）。

如何修改 OceanBase 租户变量/参数

集群的参数中有部分参数的生效范围是租户，这部分参数也称为租户的参数，会影响租户的行为。

常用租户参数如下:

参数名	参数值	参数含义
`writing_throttling_maximum_duration`	1h	增量内存写入限速剩余时间目标上限。
`writing_throttling_trigger_percentage`	100	增量内存写入限速触发阈值。建议设置为 90。

max_stale_time_for_weak_consistency	5s	弱一致性读能容忍最大延时。

除了参数，租户更常用变量（variable）来定制租户的行为。变量跟参数其实是同一类概念，参数的定位和风格取自 Oracle 数据库，变量的定位和风格取自 MySQL 数据库。

集群 sys 租户同样可以设置变量值（sys 租户变量通常不易调整，必须很谨慎），不同租户的变量设置彼此独立，互不影响。

大部分变量的设置是立即生效，生效范围根据设置方法不同可分为实例全局层面生效和会话层面生效。极少数变量（类似 MySQL 的初始化变量）不能在租户里后期修改，只能在 sys 租户里新建业务租户的时候设置。当然也可以后期在 sys 租户后期修改，只是在修改时必须充分评估影响。

变量的值、描述都可以通过以下命令查看。

```
show global | [session] variables [ like '%变量名特征%' ] ;
# 或
show global | [session] variables where variable_name in ('参数1' [, '参数2']) ;
# 您可通过以下命令修改变量的值
set global | [session] 变量名 = 变量值 ;  # 如果是字符串，用两个单引号引起来
```

其中，`global` 指全局生效，对新建会话生效，当前会话不生效。`session` 或者没有限定符，都表示对当前会话生效。会话断开重连后，又会取全局层面的默认值。

常用的租户变量如下:

变量名	变量默认值 (OceanBase)	变量含义
autocommit	ON	是否自动提交。 ON: 表示每个 DML 后会自动添加 commit 语句; OFF: 表示不会自动 commit，需要用户主动 commit 或 rollback 事务。
max_allowed_packet	4194304	需要调大，否则 SQL 文本或数据量很大的时候可能报错。
sql_mode	STRICT_ALL_TABLES	用于设置 SQL 模式。
transaction_isolation	READ-COMMITTED	用于设置事务的隔离级别。
tx_isolation	READ-COMMITTED	用于设置事务隔离级别。
lower_case_table_names	1	1（ON）表示自动将表名转小写。
ob_tcp_invited_node	%	% 表示允许所有 IP 访问。

s		
foreign_key_checks	ON	ON：检查外键约束； OFF：不检查外键约束。
ob_query_timeout	10000000	语句超时时间，单位是微秒（us）。
ob_trx_idle_timeout	120000000	事务空闲超时时间，单位是微秒（us）。
ob_trx_timeout	100000000	事务超时时间，单位是微秒（us）。
ob_trx_lock_timeout	-1	锁等待超时时间，默认是 `-1`，不超时。单位是微秒（us）。
ob_sql_work_area_percentage	3	租户里 SQL 可用内存占租户内存比例。

部分变量还可以通过 SQL HINT 设置语句级别的变量，影响范围是所修饰的语句。语句级的变量名会去掉前面的 `ob_` 部分。

下面是常用的几个语句级别的变量命名映射关系。

全局/会话变量名	语句级变量名	描述
ob_query_timeout	query_timeout	指定语句执行超时的阈值
ob_read_consistency	read_consistency	指定读一致性级别，默认是强一致性读，可以修改为弱一致性读

新建租户常用租户变量修改。可修改语句超时时间，避免复杂的语句查询时超时报错。

示例：语句超时报错

```
MySQL [test]> select sleep(11);
ERROR 4012 (HY000): Timeout
```

示例：租户变量初始化

```
set global ob_query_timeout = 100000000;
set global ob_trx_idle_timeout = 1200000000;
set global ob_trx_timeout = 1000000000;
set global ob_trx_lock_timeout = 1000000;
set global ob_sql_work_area_percentage = 50;
```

注意 以上修改完成后均需要会话断开重连，重连后才会生效。

如何对租户资源扩容/缩容

租户扩容或缩容主要是指资源扩容或缩容，具体就是调整租户的 CPU 和内存规格。对租户资源扩容的前提是集群中有节点可以分配出租户需要的资源。

通常会首先考虑租户所在节点是否有可用资源满足租户资源扩容需求；如果不满足，就考虑集群中其他节点是否有资源。后者会触发集群中租户的资源负载均衡，会伴随着数据迁移逻辑。这个数据迁移完全是 OceanBase 内部逻辑，具备高可用能力，不怕异常宕机，不需要 DBA 介入。

租户资源池分裂

在三副本集群里，租户资源池通常也是由三个 ZONE 中三个节点的资源单元（resource unit）组成。在前面创建租户的过程中，每个资源单元使用了统一的资源单元规格，创建了一个包含三个资源单元（同一个资源单元规格）的资源池。这是一种最简单的创建和使用形态。

您可通过下面 SQL 查看资源池细节。

```
select t1.name resource_pool_name, t2.`name` unit_config_name, t2.max_cpu, t2.min_cpu, round(t2.max_me
mory/1024/1024/1024) max_mem_gb, round(t2.min_memory/1024/1024/1024) min_mem_gb, t3.unit_id, t3.zone,
concat(t3.svr_ip,':',t3.`svr_port`) observer,t4.tenant_id, t4.tenant_name
from __all_resource_pool t1 join __all_unit_config t2 on (t1.unit_config_id=t2.unit_config_id)
    join __all_unit t3 on (t1.`resource_pool_id` = t3.`resource_pool_id`)
    left join __all_tenant t4 on (t1.tenant_id=t4.tenant_id)
order by t1.`resource_pool_id`, t2.`unit_config_id`, t3.unit_id
;
```

输出:

```
+--------------------+------------------+---------+---------+------------+------------+---------
| resource_pool_name | unit_config_name | max_cpu | min_cpu | max_mem_gb | min_mem_gb | unit_id
+--------------------+------------------+---------+---------+------------+------------+---------
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       1
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       2
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       3
| my_pool            | unit1            |       9 |       9 |         11 |         11 |    1001
| my_pool            | unit1            |       9 |       9 |         11 |         11 |    1002
| my_pool            | unit1            |       9 |       9 |         11 |         11 |    1003
+--------------------+------------------+---------+---------+------------+------------+---------
6 rows in set (0.132 sec)
```

说明:

- sys 租户有一个资源池 `sys_pool`，规格是 `sys_unit_config`。
- 业务租户 `obmysql` 有一个资源池 `my_pool`，规格是 `unit1`。

下面示例是将业务租户的资源池分裂为三个资源池。

```
ALTER RESOURCE POOL `my_pool` SPLIT INTO ('my_pool_zone1','my_pool_zone2','my_pool_zone3') ON ('

# 再次确认资源池细节，输出:

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

```
| resource_pool_name | unit_config_name | max_cpu | min_cpu | max_mem_gb | min_mem_gb | unit_id
+--------------------+------------------+---------+---------+------------+------------+---------
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       1
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       2
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       3
| my_pool_zone1      | unit1            |       9 |       9 |         11 |         11 |    1001
| my_pool_zone2      | unit1            |       9 |       9 |         11 |         11 |    1002
| my_pool_zone3      | unit1            |       9 |       9 |         11 |         11 |    1003
+--------------------+------------------+---------+---------+------------+------------+---------
6 rows in set (0.010 sec)
```

将原本一个资源池打散为三个资源池，主要是为了便于调整单独 ZONE 里的资源单元规格。该步骤为可选步骤，只是为了便于更细粒度的扩容或缩容租户资源。

租户资源扩容

租户资源扩容有三种方法:

- 调整租户单元使用的资源规格的大小

  示例:

```
alter resource unit sys_unit_config min_cpu=5,min_memory='2G';
alter resource unit unit1 max_cpu=5, min_cpu=5, max_memory='8G', min_memory='8G' ;

+--------------------+------------------+---------+---------+------------+------------+---------
| resource_pool_name | unit_config_name | max_cpu | min_cpu | max_mem_gb | min_mem_gb | unit_id
+--------------------+------------------+---------+---------+------------+------------+---------
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       1
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       2
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       3
| my_pool_zone1      | unit1            |       5 |       5 |          8 |          8 |    1001
| my_pool_zone2      | unit1            |       5 |       5 |          8 |          8 |    1002
| my_pool_zone3      | unit1            |       5 |       5 |          8 |          8 |    1003
+--------------------+------------------+---------+---------+------------+------------+---------
6 rows in set (0.026 sec)
```

- 更换资源单元规格

  示例:

```
CREATE resource unit unit2 max_cpu=8, min_cpu=8, max_memory='10G', min_memory='10G', max_iops=10
alter resource pool my_pool_zone1 unit = 'unit2';

# 查看资源池分布细节输出:
+--------------------+------------------+---------+---------+------------+------------+---------
| resource_pool_name | unit_config_name | max_cpu | min_cpu | max_mem_gb | min_mem_gb | unit_id
+--------------------+------------------+---------+---------+------------+------------+---------
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       1
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       2
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       3
| my_pool_zone1      | unit2            |       8 |       8 |         10 |         10 |    1001
| my_pool_zone2      | unit1            |       5 |       5 |          8 |          8 |    1002
```

```
| my_pool_zone3      | unit1            |       5 |       5 |          8 |          8 |    1003
+--------------------+------------------+---------+---------+------------+------------+---------
6 rows in set (0.047 sec)
```

- 保留资源单元规格不变，增加每个 ZONE 里的资源单元数量。

> **注意** 此方法需要每个 ZONE 中至少有 2 个节点。

具体语法如下:

```
alter resource pool my_pool_zone1 unit_num = 2;
```

默认情况下，租户在每个 ZONE 里的资源单元数量是 1，这种租户架构也是 1-1-1，这种租户也叫小租户。在执行扩容命令时，租户会从集群其他节点里找到可用资源分配该规格的资源单元。

租户资源扩容的时候，很容易遇到资源不足的报错，信息如下:

```
MySQL [oceanbase]> alter resource pool my_pool_zone1 unit_num = 2;
ERROR 4624 (HY000):  machine resource 'zone1' is not enough to hold a new unit
MySQL [oceanbase]>
```

为了避免遇到资源不足的错误，需要对集群可用资源进行准确统计，这就是前面常用的 SQL。为了确保可用资源统计准确，要求所有租户资源的 `min_cpu` 和 `max_cpu` 值保持一致、`min_memory` 和 `max_memory` 值保持一致。

默认 sys 租户的规格 `sys_unit_config` 的最小值和最大值经常不一样，需要手动调整。

```
alter resource unit sys_unit_config min_cpu=5,min_memory='2G';

select a.zone,concat(a.svr_ip,':',a.svr_port) observer, cpu_total, (cpu_total-cpu_assigned) cpu_
free, round(mem_total/1024/1024/1024) mem_total_gb, round((mem_total-mem_assigned)/1024/1024/102
4) mem_free_gb, usec_to_time(b.last_offline_time) last_offline_time, usec_to_time(b.start_servic
e_time) start_service_time, b.status, usec_to_time(b.stop_time) stop_time
from __all_virtual_server_stat a join __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_
port)
order by a.zone, a.svr_ip
;
```

输出:

```
+-------+--------------------+-----------+----------+--------------+-------------+--------------
--------------+----------------------------+--------+----------------------------+
| zone  | observer           | cpu_total | cpu_free | mem_total_gb | mem_free_gb | last_offline_
time          | start_service_time         | status | stop_time                  |
+-------+--------------------+-----------+----------+--------------+-------------+--------------
--------------+----------------------------+--------+----------------------------+
| zone1 | 172.20.249.52:2882 |        14 |        1 |           13 |           1 | 1970-01-01 08
:00:00.000000 | 2021-09-25 08:19:06.622351 | active | 1970-01-01 08:00:00.000000 |
| zone2 | 172.20.249.49:2882 |        14 |        4 |           13 |           3 | 1970-01-01 08
:00:00.000000 | 2021-09-25 08:19:07.392669 | active | 1970-01-01 08:00:00.000000 |
| zone3 | 172.20.249.51:2882 |        14 |        4 |           13 |           3 | 1970-01-01 08
```

```
:00:00.000000 | 2021-09-26 14:05:58.641570 | active | 1970-01-01 08:00:00.000000 |
+-------+--------------------+-----------+----------+--------------+-------------+--------------
--------------+----------------------------+--------+----------------------------+
3 rows in set (0.010 sec)
```

租户资源缩容

租户资源缩容就是调小租户的资源，具体方法和扩容时相同。一种方法是调小资源规格，另外一种方法就是调小资源池对应的资源单元数量（`unit_num`），详细示例请参考租户资源扩容。

租户资源缩容需要考虑一种场景，当租户有大量增量数据在增量内存 MemStore 中时，如果缩小租户资源的内存规格，可能会导致 MemStore 内存水位变高触发转储或限流。

为了避免这种情形，建议租户资源缩容之前，先对集群发起合并，降低 MemStore 里已使用的资源。此外，租户资源内存缩容可以分多步缩容。

当租户资源池对资源单元数量缩容时，会面临一个选择，即该释放那个资源单元。此时如果不指定 UNIT ID，将会由 OceanBase 自动选择。租户缩容结束后，可能会出现集群资源的负载均衡。为了避免这种不必要的负载均衡，可以指定删除的资源单元（UNIT ID）。

示例 SQL，`unit_num` 从 5 变为 4:

```
ALTER RESOURCE POOL pool_total_zone1 unit_num = 4 DELETE unit = (1013 );
```

租户缩容资源，很可能会触发租户资源负载均衡操作和数据迁移逻辑。建议关注数据迁移的进度和影响，可以通过查看集群事件日志视图，观察信息。

```
SELECT DATE_FORMAT(gmt_create, '%b%d %H:%i:%s') gmt_create_ , module, event, name1, value1, name2, val
ue2, rs_svr_ip
FROM __all_rootservice_event_history
WHERE 1 = 1
   AND gmt_create > SUBDATE(now(),interval  1 hour)
ORDER BY gmt_create DESC
LIMIT 20;
```

5.3 如何对 OceanBase 数据库进行备份和恢复

OceanBase 数据库备份/恢复概述

逻辑备份和物理备份

OceanBase 数据库的备份分为以下两类:

- 逻辑备份

  逻辑备份就是数据导出，将数据库对象的定义和数据导出为 SQL 或 CSV 文件等。

  逻辑备份的优点是功能灵活，可以指定租户名、库名、表名和对象类型等导出，针对表的逻辑备份甚至可以指定 QUERY SQL 带条件导出数据。

  缺点则是只支持离线导出，不支持增量导出。

- 物理备份。

  物理备份是数据块备份。目前是按集群级别备份，集群里的所有租户都会被备份。备份的内容包含全量数据、增量数据和事务日志。

  理论上当拥有全量数据备份以及其后的所有事务日志备份，就可以还原出该数据库到历史任意时间点。

备份介质

OceanBase 数据库开源版本 V3.1.0 仅支持逻辑备份与恢复，V3.1.1 开始支持物理备份与恢复。

OceanBase 数据库的逻辑备份与恢复是使用工具 OBDUMPER 和 OBLOADER 进行数据导出和导入，具体操作请参考 4.8 如何使用 OBDUMPER / OBLOADER 工具导出/导入 OceanBase 数据。

OceanBase 数据库的物理备份是将基线数据（sstable）和事务日志（clog）备份到一个共享目录里。该共享目录可以是 NFS 目录、阿里云的 OSS 存储或腾讯云的 COS 存储。每个目录都要挂载到每个 OceanBase 数据库节点的本地文件系统上，OceanBase 数据库备份会自动选择从哪个节点备份数据到该节点的备份目录。

在恢复的时候，目标 OceanBase 集群的每个节点上也需要挂载一个共享目录到本地文件系统，这个目录包含全部备份文件，这样就可以还原到任意历史时间点。

恢复时的共享目录可以和备份是同一个共享目录（也可以不是同一个）。备份时的共享目录通常用于将生产环境的备份还原到线下测试环境。

备份策略

OceanBase 数据库的备份策略跟传统数据库备份一样，支持数据全量备份、数据增量备份和事务日志备份。

在开启全量备份之前，需要先开启事务日志备份。OceanBase 数据库的事务日志备份跟传统数据库的日志备份不完全相同，它是近实时备份。平均每秒钟检查一下事务日志是否有新增，如果有就备份到备份目录里。

OceanBase 数据库的备份跟传统数据库的备份还有个不一样的地方是全量备份的触发要求。OceanBase 数据库的全量备份实际上是备份数据文件里的基线数据（sstable）。所以要求在全量备份之前，需先对集群发起一次合并（major freeze）操作。集群合并结束后，数据文件生成新的基线版本，全量备份就直接备份新版本的数据。

由于 OceanBase 数据库合并之后生成的基线版本是全局一致性的版本，所以 OceanBase 数据库的全量备份也就很容易满足了全局一致性要求。

此后，在 OceanBase 集群下一次合并之前，数据文件里的版本基本不变，因此无需再次做全量备份。所以 OceanBase 数据库会要求每次全量备份之前都有新的基线版本生成，否则就会报错。

租户恢复

OceanBase 数据库目前的物理备份是集群级别的备份（整个集群一起备份）。虽然集群数据是三副本，但备份时只会备份一个副本的数据。OceanBase 数据库的恢复是按租户进行恢复，如果租户架构也是三副本的，那么恢复的时候会自动恢复出三副本，并且有一个主副本和两个备副本。

如何配置备份恢复目录

目前的版本物理备份恢复使用的目录只能是 NFS 共享的目录。

```
alter system set backup_dest='file:///xxx';
```

其中该共享目录 xxx 要挂载到每个 OceanBase 节点的本地文件系统上。

NFS 服务端配置

- 安装 NFS 工具包

  ```
  sudo yum install nfs-utils
  ```

- 设置 NFS 目录

  ```
  mkdir -p /data/backup
  ```

  将该目录设置为 NFS 共享目录。

- 修改配置文件 `/etc/exports`

  ```
  vim /etc/exports
  /data/backup/ 172.xx.xx.0/16(rw,sync,all_squash)
  ```

注意:

- `172.xx.xx.0/16`: 设置的允许访问这个 NFS 目录的客户端 IP 网段。尽量只包含 OceanBase 集群相关节点服务器的 IP 的网段。
- `rw`: 允许客户端读写这个目录。
- `sync`: 同步写模式。
- `all_squash`: 将客户端的所有 UIDs 和 GIDs 映射到 NFSS SERVER 端的匿名用户。

- 设置 NFS 目录权限

```
# centos7
sudo chown nfsnobody:nfsnobody -R /data/backup

# centos8 去掉了 nfsnobody
sudo chown nobody:nobody -R /data/backup
```

- 重启 NFS 服务

```
[admin@obce00 ~]$ sudo systemctl restart nfs-server

[admin@obce00 ~]$ sudo systemctl status nfs-server
● nfs-server.service - NFS server and services
   Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; enabled; vendor preset: disabled)
  Drop-In: /run/systemd/generator/nfs-server.service.d
           └─order-with-mounts.conf
   Active: active (exited) since Sat 2021-10-23 14:19:25 CST; 9min ago
  Process: 1241106 ExecStopPost=/usr/sbin/exportfs -f (code=exited, status=0/SUCCESS)
  Process: 1241104 ExecStopPost=/usr/sbin/exportfs -au (code=exited, status=0/SUCCESS)
  Process: 1241102 ExecStop=/usr/sbin/rpc.nfsd 0 (code=exited, status=0/SUCCESS)
  Process: 1241129 ExecStart=/bin/sh -c if systemctl -q is-active gssproxy; then systemctl reloa
d gssproxy ; fi (code=exited, status=0/SUCCESS)
  Process: 1241117 ExecStart=/usr/sbin/rpc.nfsd (code=exited, status=0/SUCCESS)
  Process: 1241114 ExecStartPre=/usr/sbin/exportfs -r (code=exited, status=0/SUCCESS)
 Main PID: 1241129 (code=exited, status=0/SUCCESS)
    Tasks: 0 (limit: 195588)
   Memory: 0B
   CGroup: /system.slice/nfs-server.service

Oct 23 14:19:25 obce00 systemd[1]: Starting NFS server and services...
Oct 23 14:19:25 obce00 systemd[1]: Started NFS server and services.

[admin@obce00 ~]$ sudo showmount -e
Export list for obce00:
/data/backup 172.xx.xx.0/16

# centos 8

[admin@obce00 ~]$ sudo exportfs -arv
exporting 172.xx.xx.0/16:/data/backup

[admin@obce00 ~]$ sudo exportfs -s
/data/backup  172.xx.xx.0/16(sync,wdelay,hide,no_subtree_check,sec=sys,rw,secure,root_squash,all
```

```
_squash)
```

NFS 客户端配置

所有 OceanBase 集群节点都要部署 NFS 客户端，将 NFS SERVER 的共享目录挂载到本机。客户端也需要先安装 NFS 软件。

- 安装 NFS 工具包

```
sudo yum install nfs-utils
```

- 修改内核参数

```
sudo vim /etc/sysctl.conf +R

sunrpc.tcp_max_slot_table_entries=128
```

- 挂载目录

```
sudo mount -tnfs4 -o rw,timeo=30,wsize=1048576,rsize=1048576,namlen=512,sync,lookupcache=positiv
e 172.xx.xxx.54:/data/backup /backup
```

> **注意**
>
> 挂载前需要先运行以下命令创建挂载目录。手动挂载的目录，在机器重启后需要重新挂载，也可以在文件 `/etc/fstab` 目录里配置。

```
sudo mkdir /backup
```

- 查看结果

```
[admin@obce01 ~]$ mount |grep backup
172.xx.xxx.54:/data/backup on /backup type nfs4 (rw,relatime,sync,vers=4.2,rsize=1048576,wsize=1
048576,namlen=255,hard,proto=tcp,timeo=30,retrans=2,sec=sys,clientaddr=172.xx.xxx.53,lookupcache
=pos,local_lock=none,addr=172.xx.xxx.54)
[admin@obce01 ~]$
```

如何发起 OceanBase 集群备份

OceanBase 数据库的集群备份过程包含以下几步：

1. **（可选）配置事务日志备份**
2. **发起 MINOR FREEZE**

3. **开启事务日志备份**
4. **发起 MAJOR FREEZE**
5. **发起全量备份**

（可选）配置事务日志备份

如果备份目标是：希望能恢复到历史任意时间点。则必须配置事务日志备份。 事务日志备份支持两种模式：`Optional` 和 `Mandatory`。

- `Optional` 模式：表示业务优先。

  在事务日志备份来不及的情况下，日志可能来不及备份就被回收，将会发生事务日志备份断流。这是默认行为。

- `Mandatory` 模式：表示备份优先。

  在该模式下，如果事务日志备份速度跟不上用户事务日志的写入速度，可能会导致事务提交变慢甚至报错。

  通常情况下，如果事务日志盘目录空间满足要求（OBSERVER 节点内存的 3~4 倍），并且备份存储性能不是太差，出现这种备份事务日志赶不上的情况的概率很低。

事务日志压缩目前支持的压缩算法有：`zstd_1.3.8` 和 `lz4_1.0`，默认使用压缩算法 `lz4_1.0`。

日志压缩需要事先开启。

```
ALTER SYSTEM SET backup_log_archive_option = 'optional compression= enable';
```

设置后，也可以修改备份模式和压缩算法。

```
ALTER SYSTEM SET backup_log_archive_option='mandatory compression= zstd_1.3.8';
```

也可以关闭相应备份模式下的事务日志压缩。默认备份模式是 `optional`，建议修改时都带上实际的备份模式。

```
ALTER SYSTEM SET backup_log_archive_option = 'optional compression= disable';
或
ALTER SYSTEM SET backup_log_archive_option = 'mandatory compression= disable';
```

发起 MINOR FREEZE

事务日志开始备份时，并不是从当前的事务日志开始备份，而是从上一次 minor freeze 开始。

> **注意**
>
> 上次合并时默认也会有一个 minor freeze。 所以，为了降低事务日志开始备份的时间，您需先发起一个 minor freeze。

```
alter system minor freeze ;
```

`minor freeze` 命令会瞬时返回，很快就会有转储操作。转储的时间根据当前内存中增量数据数量而定，通常在几秒到几十秒。转储操作对租户读写性能的影响很低。

开启事务日志备份

事务日志的备份是通过命令 `archihvelog` 触发的。archivelog 启动后，集群的事务日志定期备份到指定的备份目录。触发的时间间隔由参数 `log_archive_checkpoint_interval` 指定，范围为 [5s，1h] ，默认时间是 120s。

以当前版本（V3.1.1）的能力，这个参数默认值不建议调小。因为事务日志的备份是以分区或者分区组为单位进行的，当租户的分区数非常多的时候，备份并发的小 IO 会比较多，短期内的 IOPS 可能到达事务日志盘的瓶颈。

```
alter system archivelog;
```

事务日志备份开始后，可以查看备份进度确认备份任务开始。

```
MySQL [oceanbase]> select incarnation, tenant_id, status, min_first_time, max_next_time, now() from CD
B_OB_BACKUP_ARCHIVELOG_SUMMARY;
+-------------+-----------+--------+----------------------------+----------------------------+--------
-------------+
| incarnation | tenant_id | status | min_first_time             | max_next_time              | now(
)               |
+-------------+-----------+--------+----------------------------+----------------------------+--------
-------------+
|           1 |         1 | DOING  | 2021-10-20 17:49:15.079198 | 2021-10-24 16:18:24.531211 | 2021-10
-24 16:20:17 |
|           1 |      1001 | DOING  | 2021-10-20 17:49:15.079198 | 2021-10-24 16:18:24.531211 | 2021-10
-24 16:20:17 |
+-------------+-----------+--------+----------------------------+----------------------------+--------
-------------+
2 rows in set (0.589 sec)
```

事务日志备份是按租户分开进行的。列 `status` 为 `DOING` 时，表示事务日志在进行。`max_next_time` 表示最近一次的备份时间。从时间可以看出基本上是 2 分钟触发一次。

发起 MAJOR FREEZE

默认情况下，OceanBase 数据库增量数据都在内存里。当触发转储时，内存中的增量数据直接以 SSTable 格式写到磁盘，但并没有跟磁盘上的基线数据进行合并。所以在上一次合并结束后，基线数据的内容是没有变化的。

OceanBase 数据库的数据备份就是备份磁盘上数据文件里的基线数据。如果这个基线数据的时间是在事务日志开始备份的时间之前，那这个基线数据即使备份成功了也没有意义。如果上次全量备份成功之后基线数据的内容还是没有变化，那再次全量备份也没有意义。

所以，在开启全量数据备份之前要发起合并操作，并且该合并操作需要在事务日志备份开启成功后发起。否则，全量备份会收到相应报错。

```
alter system major freeze;
```

合并的时间取决于增量数据的数量，通常要几分钟到几十分钟。可以通过下面 SQL 观察合并进度。

```
SELECT ZONE,svr_ip,major_version,ss_store_count,merged_ss_store_count,modified_ss_store_count,merge_st
art_time,merge_finish_time,merge_process
FROM __all_virtual_partition_sstable_image_info
order by major_version desc , zone ;
```

如果列 `merge_finish_time` 有数值存在，表示合并结束。

也可以通过下面视图确认合并是否结束。

```
SELECT * FROM __all_zone WHERE name='merge_status';
+----------------------------+----------------------------+-------+--------------+-------+------
| gmt_create                 | gmt_modified               | zone  | name         | value | info
+----------------------------+----------------------------+-------+--------------+-------+------
| 2021-10-20 15:21:20.765982 | 2021-10-24 16:45:57.990705 |       | merge_status |     0 | IDLE
| 2021-10-20 15:21:20.767009 | 2021-10-24 16:44:55.001369 | zone1 | merge_status |     0 | IDLE
| 2021-10-20 15:21:20.767009 | 2021-10-24 16:45:47.005686 | zone2 | merge_status |     0 | IDLE
| 2021-10-20 15:21:20.769822 | 2021-10-24 16:44:31.857504 | zone3 | merge_status |     0 | IDLE
+----------------------------+----------------------------+-------+--------------+-------+------
4 rows in set (0.734 sec)
```

当列 `info` 的值都是 `IDLE` 时，表示该 ZONE 合并完成。

发起全量备份

发起全量备份之前，您可设置备份密码（这一步是可选的）。设置密码是会话级别设置，不设置即为空。

```
MySQL [oceanbase]> set encryption on identified by 'ba******6' only;
Query OK, 0 rows affected (0.225 sec)
```

全量备份发起命令是 `alter system backup database`。如果事务日志备份已经开始，那必须等事务日志备份成功之后才可发起全量备份，否则命令会报错。

```
alter system backup database;

MySQL [oceanbase]> alter system backup database;
ERROR 9040 (HY000): backup can not start, because log archive is not doing. log archive status : BEGIN
NING.
```

此外，重新开始全量备份之前距离事务日志备份开始之后需要进行一次合并，否则命令也会报错。

```
MySQL [oceanbase]> alter system backup database;
ERROR 9040 (HY000): backup can not start, because log archive start timestamp is bigger than frozen ti
mestamp, need major freeze first. start timestamp : 1635069322951942, frozen timestamp : 1635064748441
948 .
```

查看全量备份任务先看视图 `CDB_OB_BACKUP_PROGRESS`。

```
SELECT incarnation, tenant_id, backup_type, bs_key, partition_count, start_time, completion_time, stat
us
FROM CDB_OB_BACKUP_PROGRESS;
```

输出:

```
+-------------+-----------+-------------+--------+-----------------+----------------------------+-----
----------------------+---------+
| incarnation | tenant_id | backup_type | bs_key | partition_count | start_time                 | comp
letion_time            | status  |
+-------------+-----------+-------------+--------+-----------------+----------------------------+-----
----------------------+---------+
|           1 |         1 | D           |      6 |               0 | 2021-10-24 16:56:02.682700 | 2021
-10-24 16:56:14.035231 | RUNNING |
|           1 |      1001 | D           |      6 |             220 | 2021-10-24 16:56:02.682700 | 2021
-10-24 16:56:12.510089 | RUNNING |
+-------------+-----------+-------------+--------+-----------------+----------------------------+-----
----------------------+---------+
2 rows in set (0.456 sec)
```

如果该视图有记录，表示有全量备份任务在进行。如果视图为空，则表示没有全量备份任务或者全量备份任务已经结束。可以查看备份历史记录视图 `CDB_OB_BACKUP_SET_DETAILS`。

```
select incarnation, tenant_id, bs_key , backup_type, encryption_mode, start_time, completion_time, ela
psed_secondes , keep,  output_bytes_display,  output_rate_bytes_display, status
from CDB_OB_BACKUP_SET_DETAILS
order by start_time desc limit 10;
```

输出:

```
+-------------+-----------+--------+-------------+-----------------+----------------------------
| incarnation | tenant_id | bs_key | backup_type | encryption_mode | start_time
+-------------+-----------+--------+-------------+-----------------+----------------------------
|           1 |         1 |      6 | D           | PASSWORD        | 2021-10-24 16:56:02.682700
|           1 |      1001 |      6 | D           | PASSWORD        | 2021-10-24 16:56:02.682700
|           1 |      1001 |      5 | D           | NONE            | 2021-10-21 08:46:28.618568
|           1 |         1 |      5 | D           | NONE            | 2021-10-21 08:46:28.618568
|           1 |      1001 |      4 | D           | NONE            | 2021-10-20 20:26:46.729579
|           1 |         1 |      4 | D           | NONE            | 2021-10-20 20:26:46.729579
|           1 |         1 |      3 | D           | NONE            | 2021-10-20 19:48:24.574229
|           1 |      1001 |      3 | D           | NONE            | 2021-10-20 19:48:24.574229
|           1 |         1 |      2 | D           | PASSWORD        | 2021-10-20 18:14:42.848095
|           1 |      1001 |      2 | D           | PASSWORD        | 2021-10-20 18:14:42.848095
+-------------+-----------+--------+-------------+-----------------+----------------------------
10 rows in set (0.015 sec)
```

（可选） 停止全量备份和事务日志备份

若您想要取消在进行中的全量备份，可以通过命令 `alter system cancel backup` 取消。如果全量备份已经结束或者没有全量备份任务时，使用该命令会报错。

```
ALTER SYSTEM CANCEL BACKUP;
ERROR 9049 (HY000): backup status is stopped, can not cancel
```

您可通过命令 `alter system noarchivelog` 停止事务日志备份任务。如果事务日志备份已经停止，使用该命令会报错。

```
MySQL [oceanbase]> ALTER SYSTEM NOARCHIVELOG;
Query OK, 0 rows affected (0.552 sec)

MySQL [oceanbase]> ALTER SYSTEM NOARCHIVELOG;
ERROR 9024 (HY000): log archive backup is already disabled
```

设置备份清理策略

默认情况下，OceanBase 数据库的备份会一直保存。您可通过参数 `backup_recovery_window` 设置。参数默认值是 `0` 表示备份永不过期，可以设置为 `7d` 等。

```
MySQL [oceanbase]> alter system set backup_recovery_window='2d';
Query OK, 0 rows affected (0.442 sec)
```

仅设置参数 `backup_recovery_window` 的情况下，OceanBase 数据库不会自动清理备份，您还需要设置参数 `auto_delete_expired_backup` 值为 `true`。

```
MySQL [oceanbase]> alter system set auto_delete_expired_backup=true;
Query OK, 0 rows affected (0.071 sec)
```

自动删除备份会在参数变更后异步调用，查看备份历史可以看到很多备份集在自动删除中。

```
select incarnation, tenant_id, bs_key , backup_type, encryption_mode, start_time, completion_time, ela
psed_secondes , keep,  output_bytes_display,  output_rate_bytes_display, status
from CDB_OB_BACKUP_SET_DETAILS
order by start_time desc ;
```

输出:

```
+-------------+-----------+--------+-------------+-----------------+----------------------------
| incarnation | tenant_id | bs_key | backup_type | encryption_mode | start_time
+-------------+-----------+--------+-------------+-----------------+----------------------------
|           1 |         1 |      7 | D           | PASSWORD        | 2021-10-24 19:05:29.023969
|           1 |      1001 |      7 | D           | PASSWORD        | 2021-10-24 19:05:29.023969
|           1 |         1 |      6 | D           | PASSWORD        | 2021-10-24 16:56:02.682700
|           1 |      1001 |      6 | D           | PASSWORD        | 2021-10-24 16:56:02.682700
|           1 |         1 |      5 | D           | NONE            | 2021-10-21 08:46:28.618568
```

```
|           1 |      1001 |      5 | D           | NONE            | 2021-10-21 08:46:28.618568
|           1 |         1 |      4 | D           | NONE            | 2021-10-20 20:26:46.729579
|           1 |      1001 |      4 | D           | NONE            | 2021-10-20 20:26:46.729579
|           1 |         1 |      3 | D           | NONE            | 2021-10-20 19:48:24.574229
|           1 |      1001 |      3 | D           | NONE            | 2021-10-20 19:48:24.574229
|           1 |         1 |      2 | D           | PASSWORD        | 2021-10-20 18:14:42.848095
|           1 |      1001 |      2 | D           | PASSWORD        | 2021-10-20 18:14:42.848095
+-------------+-----------+--------+-------------+-----------------+----------------------------
12 rows in set (0.155 sec)
```

关于自动删除备份具体逻辑还比较复杂，详情请参考 自动清理备份的数据。

如果自动删除过期备份没有起作用，您还可使用命令 `ALTER SYSTEM DELETE BACKUPSET [备份集ID]` 手动删除备份。

其中，备份集 ID 可从视图 `CDB_OB_BACKUP_SET_DETAILS` 的列 `BS_KEY` 中获取。重复删除同一个备份集将会提示删除中。

```
MySQL [oceanbase]> ALTER SYSTEM DELETE BACKUPSET 1;
Query OK, 0 rows affected (0.759 sec)

MySQL [oceanbase]> ALTER SYSTEM DELETE BACKUPSET 1;
ERROR 9044 (HY000): delete backup data is in progress
```

查看视图 `CDB_OB_BACKUP_TASK_CLEAN_HISTORY` 可以获取备份删除历史。

```
select incarnation, tenant_id, bs_key, backup_type, partition_count, start_time, completion_time, stat
us
from CDB_OB_BACKUP_TASK_CLEAN_HISTORY;
```

输出:

```
+-------------+-----------+--------+-------------+-----------------+----------------------------
| incarnation | tenant_id | bs_key | backup_type | partition_count | start_time
+-------------+-----------+--------+-------------+-----------------+----------------------------
|           1 |         1 |      1 | D           |             218 | 2021-10-20 17:58:42.301646
|           1 |         1 |      2 | D           |             218 | 2021-10-20 18:14:42.848095
|           1 |         1 |      3 | D           |             218 | 2021-10-20 19:48:24.574229
|           1 |         1 |      4 | D           |             218 | 2021-10-20 20:26:46.729579
|           1 |      1001 |      1 | D           |             218 | 2021-10-20 17:58:42.301646
|           1 |      1001 |      2 | D           |             218 | 2021-10-20 18:14:42.848095
|           1 |      1001 |      3 | D           |             218 | 2021-10-20 19:48:24.574229
|           1 |      1001 |      4 | D           |             218 | 2021-10-20 20:26:46.729579
+-------------+-----------+--------+-------------+-----------------+----------------------------
8 rows in set (0.003 sec)
```

当备份集很大的时候，备份的清理会需要一些时间。如果要取消正在删除中的备份，可以使用命令 `ALTER SYSTEM CANCEL DELETE BACKUP;` 取消正在执行的清理任务。已经删除的备份文件不会恢复。同时还要关闭自动清理备份机制。

```
ALTER SYSTEM SET auto_delete_expired_backup = 'False';
```

```
ALTER SYSTEM CANCEL DELETE BACKUP;
```

如何发起 OceanBase 数据库租户恢复

OceanBase 数据库的恢复是按租户恢复，且只能恢复到一个空租户环境。

这里模拟租户被删除故障，执行租户恢复过程。

```
MySQL [oceanbase]> select now();
+---------------------+
| now()               |
+---------------------+
| 2021-10-24 22:55:20 |
+---------------------+
1 row in set (0.002 sec)

MySQL [oceanbase]> drop tenant obmysql force;
Query OK, 0 rows affected (0.110 sec)

MySQL [oceanbase]> drop resource pool pool_mysql;
```

准备空租户的资源池

备份租户前需创建租户的资源池。

首先您需有一个规格合适的资源单元规格，之后您可以运行以下命令，使用该资源单元规格创建资源池。

```
create resource pool my_pool unit='my_unit_config',unit_num=2;
```

查看备份集

查看当前有效的备份集。

```
select incarnation, tenant_id, bs_key , backup_type, encryption_mode, start_time, completion_time, ela
psed_secondes , keep,  output_bytes_display,  output_rate_bytes_display, status
from CDB_OB_BACKUP_SET_DETAILS
order by start_time desc ;
```

输出:

```
+-------------+-----------+--------+-------------+-----------------+----------------------------
| incarnation | tenant_id | bs_key | backup_type | encryption_mode | start_time
+-------------+-----------+--------+-------------+-----------------+----------------------------
|           1 |         1 |      7 | D           | PASSWORD        | 2021-10-24 19:05:29.023969
|           1 |      1001 |      7 | D           | PASSWORD        | 2021-10-24 19:05:29.023969
```

```
|           1 |         1 |      6 | D           | PASSWORD        | 2021-10-24 16:56:02.682700
|           1 |      1001 |      6 | D           | PASSWORD        | 2021-10-24 16:56:02.682700
|           1 |         1 |      5 | D           | NONE            | 2021-10-21 08:46:28.618568
|           1 |      1001 |      5 | D           | NONE            | 2021-10-21 08:46:28.618568
+-------------+-----------+--------+-------------+-----------------+---------------------------
6 rows in set (0.008 sec)
```

这里有 3 个备份集，其中备份集 5 是没有加密的，备份集 6 和 7 是加密的。这里假设使用备份集 7 来恢复，那么需要在会话级别先设置备份集的解密密码。

```
MySQL [oceanbase]> SET DECRYPTION IDENTIFIED BY 'ba******6';
Query OK, 0 rows affected (0.001 sec)
```

打开租户恢复参数

参数 `restore_concurrency` 指定了恢复线程的并发数，默认是 `0`，不恢复。需要修改为大于 `0` 的值。

```
MySQL [oceanbase]> alter system set restore_concurrency = 8;
Query OK, 0 rows affected (0.130 sec)
```

通常开启恢复命令后默认还需等待一段时间才开始恢复，整个恢复期间会有三次等待。每次等待时间由内部参数 `_restore_idle_time` 设置，默认值是 `60s`。

注意

隐含参数未来版本可能会发生变化。生产环境不建议调整这个参数，在测试的时候，如果追求恢复调度时间尽可能的短，可以缩小这个时间到 10s。

```
MySQL [oceanbase]> ALTER SYSTEM SET _restore_idle_time = '10s';
Query OK, 0 rows affected (0.017 sec)
```

开始恢复

恢复命令如下:

```
 ALTER SYSTEM RESTORE <dest_tenant_name> FROM <source_tenan_tname> at 'uri' UNTIL 'timestamp' WITH 're
store_option';
```

参数	描述
dest_tenant_name	指恢复的新租户的名称。
source_te	指原集群的租户。

nant_name	
uri	指备份时设置的 backup_dest。
timestamp	指恢复的时间戳，需要大于等于最早备份的数据备份的 CDB_OB_BACKUP_SET_DETAILS 的 START_TIME，小于等于日志备份 CDB_OB_BACKUP_ARCHIVELOG_SUMMARY 的 MAX_NEXT_TIME。
restore_option	支持 backup_cluster_name、backup_cluster_id、pool_list、locality、kms_encrypt: backup_cluster_name 为必选项，填写源集群的名称。 backup_cluster_id 为必选项，填写源集群的 cluster_id。 pool_list 为必选项，填写用户的资源池。 locality 为可选项，填写租户的 Locality 信息。 kms_encrypt 为可选项，为true则表示在恢复时需要使用指定的 kms_encrypt_info。

```
MySQL [oceanbase]> ALTER SYSTEM RESTORE obmysql2 FROM obmysql at 'file:///backup' UNTIL '2021
Query OK, 0 rows affected (0.015 sec)

MySQL [oceanbase]> select * from __all_tenant;
+----------------------------+----------------------------+-----------+-------------+-----------
| gmt_create                 | gmt_modified               | tenant_id | tenant_name | replica_nu
+----------------------------+----------------------------+-----------+-------------+-----------
| 2021-10-20 15:21:20.569347 | 2021-10-20 15:21:20.569347 |         1 | sys         |          -
| 2021-10-25 00:08:10.322557 | 2021-10-25 00:08:10.322557 |      1005 | obmysql2    |          -
+----------------------------+----------------------------+-----------+-------------+-----------
2 rows in set (0.005 sec)
```

查看恢复进度和结果

- 查看集群事件日志

```
SELECT DATE_FORMAT(gmt_create, '%b%d %H:%i:%s') gmt_create_ , module, event, name1, value1, name
2, value2, rs_svr_ip,name3,value3,name4,value4
FROM __all_rootservice_event_history
WHERE 1 = 1
  AND module IN ('physical_restore','leader_coordinator')
ORDER BY gmt_create DESC
limit 30;
```

输出:

```
+---------------+-------------------+----------------------+------------+--------------------
| gmt_create_   | module            | event                | name1      | value1
+---------------+-------------------+----------------------+------------+--------------------
| Oct25 00:12:23 | leader_coordinator | switch_leader        | current_rs | "172.xx.xxx.53:2882
| Oct25 00:12:23 | leader_coordinator | switch_leader        | current_rs | "172.xx.xxx.53:2882
```

```
| Oct25 00:12:23 | leader_coordinator | switch_leader         | current_rs | "172.xx.xxx.53:2882
| Oct25 00:12:23 | leader_coordinator | switch_leader         | current_rs | "172.xx.xxx.53:2882
| Oct25 00:12:23 | leader_coordinator | switch_leader         | current_rs | "172.xx.xxx.53:2882
| Oct25 00:12:23 | leader_coordinator | switch_leader         | current_rs | "172.xx.xxx.53:2882
| Oct25 00:12:23 | leader_coordinator | switch_leader         | current_rs | "172.xx.xxx.53:2882
| Oct25 00:12:23 | leader_coordinator | switch_leader         | current_rs | "172.xx.xxx.53:2882
| Oct25 00:12:23 | leader_coordinator | switch_leader         | current_rs | "172.xx.xxx.53:2882
| Oct25 00:12:23 | leader_coordinator | switch_leader         | current_rs | "172.xx.xxx.53:2882
| Oct25 00:12:16 | physical_restore   | restore_success       | tenant     | obmysql2
| Oct25 00:12:16 | physical_restore   | change_restore_status | job_id     | 10
| Oct25 00:12:16 | physical_restore   | change_restore_status | job_id     | 10
| Oct25 00:12:16 | physical_restore   | change_restore_status | job_id     | 10
| Oct25 00:09:45 | leader_coordinator | switch_leader         | current_rs | "172.xx.xxx.53:2882
| Oct25 00:09:34 | physical_restore   | change_restore_status | job_id     | 10
| Oct25 00:09:33 | physical_restore   | change_restore_status | job_id     | 10
| Oct25 00:09:27 | physical_restore   | change_restore_status | job_id     | 10
| Oct25 00:09:27 | physical_restore   | change_restore_status | job_id     | 10
| Oct25 00:09:27 | physical_restore   | change_restore_status | job_id     | 10
| Oct25 00:09:12 | leader_coordinator | switch_leader         | current_rs | "172.xx.xxx.53:2882
| Oct25 00:08:10 | physical_restore   | change_restore_status | job_id     | 10
| Oct25 00:08:10 | physical_restore   | change_restore_status | job_id     | 10
| Oct25 00:08:10 | physical_restore   | restore_start         | ret        | 0
| Oct25 00:05:56 | physical_restore   | restore_failed        | tenant     | obmysql2
| Oct25 00:05:56 | physical_restore   | change_restore_status | job_id     | 9
| Oct25 00:05:56 | physical_restore   | change_restore_status | job_id     | 9
| Oct25 00:05:56 | physical_restore   | restore_start         | ret        | 0
| Oct25 00:05:10 | leader_coordinator | switch_leader         | current_rs | "172.xx.xxx.53:2882
| Oct25 00:05:10 | leader_coordinator | switch_leader         | current_rs | "172.xx.xxx.53:2882
+----------------+--------------------+-----------------------+------------+--------------------
30 rows in set (0.083 sec)
```

- 查看恢复进度看视图 `__all_restore_info`

```
obclient> SELECT * FROM __all_restore_info;
```

- 查看恢复历史 `__all_restore_history`

```
MySQL [oceanbase]> SELECT * FROM __all_restore_history order by gmt_create desc limit 1;
+----------------------------+----------------------------+--------+-----------------+----------
-+-------------+-----------------+----------------------------+----------------------------+----
------+-----------------+-----------------++
| gmt_create                 | gmt_modified               | job_id | external_job_id | tenant_i
d | tenant_name | status          | start_time                 | completion_time            | pg
_count | finish_pg_count | partition_count ||
+----------------------------+----------------------------+--------+-----------------+----------
-+-------------+-----------------+----------------------------+----------------------------+----
------+-----------------+-----------------++
| 2021-10-25 00:12:16.700153 | 2021-10-25 00:12:16.700153 |     10 |              -1 |      100
5 | obmysql2    | RESTORE_SUCCESS | 2021-10-25 00:08:10.270205 | 2021-10-25 00:12:16.700153
|      100 |              100 |              100 ||
+----------------------------+----------------------------+--------+-----------------+----------
-+-------------+-----------------+----------------------------+----------------------------+----
------+-----------------+-----------------++
1 row in set (0.003 sec)
```

- 查看恢复结果

```
[admin@obce00 ~]$ obclient -h172.xx.xxx.54 -uroot@obmysql2#obdemo -P2883 -p***** -c -A tpccdb -
e "select max(o_entry_d) from bmsql_oorder;"
+---------------------+
| max(o_entry_d)      |
+---------------------+
| 2021-10-24 19:07:39 |
+---------------------+
```

5.4 如何监控 OceanBase 数据库和配置告警

如何使用传统监控产品监控 OceanBase 数据库

OceanBase 数据库是单进程软件，通常情况下，其性能瓶颈首先不会是 IO，更可能是 CPU、内存和网络等。当然，IO 也会影响 OceanBase 数据库租户的读写性能。

在第二章部署中，我们介绍了 OceanBase 数据库软件的 IO 有三类:

- 运行日志
- 数据文件
- 事务日志

生产环境建议用三块独立的磁盘存储，或者至少使用三个独立的文件系统。

针对 OceanBase 数据库主机，建议部署如下监控。

监控项	描述	应对策略
CPU	监控 `CPU` 的 `USER`、`SYS`、`IOWAIT` 和整体利用率。	分析 `CPU` 利用率高的进程。如果是 `observer` 进程，则进一步分析 `SQL`。
LOAD	主机的负载，通常和 CPU 密切相关。	
内存	监控剩余内存。	剩余内存通常很稳定，如果小于 `1G`，则进程 `observer` 或 `obproxy` 有 `OOM` 风险。
监听 `2881/2882/2883` 端口	`2881` 是 `observer` 连接端口，`2883` 是 `obproxy` 连接端口。	确认进程是否存活，查看进程运行日志，分析进程故障或监听失败原因。 立即重启进程。
网络流量	监控流量是否打满网卡（万兆）	分析集群是否发生负载均衡，调低数据迁移的并发或者分析是否有大量数据抽取。
IO 吞吐量、利用率和延时	监控数据盘和日志盘的 IO 延时、吞吐量和 IO 利用率。	分析集群是否发生负载均衡，调低数据迁移的并发或者分析是否有大量数据抽取。 分析是否坏盘。 分析业务 SQL 的执行计划是否有问题等。

如何使用 Prometheus 监控 OceanBase 数据库

在第 2 章里介绍了 OceanBase 监控插件 OBAgent 如何部署。OBAgent 启动后会自动生成适合 Prometheus 系统的配置文件目录 `prometheus_config`。

Prometheus 安装部署

- 下载 Promethueus 软件

  地址：https://prometheus.io/download/

- 解压缩安装

```
sudo tar zxvf prometheus-2.30.3.linux-amd64.tar.gz -C /usr/local/

# 复制 OBAgent 生成的 Prometheus 配置文件到 Prometheus 安装目录中。
sudo mv prometheus_config/ /usr/local/prometheus-2.30.3.linux-amd64/
```

- Prometheus 服务文件

```
sudo mkdir /var/lib/prometheus
sudo vim /etc/systemd/system/prometheus.service

[Unit]
Description=Prometheus Server
Documentation=https://prometheus.io/docs/introduction/overview/
After=network-online.target

[Service]
Restart=on-failure
ExecStart=/usr/local/prometheus-2.30.3.linux-amd64/prometheus --config.file=/usr/local/prometheu
s-2.30.3.linux-amd64/prometheus_config/prometheus.yaml --storage.tsdb.path=/var/lib/prometheus -
-web.enable-lifecycle --web.external-url=http://172.20.249.54:9090

[Install]
WantedBy=multi-user.target
```

- 启动服务

```
sudo systemctl daemon-reload

sudo systemctl start prometheus

sudo systemctl status prometheus
[admin@obce00 ~]$ sudo systemctl status prometheus
● prometheus.service - Prometheus Server
 Loaded: loaded (/etc/systemd/system/prometheus.service; disabled; vendor preset: disabled)
 Active: active (running) since Thu 2021-10-21 15:54:42 CST; 49s ago
   Docs: https://prometheus.io/docs/introduction/overview/
Main PID: 902555 (prometheus)
  Tasks: 13 (limit: 195588)
 Memory: 40.6M
 CGroup: /system.slice/prometheus.service
         └─902555 /usr/local/prometheus-2.30.3.linux-amd64/prometheus --config.file=/usr/local/
prometheus-2.30.3.linux-amd64/prometheus_config/prometheus.yaml --storage.tsdb.path=/var/lib/pro
metheus --web.enable-lifecycle --web.external-url=http://172.20.249.54:9090

Oct 21 15:54:42 obce00 prometheus[902555]: level=info ts=2021-10-21T07:54:42.275Z caller=head.go
:479 component=tsdb msg="Replaying on-disk memory mappable chunks if any"
```

```
Oct 21 15:54:42 obce00 prometheus[902555]: level=info ts=2021-10-21T07:54:42.275Z caller=head.go
:513 component=tsdb msg="On-disk memory mappable chunks replay completed" duration=2.127µs
Oct 21 15:54:42 obce00 prometheus[902555]: level=info ts=2021-10-21T07:54:42.275Z caller=head.go
:519 component=tsdb msg="Replaying WAL, this may take a while"
Oct 21 15:54:42 obce00 prometheus[902555]: level=info ts=2021-10-21T07:54:42.275Z caller=head.go
:590 component=tsdb msg="WAL segment loaded" segment=0 maxSegment=0
Oct 21 15:54:42 obce00 prometheus[902555]: level=info ts=2021-10-21T07:54:42.275Z caller=head.go
:596 component=tsdb msg="WAL replay completed" checkpoint_replay_duration=39.378µs wal_replay_du
ration=185.207µs total_replay_duration=242.438µs
Oct 21 15:54:42 obce00 prometheus[902555]: level=info ts=2021-10-21T07:54:42.277Z caller=main.go
:849 fs_type=XFS_SUPER_MAGIC
Oct 21 15:54:42 obce00 prometheus[902555]: level=info ts=2021-10-21T07:54:42.277Z caller=main.go
:852 msg="TSDB started"
Oct 21 15:54:42 obce00 prometheus[902555]: level=info ts=2021-10-21T07:54:42.277Z caller=main.go
:979 msg="Loading configuration file" filename=/usr/local/prometheus-2.30.3.linux-amd64/promethe
us_config/prometheus.yaml
Oct 21 15:54:42 obce00 prometheus[902555]: level=info ts=2021-10-21T07:54:42.281Z caller=main.go
:1016 msg="Completed loading of configuration file" filename=/usr/local/prometheus-2.30.3.linux-
amd64/prometheus_config/prometheus.yaml totalDuration=4.630509ms db_storage=1>
Oct 21 15:54:42 obce00 prometheus[902555]: level=info ts=2021-10-21T07:54:42.281Z caller=main.go
:794 msg="Server is ready to receive web requests."
[admin@obce00 ~]$
```

- 确认 Prometheus 是否启动成功

```
sudo netstat -ntlp | grep 9090
[admin@obce00 ~]$ sudo netstat -ntlp | grep 9090
tcp6       0      0 :::9090                 :::*                    LISTEN      902555/prometheu
s
```

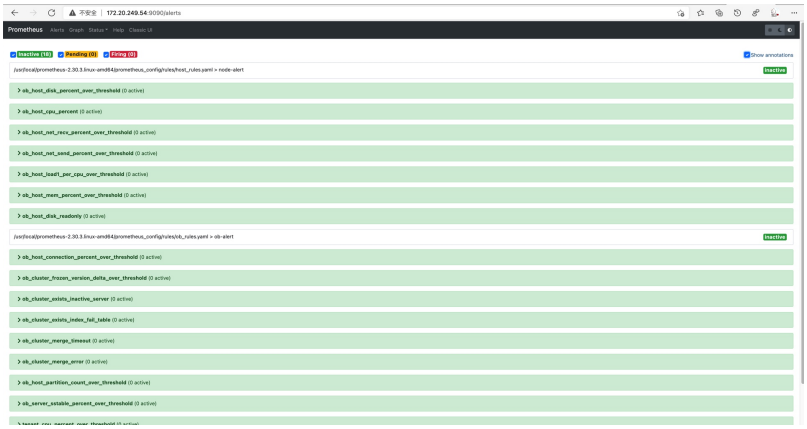
Prometheus 使用

使用浏览器测试：http://172.20.249.54:9090/alerts。

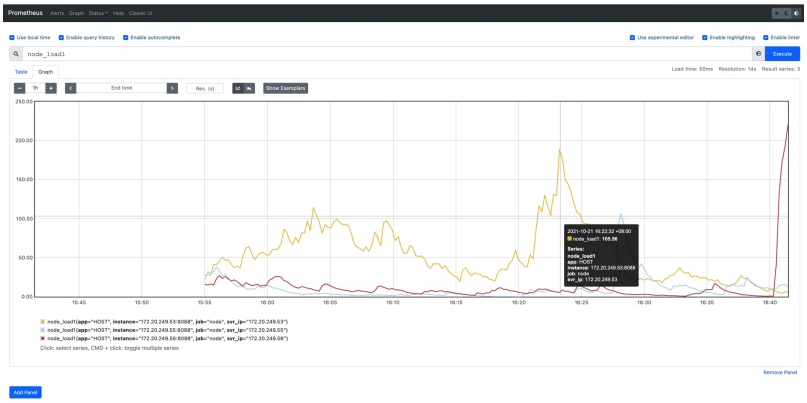
说明

此处链接中的 IP 为示例中配置 Prometheus 的服务器 IP，仅供参考。您需根据实际情况将其转换为自身配置 Prometheus 的服务器 IP。

- 查看告警事件



- 查看节点 LOAD



节点中会涉及到很多自定义的指标名，目前支持的指标名如下:

指标名	Ladel	描述	类型
node_cpu_seconds_total	cpu,mode,svr_ip	CPU 时间	counter
node_disk_read_bytes_total	device,svr_ip	磁盘读取字节数	counter
`node_disk_read_time_seconds_total`	device,svr_ip	磁盘读取消耗总时间	counter
`node_disk_reads_completed_total`	device,svr_ip	磁盘读取完成次数	counter
node_disk_written_bytes_total	device,svr_ip	磁盘写入字节数	counter
`node_disk_write_time_seconds_total`	device,svr_ip	磁盘写入消耗总时间	counter
`node_disk_writes_completed_total`	device,svr_ip	磁盘写入完成次数	counter
node_filesystem_avail_bytes	`device,fstype,mountpoint,svr_ip`	文件系统可用大小	gauge
node_filesystem_readonly	`device,fstype,mountpoint,svr_ip`	文件系统是否只读	gauge
node_filesystem_size_bytes	`device,fstype,mountpoint,svr_ip`	文件系统大小	gauge
node_load1	svr_ip	1 分钟平均 load	gauge

node_load5	svr_ip	5 分钟平均 load	gauge
node_load15	svr_ip	15 分钟平均 load	gauge
node_memory_Buffers_bytes	svr_ip	内存 buffer 大小	gauge
node_memory_Cached_bytes	svr_ip	内存 cache 大小	gauge
node_memory_MemFree_bytes	svr_ip	内存 free 大小	gauge
node_memory_MemTotal_bytes	svr_ip	内存总大小	gauge
`node_network_receive_bytes_total`	device,svr_ip	网络接受总字节数	counter
`node_network_transmit_bytes_total`	device,svr_ip	网络发送总字节数	counter
node_ntp_offset_seconds	svr_ip	NTP 时钟偏移	

5.5 如何对 OceanBase 数据库进行简单性能诊断

如何使用 Tsar 实时监控 OceanBase 数据库主机性能

Tsar 是阿里巴巴开源的主机性能采集软件，采用 C 语言开发，对主机性能占用率很低，工作效率非常高。开源地址为 https://github.com/alibaba/tsar。

Tsar 可以通过编译进行安装。安装后默认每分钟采集一次主机性能，数据存放在 `/var/log/` 目录下的 `tsar.data` 文件中。文件滚动存放，占用空间很小。

Tsar 常用用法如下：

```
# 查看历史性能数据概况，每分钟一笔
tsar -i 1

# 查看 CPU、LOAD、网络历史性能数据，每分钟一笔
tsar --cpu --load --traffic -i 1

# 查看实时 CPU、IO 性能数据，每 3 秒一笔记录
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]# df -h
文件系统                 容量  已用  可用 已用% 挂载点
devtmpfs                  15G     0   15G    0% /dev
tmpfs                     15G     0   15G    0% /dev/shm
tmpfs                     15G  508K   15G    1% /run
tmpfs                     15G     0   15G    0% /sys/fs/cgroup
/dev/vda1                100G  6.8G   94G    7% /
/dev/mapper/obvg-lvredo   20G   45M   19G    1% /redo
/dev/mapper/obvg-lvdata   79G   57M   75G    1% /data
tmpfs                    3.0G     0  3.0G    0% /run/user/0
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]# ll /dev/mapper/obvg-lvredo
lrwxrwxrwx 1 root root 7 9月  22 20:10 /dev/mapper/obvg-lvredo -> ../dm-0
[root@obce00 ob-loader-dumper-2.1.13-SNAPSHOT]# tsar --cpu --io -I dm-0 -l -i 3
Time             ----------------------cpu---------------------- ----------------------------------
-------------------------------dm-0-------------------------------------------------------------
Time                user     sys    wait    hirq    sirq    util    rrqms   wrqms   %rrqm   %wrq
m      rs      ws   rsecs   wsecs  rqsize  rarqsz  warqsz  qusize   await  rawait  wawait   svctm    u
til
29/09/21-22:32:30   0.08    0.08    0.00    0.08    0.04    0.29     0.00    0.00    0.00    0.00    0
.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
```

查看 CPU 性能

CPU 性能也可以使用 `top` 命令查看，原理和使用 Tsar 相同。

通常情况下，在数据库很忙时 load 会偏高，user 利用率也会较高。可以使用 `top` 命令查看是哪类进程。

- 如果是数据库进程，那基本确认是数据库内部性能问题。
- 如果 sys 利用率也相对比较高（如超过 20），则需留意 OS 的异常。

- 如果 wait 利用率相对较高，则需要留意 IO 的性能。

load 的结果里 `load1`、`load5` 和 `load15` 分别代表 1 分钟内、5 分钟内和 15 分钟内 load 的平均值，通过这些值可以看出 load 变化趋势。

```
[root@obce00 ~]$ tsar --cpu --load -l -i 3
Time             ----------------------cpu---------------------- -------------------load------------
-----
Time                user     sys    wait    hirq    sirq    util    load1   load5  load15    runq    p
lit
07/07/19-15:24:54  25.59   10.62    3.08    0.00    0.42   36.63    13.78   12.99   12.15    5.00    2
.8K
07/07/19-15:24:57  25.42   10.63    7.10    0.00    0.42   36.46    13.72   12.99   12.15   10.00    2
.8K
07/07/19-15:25:00  25.25   10.11    3.85    0.00    0.40   35.77    13.58   12.97   12.15    3.00    2
.8K
07/07/19-15:25:03  29.34   11.31    4.89    0.00    0.48   41.13    13.58   12.97   12.15    7.00    2
.8K
07/07/19-15:25:06  24.80    9.93    5.92    0.00    0.36   35.09    13.37   12.94   12.14   25.00    2
.8K
```

查看网络性能

查看网络性能步骤如下:

1. 通过 `ethtool` 命令查看网卡的速度。
2. 通过 `tsar` 命令看网卡实际上传下载速度以及丢包率。如果下载速度接近或者超过网卡的能力，则表示网卡此刻接近吞吐量瓶颈。

```
[root@obce00 ~]$ ethtool  bond0
Settings for bond0:
 Supported ports: [ ]
 Supported link modes:   Not reported
 Supported pause frame use: No
 Supports auto-negotiation: No
 Advertised link modes:  Not reported
 Advertised pause frame use: No
 Advertised auto-negotiation: No
 Speed: 2000Mb/s
 Duplex: Full
 Port: Other
 PHYAD: 0
 Transceiver: internal
 Auto-negotiation: off
 Link detected: yes

[root@obce00 ~]$ tsar --traffic -l -i 3
Time             ---------------------traffic--------------------
Time                bytin  bytout   pktin  pktout  pkterr  pktdrp
07/07/19-15:58:24   8.7M    2.3M    9.9K    6.1K    0.00    0.00
07/07/19-15:58:27   9.0M    2.6M   10.0K    6.2K    0.00    0.00
07/07/19-15:58:30   9.7M    2.7M   10.7K    6.5K    0.00    0.00
07/07/19-15:58:33  10.7M    2.6M   11.0K    6.1K    0.00    0.00
```

```
07/07/19-15:58:36   8.5M    2.2M    9.5K    5.7K    0.00    0.00
^C
```

在分析带宽流量的时候，还可以借助 OS 自己的命令：`iftop`。

```
# 查看网卡 IP 流量
iftop -i eth0 -nNB
# 按 L 、T 、3 、t 、 B 、l 、 p  找出具体哪个 IP 和 PORT 流量最大
```

查看内存性能

Memory 主要关注 free、buffer 和 cache 的变化是否正常。通常运行一段时间后数据库主机的内存分布就会较为固定。如有异常变化，还需要结合其他信息判断。

```
[root@obce00 ~]$ tsar --mem -l -i 3
Time              ----------------------mem----------------------
Time                 free    used    buff    cach   total    util
07/07/19-15:31:46   3.0G   31.2G    1.8G   58.2G   94.2G   33.16
07/07/19-15:31:49   3.0G   31.2G    1.8G   58.2G   94.2G   33.13
07/07/19-15:31:52   3.0G   31.2G    1.8G   58.2G   94.2G   33.15
07/07/19-15:31:55   3.0G   31.2G    1.8G   58.2G   94.2G   33.15
```

查看 IO 性能

查看 IO 性能步骤如下：

1. 确认当前的磁盘分区、文件系统设置等，可通过 `fdisk`、`mount` 和 `df -h` 命令查看。
2. 找到数据库数据和文件所在的磁盘，使用 `tsar` 命令观察 IO 性能。

说明

您应事先对磁盘的吞吐能力和响应时间水平有所了解。

```
[root@obce00 ~]$ ll /dev/mapper/ob_vg-ob_data
lrwxrwxrwx 1 root root 7 Jun 12 15:28 /dev/mapper/ob_vg-ob_data -> ../dm-1

[root@obce00 ~]$ ll /dev/mapper/ob_vg-ob_log
lrwxrwxrwx 1 root root 7 Jun 12 15:28 /dev/mapper/ob_vg-ob_log -> ../dm-0

[root@obce00 ~]$ tsar --io -I dm-0 -l -i 3
Time              ----------------------------------------dm-0-----------------------------------
----
Time               rrqms   wrqms      rs      ws   rsecs   wsecs  rqsize  qusize   await   svctm    ut
il
07/07/19-15:38:09   0.00    0.00    0.00  251.67    0.00    5.3K   21.67    2.00    8.89    3.97   99.
87
07/07/19-15:38:12   0.00    0.00    0.00  231.67    0.00    4.7K   20.85    2.00    9.08    3.95   91.
50
07/07/19-15:38:15   0.00    0.00    0.00  227.33    0.00    1.7K    7.73    1.00    8.71    4.39   99.
```

```
73
07/07/19-15:38:18   0.00    0.00    0.00  213.33    0.00    1.3K    6.36    1.00    8.31    4.31   92.
00
07/07/19-15:38:21   0.00    0.00    0.00  202.33    0.00    1.1K    5.54    1.00    9.09    4.51   91.
20
07/07/19-15:38:24   0.00    0.00    0.00  230.67    0.00    1.3K    5.57    1.00    8.34    4.32   99.
73
07/07/19-15:38:27   0.00    0.00    0.00  203.33    0.00    1.1K    5.65    1.00    8.65    4.49   91.
27
07/07/19-15:38:30   0.00    0.00    0.00  224.67    0.00    1.5K    6.84    1.00    8.34    4.43   99.
47
07/07/19-15:38:33   0.00    0.00    0.00  232.33    0.00    3.0K   13.11    2.00    8.61    3.96   92.
07
07/07/19-15:38:36   0.00    0.00    0.00  210.67    0.00    1.2K    5.80    1.00    8.27    4.36   91.
87
07/07/19-15:38:39   0.00    0.00    0.00  227.33    0.00    1.3K    5.75    1.00    8.16    4.39   99.
90
^C
```

其中 IO Util 是个比较复杂的参数。

- 如果是机械盘，这个 `Util` 可以反映出磁盘的繁忙程度，`svctm` 会在几毫秒以上。
- 如果是 SSD 盘，`svctm` 会在零点几毫秒。

  近似利用率计算公式：利用率 U = 吞吐量 * 每次平均服务时间。所以 SSD 盘需要看响应时间（`svctm`）和等待时间（`await`）等信息综合判断是否到瓶颈。

说明

主机的性能是最容易查看的，如果主机在某方面呈现性能瓶颈，数据库的性能和稳定性都会受到影响。二者之间的联系并不会表现的很直接，需要多观察总结成经验。

如何使用 DOOBA 实时监控 OceanBase 数据库租户性能

`dooba.py` 是采用 Python 语言编写的脚本，其作用是实时读取 OceanBase 数据库租户的性能。

Oracle 数据库中有 AWR 报表，方便诊断人员快速了解数据库在某个时间点的性能问题和原因。AWR 依赖很多内部视图，其中部分视图在 OceanBase 数据库里也存在，但是 OceanBase 数据库暂时还没有支持 AWR 报表。OceanBase 运维平台 OCP 里提供了丰富的性能展示功能，方便对 OceanBase 数据库进行诊断。

OceanBase 数据库里也记录了会话和 SQL 的等待事件，但这一块功能还不是很成熟，大部分性能问题只要分析 SQL 就能解决，所以等待事件可暂不理会。

OceanBase 数据库的 SQL 诊断建议关注租户的 QPS（每秒 SQL 请求数，包括 SELECT、INSERT、UPDATE、DELETE）以及其 RT（SQL 执行耗时）、TPS（每秒事务数，和 Oracle 一致）以及其 RT（事务提交延时）。

此外，还需关注 OceanBase 集群节点上的性能信息。OceanBase 集群的性能瓶颈首先不会出现在 IO，而更容易出现在内存，其次是 CPU。所以还需要关注每秒内存的变化，具体是指增量内存的使用情况。

使用 OceanBase 数据库自带的命令行监控脚本 DOOBA 可以实时观察 OceanBase 数据库租户性能。

您可在 sys 租户创建一个只读的账户，查看系统视图。

```
grant select on oceanbase.* to user1 identified by '*****';
```

```
python dooba.py -hobserver00 -u user1@sys#obdemo -P2883 -p*****
```

快捷键

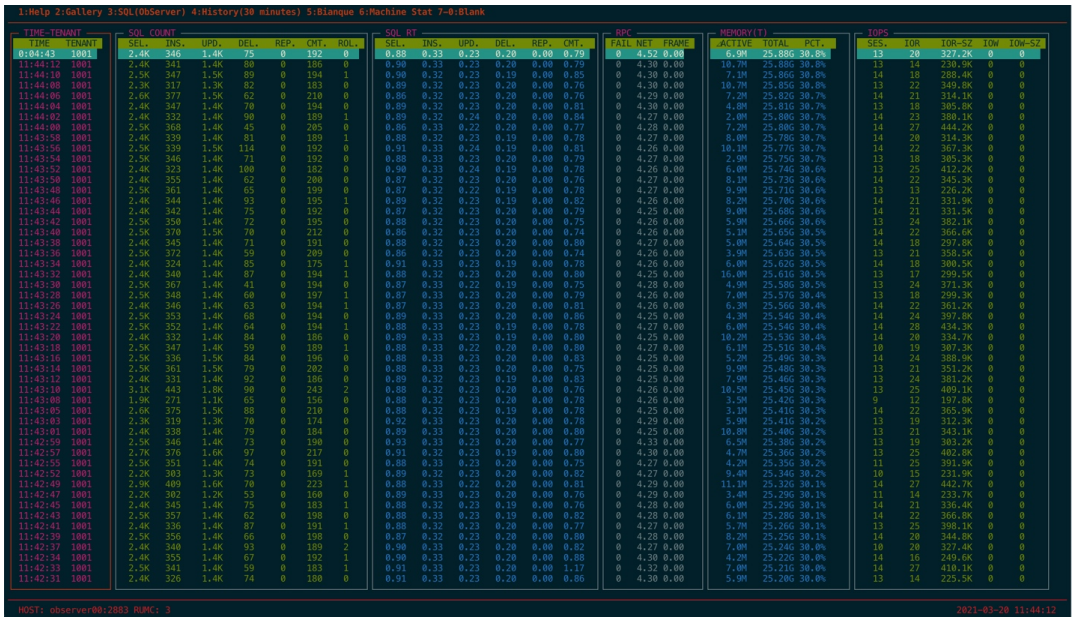
运行之后，按 `1` 可查看帮助。

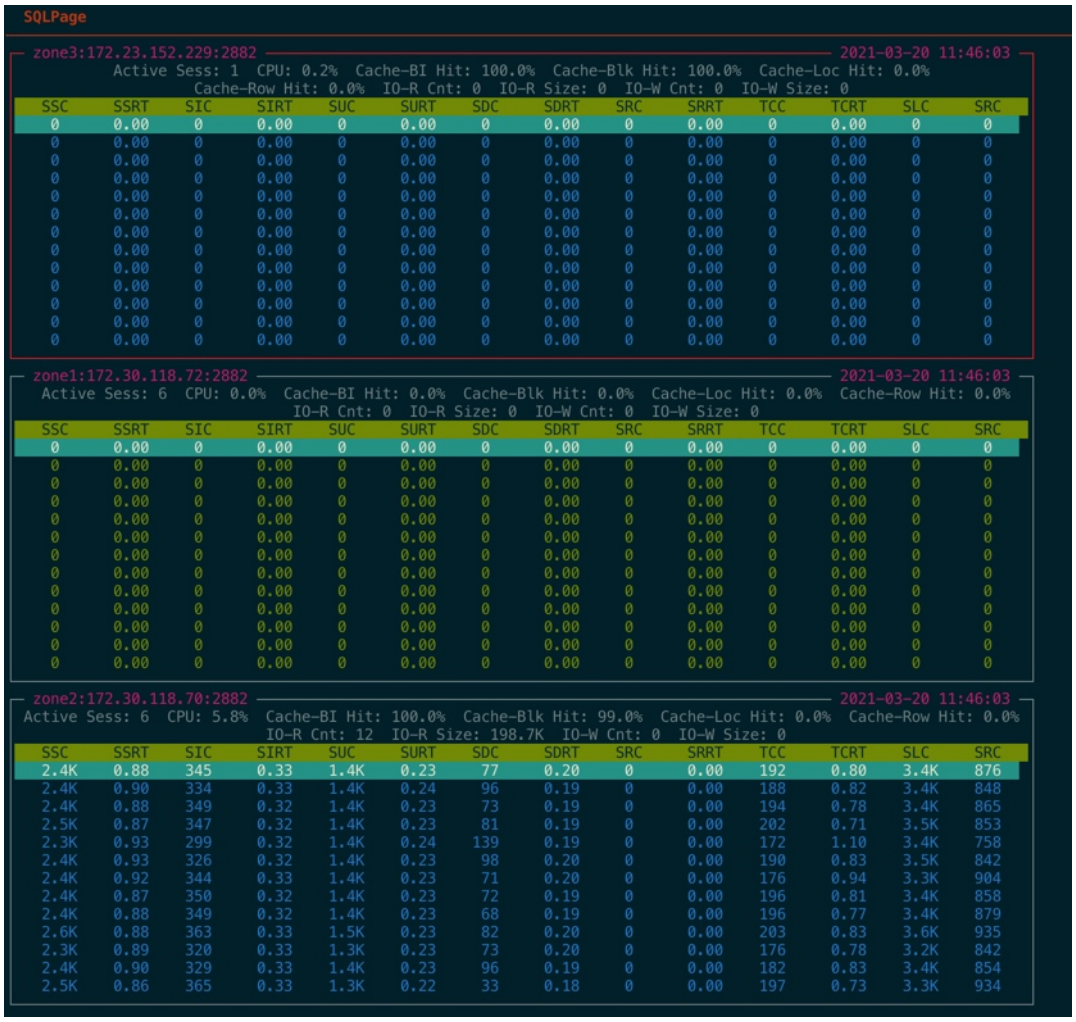
```
Hel
p
                                                                    Shown: 0 / Valid: 0 / Total: 0
_____________________________________________________________________________________________________
_____________________________________________________________________________________________________
_____________________________________________________________________________________________________
_____________________________________

    Global Keys  -  oceanbase
  ----------------------------------------------
        c             : Switch between tenants
        w             : write a screenshot file to current window

    Global Keys  -  Widget
  ----------------------------------------------
        Tab           : Select next widget
        m             : Connect to oceanbase lms for this cluster using mysql
        j             : ssh to selected host

    Global Keys  -  Page
  ----------------------------------------------
        1 F1          : Help page
        2 F2          : Gallery page
        3 F3          : Observer page
        4 F4          : History page
        d             : Delete selected widget
        R             : Restore deleted widgets
        =             : Filter Columns for current page (ms,cs,ups page only)

    Global Keys  -  Test
  ----------------------------------------------
        p             : Messsage box tooltips

    Global Keys  -  Selection Box
  ----------------------------------------------
        DOWN TAB J P  : Next item
        UP K N        : Previous item
        SPC ENTER     : Select current item
        Q q           : Quit selection box

    Global Keys  -  System
  ----------------------------------------------
        q             : quit dooba
```

```
  Support
-------------------------------------------------
      Author         : Yudi Shi (fufeng.syd)
      Mail           : fufeng.syd@alipay.com

      project page   :
      bug report     :
      feature req    :
```

使用最多的快捷键如下:

- `1`: 查看帮助。
- `2`: 查看租户性能总览。
- `3`: 查看各个节点的性能总览，但节点多的时候会展示不全。
- `c`: 选择租户。通常需要在观察业务租户时使用。
- `tab`: 在各个 TAB 之间跳转。
- `d`: 删除当前 TAB，在屏幕显示不够时使用。
- `R`: 恢复所有的 TAB。





从截图的监控数据中可以简单看出以下信息:

- 租户虽然是三节点，实际只有一个节点在提供读写服务。
- 业务 SQL 大部分是查询和更新，其次是少量插入和删除。
- 查询平均延时 900us（微秒），插入平均延时 320us，更新平均延时 230us，删除平均延时 200us。
- 租户事务平均延时 800us，事务比较小。
- 租户有少量的 IO 读，吞吐量在 230MB。
- 节点远程 SQL 比例约占总 SQL 比例的 20%。

性能指标说明

通过查看脚本内容可以知道各个指标缩写的含义，其数据多取自于视图 `gv$sysstat`，各项指标如下:

一级分类	二级分类	缩写	全称	含义
Galler	SQL COUNT（租户 SQL	SEL.	sql select coun	平均每秒查询次数

y	QPS 数据）		t	
Gallery	SQL COUNT（租户 SQL QPS 数据）	INS.	sql insert count	平均每秒插入次数
Gallery	SQL COUNT（租户 SQL QPS 数据）	UPD.	sql update count	平均每秒更新次数
Gallery	SQL COUNT（租户 SQL QPS 数据）	DEL.	sql delete count	平均每秒删除次数
Gallery	SQL COUNT（租户 SQL QPS 数据）	REP.	sql replace count	平均每秒替换次数
Gallery	SQL COUNT（租户 SQL QPS 数据）	CMT.	trans commit count	平均每秒事务提交次数
Gallery	SQL COUNT（租户 SQL QPS 数据）	ROL.	trans rollback count	平均每秒事务回滚次数
Gallery	SQL RT（租户 SQL 延时数据）	SEL.	sql select time	平均每次查询耗时
Gallery	SQL RT（租户 SQL 延时数据）	INS.	sql insert time	平均每次插入耗时
Gallery	SQL RT（租户 SQL 延时数据）	UPD.	sql update time	平均每次更新耗时
Gallery	SQL RT（租户 SQL 延时数据）	DEL.	sql delete time	平均每次删除耗时
Gallery	SQL RT（租户 SQL 延时数据）	REP.	sql replace time	平均每次替换耗时
Gallery	SQL RT（租户 SQL 延时数据）	CMT.	trans commit time	平均每次事务提交延时
Gallery	MEMORY(T)（租户内存性能数据）	☑ACTIVE	active memstore used	平均每秒增量内存变化量
Gallery	MEMORY(T)（租户内存性能数据）	TOTAL	total memstore used	增量内存累计总量
Gallery	MEMORY(T)（租户内存性能数据）	PCT.	total memstore used	增量内存占比

Gallery	IOPS（集群的 IO 性能数据）	SES.	active sessions	当前活跃会话数
Gallery	IOPS（集群的 IO 性能数据）	IOR	io read count	平均每秒读 IO 次数
Gallery	IOPS（集群的 IO 性能数据）	IOR-SZ	io read bytes	平均每次读 IO 大小
Gallery	IOPS（集群的 IO 性能数据）	IOW	io write count	平均每秒写 IO 次数
Gallery	IOPS（集群的 IO 性能数据）	IOW-SZ	io write bytes	平均每次写 IO 大小
SQL Page	zone（每个节点的性能数据）	Active Sess	active sessions	租户在该节点当前活跃会话数
SQL Page	zone（每个节点的性能数据）	CPU	cpu usage	租户在该节点的 CPU 利用率（相对值）
SQL Page	zone（每个节点的性能数据）	Cache-BI Hit	block index cache hit	租户在该节点数据块的索引块的命中率
SQL Page	zone（每个节点的性能数据）	Cache-Blk Hit	block cache hit	租户在该节点数据块的命中率
SQL Page	zone（每个节点的性能数据）	Cache-Loc Hit	location cache hit	租户分区在该节点位置缓存命中率
SQL Page	zone（每个节点的性能数据）	Cache-Row Hit	row cache hit	租户分区在该节点行缓存命中率
SQL Page	zone（每个节点的性能数据）	IO-R Cnt	io read count	租户在该节点平均每秒读 IO 次数
SQL Page	zone（每个节点的性能数据）	IO-R Size	io read bytes	租户在该节点平均每次读 IO 大小
SQL Page	zone（每个节点的性能数据）	IO-W Cnt	io write count	租户在该节点平均每次写 IO 次数
SQL Page	zone（每个节点的性能数据）	IO-W Size	io write bytes	租户在该节点平均每次写 IO 大小
SQL Page	SQL（每个节点的 SQL 性能数据）	SSC	sql select count	租户在该节点平均每秒查询次数

SQL Page	SQL（每个节点的 SQL 性能数据）	SSRT	sql select time	租户在该节点平均每次查询耗时
SQL Page	SQL（每个节点的 SQL 性能数据）	SIC	sql insert count	租户在该节点平均每秒插入次数
SQL Page	SQL（每个节点的 SQL 性能数据）	SIRT	sql insert time	租户在该节点平均每次插入耗时
SQL Page	SQL（每个节点的 SQL 性能数据）	SUC	sql update count	租户在该节点平均每秒更新次数
SQL Page	SQL（每个节点的 SQL 性能数据）	SURT	sql update time	租户在该节点平均每次更新耗时
SQL Page	SQL（每个节点的 SQL 性能数据）	SDC	sql delete count	租户在该节点平均每秒删除次数
SQL Page	SQL（每个节点的 SQL 性能数据）	SDRT	sql delete time	租户在该节点平均每次删除耗时
SQL Page	SQL（每个节点的 SQL 性能数据）	SRC	sql replace count	租户在该节点平均每秒替换次数
SQL Page	SQL（每个节点的 SQL 性能数据）	SRRT	sql replace time	租户在该节点平均每次替换耗时
SQL Page	SQL（每个节点的 SQL 性能数据）	TCC	trans commit count	租户在该节点平均每秒事务提交次数
SQL Page	SQL（每个节点的 SQL 性能数据）	TCRT	trans commit time	租户在该节点平均每次事务提交延时
SQL Page	SQL（每个节点的 SQL 性能数据）	SLC	sql local count	租户在该节点平均每秒本地 SQL 次数
SQL Page	SQL（每个节点的 SQL 性能数据）	SRC	sql remote count	租户在该节点平均每秒远程 SQL 次数

这些指标可以解答如下问题:

- 是不是每个节点都在提供读写服务？性能分别如何?
- 连接数达到多少了？每个节点是多少?
- IO 吞吐量有多少？每个节点是多少?
- 数据库 SQL 耗时有多少？每个节点是多少?

- 每个节点的跨节点访问的 SQL 是多少?
- 每个节点的数据缓存命中率多少?

如何对 OceanBase 租户内存进行调优

章节 如何管理 OceanBase 数据库内存 中介绍了 OceanBase 数据库内存的组成。每个租户的内存默认会有一半用于增量写入（MemStore），剩余的用于基线数据、SQL 执行以及其他等模块。

实际经验中，调优租户内存主要是调整 MemStore 和 SQL 工作区内存大小。

调优增量内存

调优租户内存的思路是两方面。

- 一方面尽可能扩大节点总内存资源、租户内存配额以及 MemStore 内存的比例。MemStore 内存占比默认是 50%，如果写比读多时，这个值可以小范围上调。
- 另外一方面思路就是调优内存转储，当 MemStore 内存剩余不多的时候，尽可能的转储释放内存。

  以下是示例，具体参数值的大小还要根据实际内存和性能情况微调。

```
alter system set memory_limit_percentage = 90;    --  OceanBase 占系统总内存的比例，提高 OceanBase 可用的内存量。默认值是 80，  主机内存大于 256G 时，这个可以设置到 90，最大不要超过 90。
alter system set memstore_limit_percentage = 55;  --  memstore 占租户的内存比，尽量增大 memstore 的空间（但是可能对读操作有负面影响）。
alter system set freeze_trigger_percentage = 40;  --  启动 major/minor freeze 的时机，让转储（minor freeze）尽早启动，memstore 内存尽早释放。
alter system set minor_freeze_times = 100;        --  minor freeze 的次数，尽量不在测试期间触发 major freeze。
alter system set minor_warm_up_duration_time = 0;  --  加快 minor freeze
```

当然，您也可以为租户设置内存写入限速，也有的是应用自身设置内存限速（比如 DataX 就有自己限速的设计）。

在调整的过程中也可以使用下面 SQL，通过视图 `gv$memstore` 观察 MemStore 的使用情况。

```
SELECT tenant_id, ip, round(active/1024/1024/1024) active_gb, round(total/1024/1024/1024) total_gb, round(freeze_trigger/1024/1024/1024) freeze_trg_gb, round(mem_limit/1024/1024/1024) mem_limit_gb
    , freeze_cnt , round((active/freeze_trigger),2) freeze_pct, round(total/mem_limit, 2) mem_usage
FROM `gv$memstore`
WHERE tenant_id =1002
ORDER BY tenant_id, ip;
```

输出:

```
+-----------+---------------+-----------+----------+---------------+--------------+------------+------------+-----------+
| tenant_id | ip            | active_gb | total_gb | freeze_trg_gb | mem_limit_gb | freeze_cnt | freeze_pct | mem_usage |
```

```
+-----------+--------------+----------+---------+--------------+-------------+-----------+------
------+-----------+
|      1002 | 172.20.249.49 |         1 |        1 |            4 |           5 |         0
|       0.35 |      0.25 |
|      1002 | 172.20.249.51 |         1 |        1 |            4 |           5 |         0
|       0.33 |      0.23 |
|      1002 | 172.20.249.52 |         1 |        2 |            4 |           5 |         0
|       0.34 |      0.41 |
+-----------+--------------+----------+---------+--------------+-------------+-----------+------
------+-----------+
3 rows in set (0.004 sec)
```

视图说明:

字段名称	类型	是否可以为 NULL	描述
CON_ID	NUMBER(38)	NO	租户 ID
SVR_IP	VARCHAR2(32)	NO	服务器的 IP
SVR_PORT	NUMBER(38)	NO	服务器端口
ACTIVE	NUMBER(38)	NO	当前活跃的 Memtable 的内存占用大小，单位为字节
TOTAL	NUMBER(38)	NO	当前所有 Memtable 的内存占用大小，单位为字节
FREEZE_TRIGGER	NUMBER(38)	NO	触发 Memtable 冻结的内存大小，单位为字节
MEM_LIMIT	NUMBER(38)	NO	Memtable 的内存大小限制，单位为字节
FREEZE_CNT	NUMBER(38)	NO	Memtable 的冻结次数

通常情况下，要维持 `mem_usage` 在 90% 以下，如果达到或超过 90%，则有可能触发租户内存写入限速，从而显著降低应用写入速度。

调优其他内存

其他内存模块的大小可以查看视图 `__all_virtual_memory_info` 观察。

```
select zone, svr_ip, label, ctx_name, mod_name, round(hold/1024/1024) hold_mb, round(used/1024/1024) u
sed_mb, count, alloc_count
from __all_virtual_memory_info
where tenant_id = 1002 and label <> 'OB_MEMSTORE'
order by hold desc limit 10;
```

输出:

```
+-------+--------------+------------------+---------------+------------------+---------+------
| zone  | svr_ip       | label            | ctx_name      | mod_name         | hold_mb | used_
+-------+--------------+------------------+---------------+------------------+---------+------
| zone1 | 172.20.249.52 | MysqlRequesReco  | DEFAULT_CTX_ID | MysqlRequesReco  |     197 |     1
| zone2 | 172.20.249.49 | MysqlRequesReco  | DEFAULT_CTX_ID | MysqlRequesReco  |     189 |     1
| zone3 | 172.20.249.51 | MysqlRequesReco  | DEFAULT_CTX_ID | MysqlRequesReco  |     125 |     1
| zone1 | 172.20.249.52 | TransAudit       | DEFAULT_CTX_ID | TransAudit       |      32 |
| zone2 | 172.20.249.49 | TransAudit       | DEFAULT_CTX_ID | TransAudit       |      32 |
| zone3 | 172.20.249.51 | TransAudit       | DEFAULT_CTX_ID | TransAudit       |      32 |
| zone2 | 172.20.249.49 | SqlPlanMon       | DEFAULT_CTX_ID | SqlPlanMon       |      15 |
| zone3 | 172.20.249.51 | OB_KVSTORE_CACHE | DEFAULT_CTX_ID | OB_KVSTORE_CACHE |      14 |
| zone2 | 172.20.249.49 | OB_KVSTORE_CACHE | DEFAULT_CTX_ID | OB_KVSTORE_CACHE |      14 |
| zone1 | 172.20.249.52 | Election         | DEFAULT_CTX_ID | Election         |      12 |
+-------+--------------+------------------+---------------+------------------+---------+------
10 rows in set (0.019 sec)
```

5.6 如何快速处理 OceanBase 数据库故障

如何定位判断 OceanBase 数据库故障

当 OceanBase 数据库故障时，应用端会反馈有很多报错信息。此时需要初步判断应用是全部失败，还是部分失败。

理论上 OceanBase 数据库局部节点故障时，应用只会是局部数据库读写故障或者中断，在 1 分钟左右应用就能全部恢复。同时需按以下步骤尽快确认 OceanBase 集群的状态。

1. 确认集群节点状态

```
select a.zone,concat(a.svr_ip,':',a.svr_port) observer, cpu_total, (cpu_total-cpu_assigned) cpu_
free, round(mem_total/1024/1024/1024) mem_total_gb, round((mem_total-mem_assigned)/1024/1024/102
4) mem_free_gb, usec_to_time(b.last_offline_time) last_offline_time, usec_to_time(b.start_servic
e_time) start_service_time, b.status, usec_to_time(b.stop_time) stop_time
from __all_virtual_server_stat a join __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_
port)
order by a.zone, a.svr_ip
;
```

   您需关注:

   - 节点状态 `status`：升级前没有 `inactive` 值，升级过程中会有。
   - 节点服务时间 `start_service_time`：是否是默认值（`1970-01-01 08:00:00.000000`）。如果是，则表示节点还没有恢复结束。
   - 节点停止时间 `stop_time`：是否是默认值（`1970-01-01 08:00:00.000000`），如果不是，则表示节点被停服（`stop server`）了，需要先启动服务（`start server`）。

2. 确认集群近期事件

```
SELECT DATE_FORMAT(gmt_create, '%b%d %H:%i:%s') gmt_create_ , module, event, name1, value1, name
2, value2, rs_svr_ip
FROM __all_rootservice_event_history
WHERE 1 = 1
   AND module IN ('server','root_service','balancer')
   AND gmt_create > SUBDATE(now(),interval  1 hour)
ORDER BY gmt_create DESC
LIMIT 50;
```

   需着重留意节点掉线和上线事件、合并超时事件、数据迁移事件等。

如何处理节点掉线或宕机故障

修改租户变量允许 DDL

如果有租户的架构是 1-1-1 并且宕机的节点就是该租户的成员，宕机后可能导致租户的 DDL 报错。

```
MySQL [test]> create table t2 like t1;
ERROR 4624 (HY000): machine resource is not enough to hold a new unit
```

此时需要将全局变量 `ob_create_table_strict_mode` 值设置为 `OFF`。

```
set global ob_create_table_strict_mode = off;
```

之后重新登录业务租户，就可以做 DDL 了。

注意

关闭这个变量有风险，需要尽快修复故障节点。

启动 observer 进程

如果节点掉线了，但是节点的进程还在，则很可能有两个原因:

- 节点跟其他节点的时间误差超过 200ms，此时首先应检查时间是否同步。
- 节点的事务日志目录空间使用率超过参数 `clog_disk_usage_limit_percentage` 定义（默认值是 `95`，意为 95%）。

如果节点进程不存在，则首先应尝试拉起进程，之后再观察几分钟，确认进程是否再次推出。

启动进程时需注意用户是否正确。

```
# 切换到正确用户下
cd /home/admin/oceanbase-ce && bin/observer
```

如果进程还是会退出，则需要查看最近的进程运行日志，搜索 ERROR 信息。

```
cd /home/admin/oceanbase-ce
grep ERROR log/observer.log | vim -
```

如何处理节点时间不同步故障

判断集群节点时间同步误差

您必须准确判断节点时间是否同步。您可使用测试命令 `clockdiff` 对集群中所有节点进行相互测试。

```
[root@obce01 ~]# clockdiff 172.20.249.49
```

```
...
host=172.20.249.49 rtt=428(314)ms/0ms delta=0ms/0ms Thu Sep 30 21:33:16 2021
[root@obce01 ~]# clockdiff 172.20.249.51
...
host=172.20.249.51 rtt=426(314)ms/0ms delta=0ms/0ms Thu Sep 30 21:33:22 2021

[root@obce02 ~]# clockdiff 172.20.249.50
.
host=172.20.249.50 rtt=750(187)ms/0ms delta=0ms/0ms Thu Sep 30 21:34:10 2021
[root@obce02 ~]# clockdiff 172.20.249.52
...
host=172.20.249.52 rtt=423(315)ms/0ms delta=-1ms/-1ms Thu Sep 30 21:34:14 2021
```

在某些客户机房环境下，命令 `clockdiff` 可能会报错（原因不明）。此时可使用 `ping` 命令，查看返回的信息判断时间误差。

```
ping -T tsandaddr 172.20.249.49 -c 3

输出:
[root@obce01 ~]# ping -T tsandaddr 172.20.249.49 -c 3
PING 172.20.249.49 (172.20.249.49) 56(124) bytes of data.
64 bytes from 172.20.249.49: icmp_seq=1 ttl=64 time=26.3 ms
TS:     172.20.249.52   48963954 absolute
        172.20.249.49   13
        172.20.249.49   0
        172.20.249.52   13

64 bytes from 172.20.249.49: icmp_seq=2 ttl=64 time=0.333 ms
TS:     172.20.249.52   48964963 absolute
        172.20.249.49   1
        172.20.249.49   0
        172.20.249.52   0

64 bytes from 172.20.249.49: icmp_seq=3 ttl=64 time=0.480 ms
TS:     172.20.249.52   48966011 absolute
        172.20.249.49   0
        172.20.249.49   0
        172.20.249.52   0


--- 172.20.249.49 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2057ms
rtt min/avg/max/mdev = 0.333/9.038/26.302/12.207 ms
```

修复集群节点时间同步服务

在此环节中主要是检查每个节点的时间同步服务是否正常。推荐使用 `chrony` 同步服务，不要使用 `ntpd`。

- 检查 `chrony` 节点同步服务

  使用 `chrony` 时间服务是为了保证 OceanBase 集群各个节点时间尽可能保证同步。如下命令仅供参考，具体使用请查看 `chrony` 官方使用说明：Chronyc Frequently Asked Questions。

```
查看时间同步活动
chronyc activity

查看时间服务器
chronyc sources

查看同步状态
chronyc sources -v

校准时间服务器:
chronyc tracking
```

- 检查节点与选中的时间服务器之间的时间差

  该步骤也使用 `clockdiff` 命令。 如果时间服务器是禁止 `ping` 命令，或者防火墙禁止 `ping` 命令，则无法探知时间误差。

  您可以使用 `ntpdate` 命令快速同步本节点跟时间源之间的时间。测试环境中如果时间总是不同步，可以将这个命令写到 `crontab` 里，每分钟运行一次。

```
ntpdate -b xxx.xxx.xxx.xxx
```

5.7 如何使用 OBD 运维

如何使用 OBD 调整 OceanBase 集群参数

您可通过使用 OBD 调整 OceanBase 集群参数为集群节点内存扩容，以下示例介绍如何将集群节点内存从 8GB 扩容到 16GB。

1. 使用 OBD 命令编辑参数文件

```
obd cluster edit-config obce-3zones

# 修改:
    memory_limit: 16G # The maximum running memory for an observer

# 保存:
oceanbase-ce-3.1.0 already installed.
obproxy-3.1.0 already installed.
Search param plugin and load ok
Parameter check ok
Save deploy "obce-3zones" configuration
```

2. 使用 OBD 命令 reload 参数

```
obd cluster reload obce-3zones

# 输出:
Get local repositories and plugins ok
Open ssh connection ok
Cluster status check ok
Connect to observer ok
Connect to obproxy ok
obce-3zones reload
```

3. 登录 sys 租户确认参数变更

```
MySQL [oceanbase]> show parameters like 'memory_limit';
+-------+----------+---------------+----------+--------------+-----------+-------+----------------------------
----------------------------------------------------------------------------------------------+----------+---------+-
-------------+
| zone  | svr_type | svr_ip        | svr_port | name         | data_type | value | inf
o
| section  | scope   | source  | edit_level        |
+-------+----------+---------------+----------+--------------+-----------+-------+----------------------------
----------------------------------------------------------------------------------------------+----------+---------+-
-------------+
| zone2 | observer | 172.20.249.49 |     2882 | memory_limit | NULL      | 16G   | the size of the memory res
nal use(for testing purpose), 0 means follow memory_limit_percentage. Range: 0, [8G,) | OBSERVER | CLUSTER |
MIC_EFFECTIVE |
| zone1 | observer | 172.20.249.52 |     2882 | memory_limit | NULL      | 16G   | the size of the memory res
nal use(for testing purpose), 0 means follow memory_limit_percentage. Range: 0, [8G,) | OBSERVER | CLUSTER |
MIC_EFFECTIVE |
```

```
| zone3 | observer | 172.20.249.51 |     2882 | memory_limit | NULL      | 16G   | the size of the memory res
nal use(for testing purpose), 0 means follow memory_limit_percentage. Range: 0, [8G,) | OBSERVER | CLUSTER |
MIC_EFFECTIVE |
+-------+----------+---------------+----------+--------------+-----------+-------+--------------------------
-------------------------------------------------------------------------------------+----------+---------+-
-------------+
3 rows in set (0.005 sec)
```

4. 确认集群可用内存变化

```
select a.zone,concat(a.svr_ip,':',a.svr_port) observer, cpu_total, (cpu_total-cpu_assigned) cpu_
free, round(mem_total/1024/1024/1024) mem_total_gb, round((mem_total-mem_assigned)/1024/1024/102
4) mem_free_gb, usec_to_time(b.last_offline_time) last_offline_time, usec_to_time(b.start_servic
e_time) start_service_time, b.status, usec_to_time(b.stop_time) stop_time, b.build_version
from __all_virtual_server_stat a join __all_server b on (a.svr_ip=b.svr_ip and a.svr_port=b.svr_
port)
order by a.zone, a.svr_ip
;
```

输出:

```
+-------+--------------------+-----------+----------+--------------+-------------+--------------
--------------+----------------------------+--------+----------------------------+--------------
-------------------------------------------------------------+
| zone  | observer           | cpu_total | cpu_free | mem_total_gb | mem_free_gb | last_offline_
time          | start_service_time         | status | stop_time                  | build_versio
n                                                            |
+-------+--------------------+-----------+----------+--------------+-------------+--------------
--------------+----------------------------+--------+----------------------------+--------------
-------------------------------------------------------------+
| zone1 | 172.20.249.52:2882 |        14 |        0 |           13 |           8 | 1970-01-01 08
:00:00.000000 | 2021-09-25 08:19:06.622351 | active | 1970-01-01 08:00:00.000000 | 3.1.0_3-b2090
1e8c84d3ea774beeaca963c67d7802e4b4e(Aug 10 2021 08:10:38) |
| zone2 | 172.20.249.49:2882 |        14 |        0 |           13 |           8 | 1970-01-01 08
:00:00.000000 | 2021-09-25 08:19:07.392669 | active | 1970-01-01 08:00:00.000000 | 3.1.0_3-b2090
1e8c84d3ea774beeaca963c67d7802e4b4e(Aug 10 2021 08:10:38) |
| zone3 | 172.20.249.51:2882 |        14 |        0 |           13 |           8 | 1970-01-01 08
:00:00.000000 | 2021-09-26 14:05:58.641570 | active | 1970-01-01 08:00:00.000000 | 3.1.0_3-b2090
1e8c84d3ea774beeaca963c67d7802e4b4e(Aug 10 2021 08:10:38) |
+-------+--------------------+-----------+----------+--------------+-------------+--------------
--------------+----------------------------+--------+----------------------------+--------------
-------------------------------------------------------------+
3 rows in set (0.009 sec)
```

```
select t1.name resource_pool_name, t2.`name` unit_config_name, t2.max_cpu, t2.min_cpu, round(t2.
max_memory/1024/1024/1024) max_mem_gb, round(t2.min_memory/1024/1024/1024) min_mem_gb, t3.unit_i
d, t3.zone, concat(t3.svr_ip,':',t3.`svr_port`) observer,t4.tenant_id, t4.tenant_name
from __all_resource_pool t1 join __all_unit_config t2 on (t1.unit_config_id=t2.unit_config_id)
    join __all_unit t3 on (t1.`resource_pool_id` = t3.`resource_pool_id`)
    left join __all_tenant t4 on (t1.tenant_id=t4.tenant_id)
order by t1.`resource_pool_id`, t2.`unit_config_id`, t3.unit_id
;
```

输出:

```
+--------------------+------------------+---------+---------+------------+------------+---------
| resource_pool_name | unit_config_name | max_cpu | min_cpu | max_mem_gb | min_mem_gb | unit_id
+--------------------+------------------+---------+---------+------------+------------+---------
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       1
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       2
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       3
| my_pool            | unit1            |       9 |       9 |          3 |          3 |    1001
| my_pool            | unit1            |       9 |       9 |          3 |          3 |    1002
| my_pool            | unit1            |       9 |       9 |          3 |          3 |    1003
+--------------------+------------------+---------+---------+------------+------------+---------
6 rows in set (0.010 sec)
```

从输出结果可以看出新增可用内存 8G。

5. 调大业务租户内存规格

```
alter resource unit unit1 max_memory='11G',min_memory='11G';

# 重复查看集群可用内存

+-------+--------------------+-----------+----------+--------------+-------------+--------------
-------------+----------------------------+--------+----------------------------+--------------
------------------------------------------------------------+
| zone  | observer           | cpu_total | cpu_free | mem_total_gb | mem_free_gb | last_offline_
time          | start_service_time         | status | stop_time                  | build_versio
n                                                          |
+-------+--------------------+-----------+----------+--------------+-------------+--------------
-------------+----------------------------+--------+----------------------------+--------------
------------------------------------------------------------+
| zone1 | 172.20.249.52:2882 |        14 |        0 |           13 |           0 | 1970-01-01 08
:00:00.000000 | 2021-09-25 08:19:06.622351 | active | 1970-01-01 08:00:00.000000 | 3.1.0_3-b2090
1e8c84d3ea774beeaca963c67d7802e4b4e(Aug 10 2021 08:10:38) |
| zone2 | 172.20.249.49:2882 |        14 |        0 |           13 |           0 | 1970-01-01 08
:00:00.000000 | 2021-09-25 08:19:07.392669 | active | 1970-01-01 08:00:00.000000 | 3.1.0_3-b2090
1e8c84d3ea774beeaca963c67d7802e4b4e(Aug 10 2021 08:10:38) |
| zone3 | 172.20.249.51:2882 |        14 |        0 |           13 |           0 | 1970-01-01 08
:00:00.000000 | 2021-09-26 14:05:58.641570 | active | 1970-01-01 08:00:00.000000 | 3.1.0_3-b2090
1e8c84d3ea774beeaca963c67d7802e4b4e(Aug 10 2021 08:10:38) |
+-------+--------------------+-----------+----------+--------------+-------------+--------------
-------------+----------------------------+--------+----------------------------+--------------
------------------------------------------------------------+
3 rows in set (0.005 sec)
```

如何使用 OBD 对 OceanBase 集群或 OBProxy 集群扩容

使用 OBD 对 OceanBase 集群扩容主要是通过增加每个 ZONE 内的节点数来实现。前面已经介绍过集群扩容手动步骤，这里主要是介绍使用 OBD 工具进行扩容的操作步骤。

注意

OBD 的定位并不是运维工具，所以对于集群扩容节点并不是特别顺畅。

使用 OBD 对集群扩容可分为如下几步：

1. OBD 初始化新机器节点目录。
2. OBD 启动旧集群节点。
3. OceanBase 集群添加新节点

在进行扩容操作之前，您需先查看当前部署的集群信息。

```
[admin@obce00 ~]$ obd cluster display obce-3zones
Get local repositories and plugins ok
Open ssh connection ok
Cluster status check ok
Connect to observer ok
Wait for observer init ok
+-------------------------------------------------+
|                    observer                     |
+---------------+---------+------+-------+--------+
| ip            | version | port | zone  | status |
+---------------+---------+------+-------+--------+
| 172.20.249.49 | 3.1.0   | 2881 | zone2 | active |
| 172.20.249.51 | 3.1.0   | 2881 | zone3 | active |
| 172.20.249.52 | 3.1.0   | 2881 | zone1 | active |
+---------------+---------+------+-------+--------+

Connect to obproxy ok
+-------------------------------------------------+
|                    obproxy                      |
+---------------+------+-----------------+--------+
| ip            | port | prometheus_port | status |
+---------------+------+-----------------+--------+
| 172.20.249.52 | 2883 | 2884            | active |
| 172.20.249.49 | 2883 | 2884            | active |
| 172.20.249.51 | 2883 | 2884            | active |
+---------------+------+-----------------+--------+
[admin@obce00 ~]$ obd --version
OceanBase Deploy: 1.1.0
REVISION:
BUILD_BRANCH:
BUILD_TIME: Aug 10 2021 11:50:40OURCE
Copyright (C) 2021 OceanBase
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

OBD 初始化新机器节点目录

当给集群扩容节点时，建议每个 ZONE 新增相同配置的机器，但是不能直接在现有配置文件中编辑。当前版本（`1.1.0`）采取的方式是复制现有的配置文件为新文件，然后修改配置文件中的 `servers` 下的 IP 为新的 IP。如果 OBProxy 集群不扩容，就删除 OBProxy 配置节。

之后运行 `obd cluster deploy` 命令。`deploy` 命令会自动在新节点安装 OceanBase 软件包、初始化相关目录。

```
obd cluster deploy obce-3zones2 -c obce-3zones2.yaml
```

注意

deploy 成功后，不要运行 `obd cluster start` 命令，否则会初始化一个新的集群，不符合原本的目的。

OBD 启动旧集群节点

此时需要使用命令 `obd cluster edit-config` 编辑当前运行的集群的配置文件，在 `servers` 下加入新的节点 IP，新 IP 紧跟在旧的 IP 之后（注意换行）。

然后针对旧集群运行 `obd cluster start` 命令。运行该命令会启动集群中所有节点，如果节点的进程已经启动，就会跳过。所以，只会启动新增节点的进程。

```
obd cluster start obce-3zones
```

OceanBase 集群添加新节点

对于 OceanBase 集群扩容节点，还需要在集群 sys 租户里运行如下命令添加新的节点到相应 ZONE 里。

```
ALTER SYSTEM ADD SERVER '节点IP:RPC端口' ZONE '节点所属ZONE';
# 例如
alter system add server '11.166.87.5:2882' zone 'zone1';
```

运行以上命令后，查看节点状态可以看到新节点的状态是 `active`。

5.8 附录

A1. OBAgent 的常用指标的查询表达式

本文介绍 OBAgent 常用指标的查询表达式。有关 OBAgent 的具体介绍，请参考 OBAgent 文档。

常用指标在查询时，必须将变量替换成待查询的具体信息。需要替换的字段如下:

- @LABELS，您需替换为具体 Label 的过滤条件。
- @INTERVAL，您需替换为计算周期。
- @GBLABELS，您需替换为聚合的 Label 名称。

指标	表达式	单位
活跃 MEMStore 大小	`sum(ob_sysstat{stat_id="130000",@LABELS}) by (@GBLABELS) / 1048576`	MB
当前活跃会话数	`sum(ob_active_session_num{@LABELS}) by ([@GBLABELS)`	-
块缓存命中率	`100 * 1 / (1 + sum(rate(ob_sysstat{stat_id="50009",@LABELS}[@INTERVAL])) by ([@GBLABELS) ](/GBLABELS)) / sum(rate(ob_sysstat{stat_id="50008",@LABELS}[@INTERVAL])) by (@GBLABELS))`	%
块缓存大小	`sum(ob_cache_size_bytes{cache_name="user_block_cache",@LABELS}) by ([@GBLABELS) ](/GBLABELS)) / 1048576`	MB
每秒提交的事务日志大小	`sum(rate(ob_sysstat{stat_id="80057",@LABELS}[@INTERVAL])) by ([@GBLABELS) ](/GBLABELS))`	byte
每次事务日志写盘平均耗时	`sum(rate(ob_sysstat{stat_id="80041",@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_sysstat{stat_id="80040",@LABELS}[@INTERVAL])) by (@GBLABELS)`	us
CPU 使用率	`100 * (1 - sum(rate(node_cpu_seconds_total{mode="idle", @LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(node_cpu_seconds_total{@LABELS}[@INTERVAL])) by (@GBLABELS))`	%
磁盘分	`sum(node_filesystem_size_bytes{@LABELS}- node_filesystem_avail_bytes{@LABELS}) by (@GBLABELS) / 1073741824`	GB

区已使用容量		
每秒读次数	`avg(rate(node_disk_reads_completed_total{@LABELS}[@INTERVAL])) by (@GBLABELS)`	次/s
每次读取数据量	`avg(rate(node_disk_read_bytes_total{@LABELS}[@INTERVAL])) by (@GBLABELS) / 1048576`	MB
SSStore 每秒读次数	`sum(rate(ob_sysstat{stat_id="60000",@LABELS}[@INTERVAL])) by (@GBLABELS)`	次/s
SSStore 每次读取平均耗时	`sum(rate(ob_sysstat{stat_id="60001",@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_sysstat{stat_id="60000",@LABELS}[@INTERVAL])) by (@GBLABELS)`	us
SSStore 每秒读取数据量	`sum(rate(ob_sysstat{stat_id="60002",@LABELS}[@INTERVAL])) by (@GBLABELS)`	byte
平均每次读取耗时	`1000000 * (avg(rate(node_disk_read_time_seconds_total{@LABELS}[@INTERVAL])) by (@GBLABELS)) / (avg(rate(node_disk_reads_completed_total{@LABELS}[@INTERVAL])) by (@GBLABELS))`	us
平均每次 IO 读取耗时	`1000000 * (avg(rate(node_disk_read_time_seconds_total{@LABELS}[@INTERVAL])) by (@GBLABELS)) / (avg(rate(node_disk_reads_completed_total{@LABELS}[@INTERVAL])) by (@GBLABELS))`	us
每秒写次数	`avg(rate(node_disk_writes_completed_total{@LABELS}[@INTERVAL])) by (@GBLABELS)`	次/s
每次写入数据量	`avg(rate(node_disk_written_bytes_total{@LABELS}[@INTERVAL])) by (@GBLABELS) / 1048576`	MB
SSStore 每	`sum(rate(ob_sysstat{stat_id="60003",@LABELS}[@INTERVAL])) by (@GBLABELS)`	次/s

秒写次数		
SSStore 每次写入平均耗时	`sum(rate(ob_sysstat{stat_id="60004",@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_sysstat{stat_id="60003",@LABELS}[@INTERVAL])) by (@GBLABELS)`	us
SSStore 每秒写入数据量	`sum(rate(ob_sysstat{stat_id="60005",@LABELS}[@INTERVAL])) by (@GBLABELS)`	byte
平均每次写入耗时	`1000000 * (avg(rate(node_disk_write_time_seconds_total{@LABELS}[@INTERVAL])) by (@GBLABELS)) / (avg(rate(node_disk_writes_completed_total{@LABELS}[@INTERVAL])) by (@GBLABELS))`	us
平均每次 IO 写入耗时	`1000000 * (avg(rate(node_disk_write_time_seconds_total{@LABELS}[@INTERVAL])) by (@GBLABELS)) / (avg(rate(node_disk_writes_completed_total{@LABELS}[@INTERVAL])) by (@GBLABELS))`	us
过去 1 分钟系统平均负载	`avg(node_load1{@LABELS}) by (@GBLABELS)`	-
过去 15 分钟系统平均负载	`avg(node_load15{@LABELS}) by (@GBLABELS)`	-
过去 5 分钟系统平均负载	`avg(node_load5{@LABELS}) by (@GBLABELS)`	-
触发合并阈值	`sum(ob_sysstat{stat_id="130002",@LABELS}) by (@GBLABELS) / 1048576`	MB
内核 Buffe	`avg(node_memory_Buffers_bytes{@LABELS}) by (@GBLABELS) / 1073741824`	GB

r Cache 大小		
可用物理内存大小	avg(node_memory_MemFree_bytes{@LABELS}) by (@GBLABELS) / 1073741824	GB
使用物理内存大小	(avg(node_memory_MemTotal_bytes{@LABELS}) by (@GBLABELS) avg(node_memory_MemFree_bytes{@LABELS}) by (@GBLABELS) avg(node_memory_Cached_bytes{@LABELS}) by (@GBLABELS) avg(node_memory_Buffers_bytes{@LABELS}) by (@GBLABELS)) / 1073741824	GB
MEMStore 的上限	sum(ob_sysstat{stat_id="130004",@LABELS}) by (@GBLABELS) / 1048576	MB
MEMStore 使用百分比	100 * sum(ob_sysstat{stat_id="130001",@LABELS}) by (@GBLABELS) / sum(ob_sysstat{stat_id="130004",@LABELS}) by (@GBLABELS)	%
写锁等待失败次数	sum(rate(ob_sysstat{stat_id="60022",@LABELS}[@INTERVAL])) by (@GBLABELS)	次/s
写锁等待成功次数	sum(rate(ob_sysstat{stat_id="60021",@LABELS}[@INTERVAL])) by (@GBLABELS)	次/s
写锁平均等待耗时	sum(rate(ob_sysstat{stat_id="60023",@LABELS}[@INTERVAL])) by (@GBLABELS) / (sum(rate(ob_sysstat{stat_id="60021",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="60022",@LABELS}[@INTERVAL])) by (@GBLABELS))	us
每秒接收数据量	avg(rate(node_network_receive_bytes_total{@LABELS}[@INTERVAL])) by (@GBLABELS) / 1048576	MB
每秒发送数据量	avg(rate(node_network_transmit_bytes_total{@LABELS}[@INTERVAL])) by (@GBLABELS) / 1048576	MB
CPU 使用率	100 * sum(ob_sysstat{stat_id="140006",@LABELS}) by (@GBLABELS) / sum(ob_sysstat{stat_id="140005",@LABELS}) by (@GBLABELS)	%
	sum(ob_partition_num{@LABELS}) by (@GBLABELS)	

分区数量		-
执行计划缓存命中率	`100 * sum(rate(ob_plan_cache_hit_total{@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_plan_cache_access_total{@LABELS}[@INTERVAL])) by (@GBLABELS)`	%
执行计划缓存大小	`sum(ob_plan_cache_memory_bytes{@LABELS}) by (@GBLABELS) / 1048576`	MB
平均每秒SQL进等待队列的次数	`sum(rate(ob_sysstat{stat_id="20001",@LABELS}[@INTERVAL])) by (@GBLABELS)`	次/s
SQL在等待队列中等待耗时	`sum(rate(ob_sysstat{stat_id="20002",@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_sysstat{stat_id="20001",@LABELS}[@INTERVAL])) by (@GBLABELS)`	us
行缓存命中率	`100 * 1 / (1 + sum(rate(ob_sysstat{stat_id="50001",@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_sysstat{stat_id="50000",@LABELS}[@INTERVAL])) by (@GBLABELS))`	%
缓存大小	`sum(ob_cache_size_bytes{@LABELS}) by (@GBLABELS) / 1048576`	MB
RPC收包吞吐量	`sum(rate(ob_sysstat{stat_id="10001",@LABELS}[@INTERVAL])) by (@GBLABELS)`	byte
RPC收包平均耗时	`(sum(rate(ob_sysstat{stat_id="10005",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="10006",@LABELS}[@INTERVAL])) by (@GBLABELS)) / sum(rate(ob_sysstat{stat_id="10000",@LABELS}[@INTERVAL])) by (@GBLABELS)`	us
RPC发包吞吐量	`sum(rate(ob_sysstat{stat_id="10003",@LABELS}[@INTERVAL])) by (@GBLABELS)`	byte
RPC发包平均耗时	`(sum(rate(ob_sysstat{stat_id="10005",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="10006",@LABELS}[@INTERVAL])) by (@GBLABELS)) / sum(rate(ob_sysstat{stat_id="10002",@LABELS}[@INTERVAL])) by (@GBLABELS)`	us

每秒处理 SQL 语句数	`sum(rate(ob_sysstat{stat_id="40002",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="40004",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="40006",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="40008",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="40000",@LABELS}[@INTERVAL])) by (@GBLABELS)`	次/s
服务端每条 SQL 语句平均处理耗时	`(sum(rate(ob_sysstat{stat_id="40003",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="40005",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="40007",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="40009",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="40001",@LABELS}[@INTERVAL])) by (@GBLABELS)) / (sum(rate(ob_sysstat{stat_id="40002",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="40004",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="40006",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="40008",@LABELS}[@INTERVAL])) by (@GBLABELS) + sum(rate(ob_sysstat{stat_id="40000",@LABELS}[@INTERVAL])) by (@GBLABELS))`	us
每秒处理 Delete 语句数	`sum(rate(ob_sysstat{stat_id="40008",@LABELS}[@INTERVAL])) by (@GBLABELS)`	us
服务端每条 Delete 语句平均处理耗时	`sum(rate(ob_sysstat{stat_id="40009",@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_sysstat{stat_id="40008",@LABELS}[@INTERVAL])) by (@GBLABELS)`	us
每秒处理分布式执行计划数	`sum(rate(ob_sysstat{stat_id="40012",@LABELS}[@INTERVAL])) by (@GBLABELS)`	次/s
每秒处理 Insert 语句数	`sum(rate(ob_sysstat{stat_id="40002",@LABELS}[@INTERVAL])) by (@GBLABELS)`	次/s
服务端每条 Insert 语句平均	`sum(rate(ob_sysstat{stat_id="40003",@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_sysstat{stat_id="40002",@LABELS}[@INTERVAL])) by (@GBLABELS)`	us

处理耗时		
每秒处理本地执行数	`sum(rate(ob_sysstat{stat_id="40010",@LABELS}[@INTERVAL])) by (@GBLABELS)`	次/s
每秒处理远程执行计划数	`sum(rate(ob_sysstat{stat_id="40011",@LABELS}[@INTERVAL])) by (@GBLABELS)`	次/s
每秒处理 Replace 语句数	`sum(rate(ob_sysstat{stat_id="40004",@LABELS}[@INTERVAL])) by (@GBLABELS)`	次/s
服务端每条 Replace 语句平均处理耗时	`sum(rate(ob_sysstat{stat_id="40005",@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_sysstat{stat_id="40004",@LABELS}[@INTERVAL])) by (@GBLABELS)`	us
每秒处理 Select 语句数	`sum(rate(ob_sysstat{stat_id="40000",@LABELS}[@INTERVAL])) by (@GBLABELS)`	次/s
服务端每条 Select 语句平均处理耗时	`sum(rate(ob_sysstat{stat_id="40001",@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_sysstat{stat_id="40000",@LABELS}[@INTERVAL])) by (@GBLABELS)`	us
每秒处理 Update	`sum(rate(ob_sysstat{stat_id="40006",@LABELS}[@INTERVAL])) by (@GBLABELS)`	次/s

语句数		
服务端每条 Update 语句平均处理耗时	sum(rate(ob_sysstat{stat_id="40007",@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_sysstat{stat_id="40006",@LABELS}[@INTERVAL])) by (@GBLABELS)	us
表数量	max(ob_table_num{@LABELS}) by (@GBLABELS)	-
MEMStore 总大小	sum(ob_sysstat{stat_id="130001",@LABELS}) by (@GBLABELS) / 1048576	MB
每秒处理事务数	sum(rate(ob_sysstat{stat_id="30005",@LABELS}[@INTERVAL])) by (@GBLABELS)	次/s
服务端每个事务平均处理耗时	sum(rate(ob_sysstat{stat_id="30006",@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_sysstat{stat_id="30005",@LABELS}[@INTERVAL])) by (@GBLABELS)	us
每秒提交的事务日志数	sum(rate(ob_sysstat{stat_id="30002",@LABELS}[@INTERVAL])) by (@GBLABELS)	次/s
每次事务日志网络同步平均耗时	sum(rate(ob_sysstat{stat_id="30000",@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_sysstat{stat_id="30001",@LABELS}[@INTERVAL])) by (@GBLABELS)	us
每秒等待事件次数	sum(rate(ob_waitevent_wait_total{@LABELS}[@INTERVAL])) by (@GBLABELS)	次/s
等待事件平均耗时	sum(rate(ob_waitevent_wait_seconds_total{@LABELS}[@INTERVAL])) by (@GBLABELS) / sum(rate(ob_waitevent_wait_total{@LABELS}[@INTERVAL])) by (@GBLABELS)	s

第 6 章：测试 OceanBase 数据库

本章目录

6.1 性能测试概述

测试标准

测试 OceanBase 集群性能的方法有很多，如 `sysbench` 和 `benchmarksql` 等。如果想对比 OceanBase 数据库与传统数据库、其他分布式数据库产品的性能差异，需要找到一个能适用彼此的测试方案。

`sysbench` 的功能相对较弱，只有 0.4 版本支持 Oracle，并且不同版本的 SQL 并不完全一样。相比之下，`benchmarksql` 更为合适，`benchmarksql` 有一定的业务模型且标准简单，容易统一比较。

若您还想测试业务 SQL，可选择自己写程序或者使用 `JMeter`。

多副本同步

传统数据库或者基于 MySQL、PG 数据库的分布式数据库，通常都能以单副本实例运行。为了高可用和容灾，也支持一主多备的形态部署，主备之间的同步多是异步同步。

异步同步的优点是写性能好，风险是主副本实例故障时主备可能会不一致。即使使用了 MySQL 的半同步技术，也不能严格保证主备数据的强一致性。

OceanBase 数据库在生产环境中默认以三副本形态运行，主备副本之间通过事务日志来实现同步，同步协议使用 Paxos 强同步协议。OceanBase 数据库不支持异步同步，不支持牺牲数据安全换取性能。

故障应对能力

OceanBase 集群部署在普通商用服务器上，充分考虑了服务器的故障风险。所以 OceanBase 数据库的架构设计始终把数据安全放在首位，采用三副本强同步协议，在任意时刻发生故障时，OceanBase 数据库都能自动切换，选出新的主副本。

故障切换不需要 DBA 介入，数据库恢复时间（RTO）在 30s 左右，数据库恢复后保证数据不丢失（RPO = 0）。OceanBase 数据库的这个能力在平时很难看到，只有在发生故障时才能体现。所以，测试 OceanBase 数据库的性能时需要带着业务压力，并随机注入故障，这样才接近实际生产情况。

OceanBase 数据库的在线扩容和缩容、大表在线 DDL，都有高可用保障。节点故障后，OceanBase 数据库在恢复的同时，原有的任务都能自动继续进行。

6.2 影响性能的因素

磁盘划分

如果 OBServer 的运行日志、事务日志、数据文件都在一块盘上，则被称为单盘部署。单盘部署的风险较大，目前已知问题有如下几项:

- 事务日志（CLOG）空间利用率超过 80% 会开始回收，超过 95% 节点就会停止写入、掉线。
- 转储和合并时需要额外的 IO、CPU 资源，和业务读写的 IO 可能会互相影响。
- 运行日志（LOG）写的太多太快，可能会导致节点掉线，LOG 写满可能会导致节点异常。
- 影响 `liboblog` 同步，影响 CLOG 回收和副本迁移速度等。
- 参数文件 `observer.config.bin` 因为盘满导致持久化失败。

以上对 TPC-C 和 SYSBENCH 测试影响多一些，对 TPC-H 影响少一些。

应对策略:

- CLOG、LOG、SSTable 分多个独立文件系统。无论是多盘部署还是单盘部署，均使用 LVM 划分出多个 LV。
- 开启 LOG 限流，相关参数为：`syslog_io_bandwidth_limit`。

```
alter system set syslog_io_bandwidth_limit='10M';
```

- LOG 滚动输出，设置最大数目。

```
alter system set enable_syslog_recycle=true; alter system set max_syslog_file_count=50;
```

- CLOG 参数
  - `clog_disk_usage_limit_percentage` 默认为 95，不建议调大。
  - `clog_disk_utilization_threshold` 默认为 80，可以下调到 50，但不能太低。

OceanBase 和 OBProxy 参数

OceanBase 参数

```
alter system set net_thread_count = 32; -- 网络线程数
alter system set _clog_aggregation_buffer_amount=4;
alter system set _flush_clog_aggregation_buffer_timeout='1ms' ; -- 把大概几毫秒之内的日志都聚合到一个 rpc 中发送，减小网
```

```
alter system set trace_log_slow_query_watermark='10s' ;   --打印 SQL QUERY 的阈值
alter system set large_query_threshold='1s' ; -- large query 的阈值，超过后进入大查询队列,避免大查询阻塞小查询
alter system set syslog_level='PERF'; -- 控制日志输出级别
alter system set clog_sync_time_warn_threshold='2000ms' ;  -- clog 日志同步慢的时候触发 debug 日志的输出。
alter system set syslog_io_bandwidth_limit='10M' ;  -- 超过 10M 会日志限流，减小写日志文件带来的 IO 消耗
alter system set enable_sql_audit=false;   -- 关闭 sql audit
alter system set enable_perf_event=false;   -- 关闭监控项
alter system set clog_max_unconfirmed_log_count=5000;  -- 增大滑动窗口，解决 clog 滑动窗口满导致的性能问题
alter system set minor_warm_up_duration_time=0 ; -- 转储 SSTable 预热时间，转储完成后到预热时间内，所有对应 partition 的
alter system set memory_chunk_cache_size = '150G';  -- 降低 OceanBase 内部 2MB 内存块被 OS 回收的概率，增大 2MB
alter system set minor_merge_concurrency = 32;  -- 增大转储的线程数，期望提高转储的速度。
alter system set _mini_merge_concurrency = 8;  -- 增大 mini_merge 的线程数，期望提高 mini_merge 的速度
alter system set freeze_trigger_percentage = 40;  -- 触发转储的时机
alter system set autoinc_cache_refresh_interval='86400s';  -- 调大自增列刷新的频率，减少性能损耗。
alter system set cpu_quota_concurrency=2;  -- 这个数*租户cpu=工作线程数，具体调整的数值需要根据业务模型和机器配置调整，工作线程超过实际 CP
alter system set builtin_db_data_verify_cycle = 0 ; -- 宏块巡检周期参数，当设置为 0 时关闭巡检
alter system set micro_block_merge_verify_level = 0;  -- 微块校验等级设置，0：不做任何校验；1：对 encoding 做 decode
alter system set _ob_get_gts_ahead_interval = '0ms';
alter system set bf_cache_priority = 10;  -- 为频繁空查的宏块建立 bloomfilter 并缓存，减少磁盘 IO 和 CPU 消耗，提升写入性能
alter system set user_block_cache_priority=5;
alter system set merge_stat_sampling_ratio = 1 ;  -- 合并时统计信息采样率，当设置为 0 时则关闭统计信息采集
alter system set _enable_static_typing_engine=true;

alter system set enable_early_lock_release=false tenant=all; -- 提前解行锁场景下，用于租户级别控制，是否打开该优化
```

OBProxy 参数

```
alter proxyconfig set enable_ob_protocol_v2=false ;
alter proxyconfig set enable_qos=false ;
alter proxyconfig set enable_compression_protocol=false ;
alter proxyconfig set automatic_match_work_thread = false;  -- 关闭自动计算线程个数
alter proxyconfig set work_thread_num = 32;  -- 手动设置工作线程个数，需要 restart
alter proxyconfig set syslog_level='ERROR';
alter proxyconfig set monitor_log_level = 'ERROR';
alter proxyconfig set enable_monitor_stat = false ;
alter proxyconfig set xflush_log_level="ERROR";
```

JDBC URL 参数

```
conn=jdbc:mysql://x.x.x.x(ip):xx(port)/xxxx(dbname)?rewriteBatchedStatements=true&allowMultiQueries=true&useLocalSessionState=true&useUnicode=true&characterEncoding=utf-8&socketTimeout=3000000
```

- rewriteBatchedStatements：此参数影响导数据效率，不可以忽略。
- useLocalSessionState：是否使用 `autocommit`、`read_only` 和 `transaction isolation` 的内部值（jdbc 端的本

地值），建议设置为 `true`。如果设置为 `false`，则需要发语句到远端请求，增加发送请求频次将会影响性能。

转储与合并

转储是租户级别的操作，当一台服务器的一个租户中 MEMTable 的内存占用超过租户内存上限的一定比例时，就会触发小版本冻结（Minor Freeze）。

所谓冻结，就是禁止当前活跃 MEMTable 的写入，并同时创建新的空 MEMTable 以支持新的写入。被冻结后的 MEMTable 就成为一个静态的数据结构，可以被后台线程读取并转储为 SSTable。

合并是全局级别的操作，合并会把当前大版本的 SSTable 和 MEMTable 与前一个大版本的全量静态数据进行合并，产生新的全量数据。

> **说明**
> 合并操作包含转储操作。

转储和合并是把内存数据刷到磁盘上，存储层统计信息可以更准确，生成的执行计划也就更稳定准确。
MEMTable 扫描性能很差，合并后 RANGE 查询性能会有提升。

PRIMARY_ZONE 设置

租户的 PRIMARY_ZONE 用来控制租户里主副本的分布。如果设置为 RANDOM，就可以将不同分区的主副本分散在不同 zone 的节点上，机器利用率将达到最大化。反过来，设置为某个具体的 zone，则只能利用部分节点。

使用分区表

分区表使用水平拆分策略将一个大表切割为多个独立的分区。

> **说明**
> 独立的分区并不是分表，依然是分区表的一部分。

分区表有以下作用：

- 将海量数据分散到不同分区存储，不同分区可以分散在不同节点存储，从而化解单机空间瓶颈。
- 将海量请求分散到不同分区，不同分区主副本可以分散在不同节点，从而化解单机处理能力瓶颈。

分区表的索引分为全局索引和本地索引：

- 全局索引对某些不带分区键的查询比较友好，但是对写不友好，可能导致分布式事务。此外，当 RANGE 查询要返回很多数据时，全局索引也会有很多分布式查询，性能并不一定比本地索引好。
- 使用本地索引读写数据，可以规避分布式事务，可以多分区多节点并行读。

使用表分组 Table Group

表分组是将一组业务联系紧密的表的分区分布调度在一起。

- `table_group`： Table Group 是一个逻辑概念，它和物理数据文件没有关联关系，Table Group 只影响表分区的调度方法，OceanBase 数据库会优先把属于同一个 Table Group 的相同分区号的分区调度到同一台节点上，以减少跨节点分布式事务。
- `partition_group`：Table Group 的每个分区表中下标相同的一组分区为 Partition Group，作用是将分区号相同的 partition 放到 1 个 Partition Group 的物理结构里面达到提升性能的目的。

注意

同一个表分组里面的分区表的分区策略必须一致。

6.3 如何运行 Sysbench 测试

Sysbench 是一个支持多线程的跨平台模块化基准测试工具，可以执行 CPU、内存、线程、IO、数据库等方面的性能测试，用于评估系统在运行高负载的数据库时相关核心参数的性能表现。

使用 Sysbench 可以绕过复杂的数据库基准设置，甚至在没有安装数据库的前提下，快速了解数据库系统的性能。

Sysbench 可以做多种性能测试。本文主要介绍对数据库性能（OLTP）的测试。

Sysbench 安装

编译安装

1. 下载 Sysbench：https://github.com/akopytov/sysbench/releases/tag/1.0.20
2. 解压缩后，编译安装。

```
cd sysbench-1.0.20
# yum -y install automake libtool
# yum -y install mysql.x86_64 mysql-devel.x86_64 --allowerasing

# ./autogen.sh
# ./configure --prefix=/usr/sysbench/ --with-mysql-includes=/usr/include/mysql/ --with-mysql-lib
s=/usr/lib64/mysql/ --with-mysql

# make
# make install
# cp /usr/sysbench/bin/sysbench /usr/local/bin/

# cd /usr/sysbench/
# bin/sysbench --help
```

3. 查看帮助命令，验证 Sysbench 是否安装成功。

```
[root@obce-0000 ~]# cd /usr/sysbench/
[root@obce-0000 sysbench]# bin/sysbench --help
Usage:
  sysbench [options]... [testname] [command]

Commands implemented by most tests: prepare run cleanup help

General options:
  --threads=N                     number of threads to use [1]
  --events=N                      limit for total number of events [0]
  --time=N                        limit for total execution time in seconds [10]
  --forced-shutdown=STRING        number of seconds to wait after the --time limit before forcin
g shutdown, or 'off' to disable [off]
  --thread-stack-size=SIZE        size of stack per thread [64K]
```

```
  --rate=N                        average transactions rate. 0 for unlimited rate [0]
  --report-interval=N             periodically report intermediate statistics with a specified i
nterval in seconds. 0 disables intermediate reports [0]
  --report-checkpoints=[LIST,...] dump full statistics and reset all counters at specified point
s in time. The argument is a list of comma-separated values representing the amount of time in s
econds elapsed from start of test when report checkpoint(s) must be performed. Report checkpoint
s are off by default. []
  --debug[=on|off]                print more debugging info [off]
  --validate[=on|off]             perform validation checks where possible [off]
  --help[=on|off]                 print help and exit [off]
  --version[=on|off]              print version and exit [off]
  --config-file=FILENAME          File containing command line options
  --tx-rate=N                     deprecated alias for --rate [0]
  --max-requests=N                deprecated alias for --events [0]
  --max-time=N                    deprecated alias for --time [0]
  --num-threads=N                 deprecated alias for --threads [1]

Pseudo-Random Numbers Generator options:
  --rand-type=STRING random numbers distribution {uniform,gaussian,special,pareto} [special]
  --rand-spec-iter=N number of iterations used for numbers generation [12]
  --rand-spec-pct=N  percentage of values to be treated as 'special' (for special distribution)
[1]
  --rand-spec-res=N  percentage of 'special' values to use (for special distribution) [75]
  --rand-seed=N      seed for random number generator. When 0, the current time is used as a RN
G seed. [0]
  --rand-pareto-h=N  parameter h for pareto distribution [0.2]

Log options:
  --verbosity=N verbosity level {5 - debug, 0 - only critical messages} [3]

  --percentile=N       percentile to calculate in latency statistics (1-100). Use the special va
lue of 0 to disable percentile calculations [95]
  --histogram[=on|off] print latency histogram in report [off]

General database options:

  --db-driver=STRING  specifies database driver to use ('help' to get list of available drivers
) [mysql]
  --db-ps-mode=STRING prepared statements usage mode {auto, disable} [auto]
  --db-debug[=on|off] print database-specific debug information [off]


Compiled-in database drivers:
  mysql - MySQL driver

mysql options:
  --mysql-host=[LIST,...]          MySQL server host [localhost]
  --mysql-port=[LIST,...]          MySQL server port [3306]
  --mysql-socket=[LIST,...]        MySQL socket
  --mysql-user=STRING              MySQL user [sbtest]
  --mysql-password=STRING          MySQL password []
  --mysql-db=STRING                MySQL database name [sbtest]
  --mysql-ssl[=on|off]             use SSL connections, if available in the client library [off]
  --mysql-ssl-cipher=STRING        use specific cipher for SSL connections []
  --mysql-compression[=on|off]     use compression, if available in the client library [off]
  --mysql-debug[=on|off]           trace all client library calls [off]
  --mysql-ignore-errors=[LIST,...] list of errors to ignore, or "all" [1213,1020,1205]
  --mysql-dry-run[=on|off]         Dry run, pretend that all MySQL client API calls are successf
```

```
ul without executing them [off]

Compiled-in tests:
  fileio - File I/O test
  cpu - CPU performance test
  memory - Memory functions speed test
  threads - Threads subsystem performance test
  mutex - Mutex performance test

See 'sysbench <testname> help' for a list of options for each test.
```

常见安装问题

您在安装过程中可能遇到的报错如下:

- `automake 1.10.x (aclocal) wasn't found, exiting`

  原因: 操作系统没有安装 `automake`。

  解决办法: 运行命令 `yum install automake.noarch`, 即可安装 `automake`。

- `libtoolize 1.4+ wasn't found, exiting`

  原因: 操作系统没有安装 `libtool`。

  解决办法: 运行命令 `yum install libtool`, 即可安装 `libtool`。

- `drv_mysql.c:35:19: fatal error: mysql.h: No such file or directory`

  原因: 操作系统没有安装 MySQL 的开发 lib 库。

  解决办法: 运行命令 `yum install mysql-devel`, 即可安装 MySQL 的开发 lib 库。

- `error while loading shared libraries: libmysqlclient_r.so.16`

  该错误通常在直接下载编译好的文件的情况下碰到。

  解决办法: 将下载文件中的 `libmysqlclient_r.so.16.0.0` 复制到目录 `/usr/lib64/mysql` 中, 并做一个软链接。

```
sudo cp libmysqlclient_r.so.16.0.0 /usr/lib64/mysql
sudo ln -s /usr/lib64/mysql/libmysqlclient_r.so.16.0.0 /usr/lib64/mysql/libmysqlclient_r.so.16
sudo ldconfig
```

测试准备

参数设置

OS 参数

```
sudo sh -c 'for x in /sys/class/net/eth0/queues/rx-*; do echo ff>$x/rps_cpus; done'
sudo sh -c "echo 32768 > /proc/sys/net/core/rps_sock_flow_entries"
sudo sh -c "echo 4096 > /sys/class/net/eth0/queues/rx-0/rps_flow_cnt"
```

说明

`ffffffff` 表示使用 32 个核，您可根据实际配置进行修改。例如：ECS 为 8 核，则输入 `ff`。

OceanBase 参数

- sys 租户参数

```
alter system set enable_auto_leader_switch=false;
alter system set enable_one_phase_commit=false;
alter system set enable_monotonic_weak_read = true;
alter system set weak_read_version_refresh_interval='5s';
alter system _ob_minor_merge_schedule_interval='5s';

alter system set memory_limit_percentage = 90;    --  OceanBase 数据库占系统总内存的比例，提高 OceanB
ase 数据库可用的
alter system set memstore_limit_percentage = 55;  --  memstore 占租户的内存比，尽量增大 memstore 的空
间（但是可能对读操
alter system set freeze_trigger_percentage = 70;  --  启动 major/minor freeze 的时机，让转储（mino
r freeze
alter system set minor_freeze_times = 50;        --  minor freeze 的次数，尽量不在测试期间触发 major
freeze
alter system set minor_warm_up_duration_time = 0;  --  加快 minor freeze

alter system set merge_thread_count = 32;  --  增大合并的线程数
alter system set minor_merge_concurrency = 8;  --  增大转储的线程数，期望提高转储的速度
alter system set _mini_merge_concurrency = 4;  --  增大 mini_merge 的线程数，期望提高 mini_merge 的
速度（默认值为 3
```

- Proxy 参数

```
alter proxyconfig set proxy_mem_limited='4G'; --防止 oom，可根据实际环境动态调整
alter proxyconfig set enable_compression_protocol=false; --关闭压缩，降低 CPU 百分率
alter proxyconfig set work_thread_num=32; -- 调整工作线程数，寻找最优性能
alter proxyconfig set enable_compression_protocol=false;
alter proxyconfig set enable_metadb_used=false;
alter proxyconfig set enable_standby=false;
alter proxyconfig set enable_strict_stat_time=false;
alter proxyconfig set use_local_dbconfig=true;
```

- 租户参数

```
数据库下租户设置，防止事务超时
set global ob_timestamp_service='GTS' ;
set global autocommit=ON;
set global ob_query_timeout=36000000000;
set global ob_trx_timeout=36000000000;
set global max_allowed_packet=67108864;
```

```
set global ob_sql_work_area_percentage=100;
set global parallel_max_servers=800;
set global parallel_servers_target=800;
```

数据准备

- 准备数据库账户

```
create database sysbenchdb;
grant all privileges on sysbenchdb.* to u_sysbench identified by '*****';
```

- （可选）修改建表语句

  建表语句存放在 `/usr/sysbench/share/sysbench/oltp_common.lua` 中。

```
cd /usr/sysbench/
vim share/sysbench/oltp_common.lua +150
150 function create_table(drv, con, table_num)
```

  若您想要创建分区表，需要修改这里的脚本。分区表需要选择一个分区键，并且主键要包含分区键。

- 初始化表和数据

  Sysbench 参数说明：

  - `--tables`：指定表的数量。
  - `--table_size`：指定表的数据量。
  - `--threads`：指定并发数。
  - `--mysql-ignore-errors`：指定忽略的错误号，忽略后测试继续运行。否则，测试报错中断。
  - `--time`：指定运行时间。
  - `--report-interval`：报告间隔。

  示例：

```
sysbench --test=./oltp_read_only.lua --mysql-host=172.xx.xxx.54 --mysql-port=2883 --mysql-db=sys
benchdb --mysql-user="obdemo:test:sysbench"  --mysql-password=***** --tables=12 --table_size=100
00000 --threads=12 --time=300 --report-interval=3 --db-driver=mysql --db-ps-mode=disable --skip-
trx=on --mysql-ignore-errors=6002,6004,4012,2013,4016,1062 prepare
```

  输出：

```
WARNING: the --test option is deprecated. You can pass a script name or path on the command lin
e without any options.
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)
```

```
Initializing worker threads...

Creating table 'sbtest3'...
Creating table 'sbtest1'...
Creating table 'sbtest6'...
Creating table 'sbtest7'...
Creating table 'sbtest5'...
Creating table 'sbtest2'...
Creating table 'sbtest8'...
Creating table 'sbtest4'...
Inserting 1000000 records into 'sbtest3'
Inserting 1000000 records into 'sbtest1'
Inserting 1000000 records into 'sbtest6'
Inserting 1000000 records into 'sbtest5'
Inserting 1000000 records into 'sbtest7'
Inserting 1000000 records into 'sbtest8'
Inserting 1000000 records into 'sbtest4'
Inserting 1000000 records into 'sbtest2'
Creating a secondary index on 'sbtest6'...
Creating a secondary index on 'sbtest8'...
Creating a secondary index on 'sbtest3'...
Creating a secondary index on 'sbtest5'...
Creating a secondary index on 'sbtest4'...
Creating a secondary index on 'sbtest1'...
Creating a secondary index on 'sbtest7'...
Creating a secondary index on 'sbtest2'...
Creating table 'sbtest9'...
Inserting 1000000 records into 'sbtest9'
Creating a secondary index on 'sbtest9'...
[root@obce-0000 sysbench]#
```

常见初始化问题:

`FATAL: mysql_drv_query() returned error 4030 (Over tenant memory limits) for query 'INSERT INTO sbtest5`

这是典型的由于增量内存消耗速度高于转储速度而造成的问题。如果租户内存很小，写入的并发就不能太高。

初始化配置供参考:

```
+--------------------+------------------+---------+---------+------------+------------+---------
| resource_pool_name | unit_config_name | max_cpu | min_cpu | max_mem_gb | min_mem_gb | unit_id
+--------------------+------------------+---------+---------+------------+------------+---------
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       1
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       2
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       3
| my_pool_zone1      | unit1            |       9 |       9 |         19 |         19 |    1001
| my_pool_zone2      | unit1            |       9 |       9 |         19 |         19 |    1002
| my_pool_zone3      | unit1            |       9 |       9 |         19 |         19 |    1003
+--------------------+------------------+---------+---------+------------+------------+---------
6 rows in set (0.042 sec)

alter system set merge_thread_count = 32;  --  增大合并的线程数。
alter system set minor_merge_concurrency = 8;  --  增大转储的线程数，期望提高转储的速度。
alter system set _mini_merge_concurrency = 4;  --  增大 mini_merge 的线程数，期望提高 mini_merge 的
```

```
速度（默认值为 3

alter system set memory_limit = '24G';    --  OceanBase 占系统总内存的比例，提高 OceanBase 可用的内存量。
alter system set memstore_limit_percentage = 55;  --  memstore 占租户的内存比，尽量增大 memstore 的空间（但是可能对读操
alter system set freeze_trigger_percentage = 70;  --  启动 major/minor freeze 的时机，让转储（mino
r freeze
alter system set minor_freeze_times = 50;        --  minor freeze 的次数，尽量不在测试期间触发 major freeze。
alter system set minor_warm_up_duration_time = 0;  --  加快 minor freeze
```

场景测试

不同场景需使用不同的 LUA 脚本文件。

```
[root@obce-0000 sysbench]# pwd
/usr/sysbench/share/sysbench
[root@obce-0000 sysbench]# ls *.lua
bulk_insert.lua  oltp_delete.lua  oltp_point_select.lua  oltp_read_write.lua    oltp_update_non_index.
lua  select_random_points.lua
oltp_common.lua  oltp_insert.lua  oltp_read_only.lua     oltp_update_index.lua  oltp_write_only.lu
a        select_random_ranges.lua
```

复制初始化脚本命令，替换场景文件名，`prepare` 改为 `run`。

Sysbench 的场景测试，建议同一个场景下设置不同的并发运行测试，然后用 EXCEL 记录相应的性能值，连成曲线，观察线性扩展关系。

- 只读测试

  只读测试的 SQL 都是读。

  OceanBase 数据库的读默认是强一致性读，只读取数据分区的主副本。如果要观察 OceanBase 数据库的读写分离能力，可以修改这个测试脚本，在读 SQL 里增加弱一致性读 Hint（`/*+ read_consistency(weak) */`）即可随机读取备副本。

  但是，OceanBase 数据库的弱一致性读备副本还受参数 `max_stale_time_for_weak_consistency` 限制。如果备副本延迟时间超出这个参数定义（默认 5s），则备副本不提供读服务。

  ```
  sysbench --test=./oltp_read_only.lua --mysql-host=172.xx.xxx.52 --mysql-port=2883 --mysql-db=sys
  benchdb --mysql-user="obdemo:test:sysbench"  --mysql-password=***** --tables=9 --table_size=1000
  000 --threads=8 --time=300 --report-interval=60 --db-driver=mysql --db-ps-mode=disable --skip-tr
  x=on --mysql-ignore-errors=6002,6004,4012,2013,4016,1062 run
  ```

  输出:

  ```
  WARNING: the --test option is deprecated. You can pass a script name or path on the command lin
  e without any options.
  sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)
  ```

```
Running the test with following options:
Number of threads: 8
Report intermediate results every 60 second(s)
Initializing random number generator from current time


Initializing worker threads...

Threads started!

[ 60s ] thds: 8 tps: 114.73 qps: 1607.23 (r/w/o: 1607.23/0.00/0.00) lat (ms,95%): 179.94 err/s:
0.00 reconn/s: 0.00
[ 120s ] thds: 8 tps: 79.95 qps: 1119.26 (r/w/o: 1119.26/0.00/0.00) lat (ms,95%): 262.64 err/s:
0.00 reconn/s: 0.00
[ 180s ] thds: 8 tps: 97.37 qps: 1362.75 (r/w/o: 1362.75/0.00/0.00) lat (ms,95%): 215.44 err/s:
0.00 reconn/s: 0.00
[ 240s ] thds: 8 tps: 109.87 qps: 1538.27 (r/w/o: 1538.27/0.00/0.00) lat (ms,95%): 193.38 err/s
: 0.00 reconn/s: 0.00
[ 300s ] thds: 8 tps: 193.03 qps: 2702.82 (r/w/o: 2702.82/0.00/0.00) lat (ms,95%): 95.81 err/s:
0.00 reconn/s: 0.00
SQL statistics:
    queries performed:
        read:                            499870
        write:                           0
        other:                           0
        total:                           499870
    transactions:                        35705  (118.99 per sec.)
    queries:                             499870 (1665.81 per sec.)
    ignored errors:                      0      (0.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          300.0755s
    total number of events:              35705

Latency (ms):
         min:                                    11.09
         avg:                                    67.22
         max:                                  1347.51
         95th percentile:                       179.94
         sum:                                2400071.63

Threads fairness:
    events (avg/stddev):           4463.1250/52.63
    execution time (avg/stddev):   300.0090/0.03
```

- 纯写场景

  纯写场景包含 `insert`、`update` 和 `delete` SQL。如果表是分区表，大概率会有跨节点的分布式事务。

  OceanBase 数据库的分布式事务是强一致性模型，所以在并发不高节点规模不大时，相比传统数据库，这个测试性能会相对差一些。随着数据量的增长，并发的提升以及机器规模的增长，OceanBase 数据库的多活和线性扩展优势会逐步体现出来。

```
sysbench --test=./oltp_write_only.lua --mysql-host=172.xx.xxx.52 --mysql-port=2883 --mysql-db=sy
sbenchdb --mysql-user="obdemo:test:sysbench"  --mysql-password=***** --tables=9 --table_size=100
```

```
0000 --threads=8 --time=300 --report-interval=60 --db-driver=mysql --db-ps-mode=disable --skip-t
rx=on --mysql-ignore-errors=6002,6004,4012,2013,4016 run
```

- 读写混合场景

```
sysbench --test=./oltp_read_write.lua --mysql-host=172.xx.xxx.52 --mysql-port=2883 --mysql-db=sy
sbenchdb --mysql-user="obdemo:test:sysbench"  --mysql-password=***** --tables=9 --table_size=100
0000 --threads=8 --time=300 --report-interval=60 --db-driver=mysql --db-ps-mode=disable --skip-t
rx=on --mysql-ignore-errors=6002,6004,4012,2013,4016,1062 run
```

性能调优经验

转储与合并

对 Sysbench 来说（1000000*30 行表），Sysbench 一般数据量较小，导完数据后不会发生转储，此时做合并有消极影响。

例如：合并转储将 `memtable` 刷到硬盘，多了一个 `major sstable`，从硬盘操作数据肯定比在 `memtable` 要慢，将会导致性能下降。

对比项	合并转储前（qps/rt）	转储后（qps/rt）	合并后（qps/rt）
point_select	367240.41/7.70	332106.87/8.28	342172.34/8.43
write_only	137125.60/81.48	138654.65/84.47	135978.23/84.47
3 千万行单表 range 查询	18.601s	10.853s	10.709s

调整 PRIMARY_ZONE

对比项	集中式部署（zone1）	random 部署
sysbench(point_select)	261505.51/11.87	373923.98/7.56
sysbench(read_write)	81818.40/297.92	125315.97/427.07

6.4 如何跑 TPC-C 测试

TPC（Transaction Processing Performance Council，事务处理性能委员会）是由数十家会员公司创建的非盈利组织，总部设在美国。TPC 的成员主要是计算机软硬件厂家，而非计算机用户，其功能是制定商务应用基准程序的标准规范、性能和价格度量，并管理测试结果的发布。

TPC-C 是 TPC 推出的一系列性能测试标准中的一款，自 1992 年推出，便成为了数据库性能测试的标杆，各个数据库大厂都向 TPC 委员会提交了测试结果，以期望在 TPC-C 测试的排行榜上能拥有一席之地。

2019 年之前，TPC-C 榜首一直是 Oracle 公司，持续 8 年多。2019 年 9 月底，OceanBase 以分布式数据库身份参与 TPC-C 评测，夺得榜首。2020 年 5 月，OceanBase 再次参与 TPC-C 评测，刷新了之前的成绩（提升了 10 倍）。

OceanBase 第一次参加 TPC-C 使用的是云服务器，规模 204 台。第二次也是云服务器，规模 1500 台左右。这是由于 TPC-C 的标准定义非常严格，对数据量、业务读写行为要求都很具体，导致机器规模数很大，普通的企业测试不能采用这个标准。业界参照 TPC-C 的业务模型和标准有类似的测试程序开源，如 BenchmarkSQL。

TPC-C 简介

数据库模型

在测试开始前，TPC-C Benchmark 规定了数据库的初始状态（也就是数据库中数据生成的规则），其中 ITEM 表中固定包含 10 万种商品，仓库的数量可进行调整，假设 WAREHOUSE 表中有 W 条记录，那么：

- STOCK 表中应有 W×10 万条记录（每个仓库对应 10 万种商品的库存数据）；
- DISTRICT 表中应有 W×10 条记录（每个仓库为 10 个地区提供服务）；
- CUSTOMER 表中应有 W×10×3000 条记录（每个地区有 3000 个客户）；
- HISTORY 表中应有 W×10×3000 条记录（每个客户一条交易历史）；
- ORDER 表中应有 W×10×3000 条记录（每个地区 3000 个订单），并且最后生成的 900 个订单将被添加到 NEW-ORDER 表中，每个订单随机生成 5~15 条 ORDER-LINE 记录。

在测试过程中，每一个地区（DISTRICT）都有一个对应的终端（Terminal），模拟为用户提供服务。在每个终端的生命周期内，要循环往复地执行各类事务，当终端执行完一个事务的周期后，就进入下一个事务的周期。

客户下单后，包含若干个订单明细（ORDER-LINE）的订单（ORDER）被生成，并被加入新订单（NEW-ORDER）列表。

客户支付订单会产生交易历史（HISTORY）。 每个订单（ORDER）平均包含 10 条订单项（ORDER-LINE），其中 1% 需要从远程仓库中获取，这些就是 TPC-C 模型中的 9 个数据表。其中，仓库的数量 W 可以根据系统的实际情况进行调整，以使系统性能测试结果达到最佳。

事务类型

该 benchmark 包含 5 类事务:

- NewOrder:新订单请求

  从某一仓库中随机选取 5 ~ 15 件商品,创建新订单。其中 1% 的事务需要回滚(即 err)。一般地,新订单请求不可能超出全部事务请求的 45%。

- Payment:订单付款

  更新客户账户余额,反映其支付情况。在全部事务请求中占比 43%。

- OrderStatus:最近订单查询

  随机选择一个用户,查询其最近一条订单,显示该订单内的每个商品状态。在全部事务请求中占比 4%。

- Delivery:配送

  模拟批处理交易,更新该订单用户的余额,把发货单从 NewOrder 中删除。在全部事务请求中占比 4%。

- StockLevel:库存

  缺货状态分析,在全部事务请求中占比 4%。

软件准备

BenchmarkSQL 下载

BenchmarkSQL 是开源项目,官方下载地址为:https://sourceforge.net/projects/benchmarksql/。

为节省测试时间,OceanBase 团队对这个程序进行了修改,导入数据报错时,只针对报错的仓库进行补充加载,而不是所有数据重新导入。该版本可以向 OceanBase 技术支持人员获取,或者直接访问链接(https://github.com/obpilot/benchmarksql-5.0)下载。

下载后可直接使用。

> **注意** 需要有 Java 运行环境,且版本不低于 V1.8.0。

```
[root@obce-0000 bmsql_vivid]# pwd
/root/bmsql_vivid
[root@obce-0000 bmsql_vivid]# tree -L 1
.
├── build
├── build.xml
├── dist
├── doc
├── HOW-TO-RUN.txt
├── lib
├── README.md
```

```
├── run
└── src

6 directories, 3 files
[root@obce-0000 bmsql_vivid]#
```

配置文件

配置文件 `props.ob` 在 `run` 目录下。

```
db=oracle
driver=com.alipay.oceanbase.jdbc.Driver
// conn=jdbc:oceanbase://122.xx.xxx.125:2883/icbc?useUnicode=true&characterEncoding=utf-8
conn=jdbc:oceanbase://172.xx.xxx.52:2883/tpccdb?useUnicode=true&characterEncoding=utf-8&rewriteBatchedS
tatements=true&allowMultiQueries=true
user=tpcc@test#obdemo
password=******

warehouses=10
loadWorkers=2
//fileLocation=/data/temp/

terminals=10
//To run specified transactions per terminal- runMins must equal zero
runTxnsPerTerminal=0
//To run for specified minutes- runTxnsPerTerminal must equal zero
runMins=10
//Number of total transactions per minute
limitTxnsPerMin=0

//Set to true to run in 4.x compatible mode. Set to false to use the
//entire configured database evenly.
terminalWarehouseFixed=true

//The following five values must add up to 100
newOrderWeight=45
paymentWeight=43
orderStatusWeight=4
deliveryWeight=4
stockLevelWeight=4

// Directory name to create for collecting detailed result data.
// Comment this out to suppress.
resultDirectory=my_result_%tY-%tm-%td_%tH%tM%tS
osCollectorScript=./misc/os_collector_linux.py
osCollectorInterval=1
//osCollectorSSHAddr=user@dbhost
//osCollectorDevices=net_eth0 blk_sda

//LoadStartW=1
//LoadStopW=1000
```

说明:

- `db`：指定数据库类型。

  这里复用 OceanBase 的类型。后面 OceanBase 数据库相应的 LIBRARY 放到对应目录下（`lib/`）。

- `warehouses`：指定仓库数。

  仓库数决定性能测试的成绩。如果期较高有望测试结果，仓库数就不能太低。生产环境机器测试，建议 5000 仓库起步。如果机器配置较差，建议 100 仓起步。

- `loadWorkers`：指定仓库数据加载时的并发。

  如果机器配置很高，该值可以设置大一些，比如说 100 个。 如果机器配置不高（尤其是内存），该值需要设置小一些，如 10 个并发。并发指定得过高，可能导致内存消耗太快，出现报错，导致数据加载前功尽弃。

  > **说明**
  >
  > 第一次使用时，建议并发设置低一些，宁可慢一点，也不要报错。

- `terminals`：指定性能压测时的并发数。

  建议并发数不要高于仓库数 * 10。否则，会有不必要的锁等待。在生产环境中，该并发数设置到 1000 就很高了。一般环境测试建议从 100 开始。

- `runMins`：指定性能测试持续的时间。

  时间越久，越能考验数据库的性能和稳定性。建议不要少于 10 分钟，生产环境中机器建议不少于 1 小时。

- `LoadStartW` 和 `LoadStopW`：指定补仓时的开始值和截止值。

  如果导数据时发现某个仓库数据导入失败（大事务超时），您可以指定这个仓库重新导入。

库文件

OceanBase 数据库提供自己的驱动文件 `oceanbase-client-x.x.x.jar` 文件。无论是测试 MySQL 租户还是 Oracle 租户，都可以使用这个驱动文件。 驱动文件的读取是脚本 `run/func.sh` 里的逻辑。

```
# ----
# getCP()
#
#   Determine the CLASSPATH based on the database system.
# ----
function setCP()
{
    case "$(getProp db)" in
        firebird)
            cp="../lib/firebird/*:../lib/*"
            ;;
        oracle)
            cp="../lib/oracle/*"
            if [ ! -z "${ORACLE_HOME}" -a -d ${ORACLE_HOME}/lib ] ; then
                cp="${cp}:${ORACLE_HOME}/lib/*"
            fi
            cp="${cp}:../lib/*"
```

```
            ;;
        postgres)
            cp="../lib/postgres/*:../lib/*"
            ;;
        oceanbase)
            # cp="../lib/oceanbase/oceanbase-client-1.0.1.jar:../lib/oceanbase/guava-parent-18.0-site.
jar:../lib/*"
            cp="../lib/oceanbase/*:../lib/*"
            ;;
    esac
    myCP=".:${cp}:../dist/*"
    export myCP
}
```

因此把 OceanBase 数据库的驱动放到 `lib/` 目录或 `lib/oracle` 下即可。

```
[root@obce-0000 bmsql_vivid]# tree lib/
lib/
├── apache-log4j-extras-1.1.jar
├── firebird
│   ├── connector-api-1.5.jar
│   └── jaybird-2.2.9.jar
├── gsjdbc4.jar
├── log4j-1.2.17.jar
├── mysql-connector-java-5.1.47.jar
├── oceanbase
│   ├── commons-lang-2.3.jar
│   ├── guava-18.0.jar
│   ├── json-20160810.jar
│   ├── oceanbaseclient1.1.10.jar
│   └── toolkit-common-logging-1.10.jar
├── oracle
│   ├── oceanbaseclient1.1.10.jar
│   └── README.txt
└── postgres
    └── postgresql-9.3-1102.jdbc41.jar

4 directories, 14 files
```

数据准备

建表

建表脚本通常放在 `run/sql.common` 下或者其他指定目录下。建表脚本可以选择非分区表方案和分区表方案。

- 非分区表

```
[root@obce-0000 run]# sh runSQL.sh props.ob sql.common/tableCreates.sql
# ------------------------------------------------------------
# Loading SQL file sql.common/tableCreates.sql
# ------------------------------------------------------------
```

```
.:../lib/oracle/*:../lib/*:../dist/*
-Dprop=props.ob -DcommandFile=sql.common/tableCreates.sql
create table bmsql_config (
cfg_name    varchar(30) primary key,
cfg_value   varchar(50)
);
create tablegroup tpcc_group ;
create table bmsql_warehouse (
w_id        integer   not null,
w_ytd       decimal(12,2),
w_tax       decimal(4,4),
w_name      varchar(10),
w_street_1  varchar(20),
w_street_2  varchar(20),
w_city      varchar(20),
w_state     char(2),
w_zip       char(9),
primary key(w_id)
)tablegroup=tpcc_group;
create table bmsql_district (
d_w_id       integer       not null,
d_id         integer       not null,
d_ytd        decimal(12,2),
d_tax        decimal(4,4),
d_next_o_id  integer,
d_name       varchar(10),
d_street_1   varchar(20),
d_street_2   varchar(20),
d_city       varchar(20),
d_state      char(2),
d_zip        char(9),
PRIMARY KEY (d_w_id, d_id)
)tablegroup=tpcc_group ;
create table bmsql_customer (
c_w_id         integer        not null,
c_d_id         integer        not null,
c_id           integer        not null,
c_discount     decimal(4,4),
c_credit       char(2),
c_last         varchar(16),
c_first        varchar(16),
c_credit_lim   decimal(12,2),
c_balance      decimal(12,2),
c_ytd_payment  decimal(12,2),
c_payment_cnt  integer,
c_delivery_cnt integer,
c_street_1     varchar(20),
c_street_2     varchar(20),
c_city         varchar(20),
c_state        char(2),
c_zip          char(9),
c_phone        char(16),
c_since        timestamp,
c_middle       char(2),
c_data         varchar(500),
PRIMARY KEY (c_w_id, c_d_id, c_id)
)tablegroup=tpcc_group ;
```

```
create table bmsql_history (
hist_id  integer,
h_c_id   integer,
h_c_d_id integer,
h_c_w_id integer,
h_d_id   integer,
h_w_id   integer,
h_date   timestamp,
h_amount decimal(6,2),
h_data   varchar(24)
)tablegroup=tpcc_group ;
create table bmsql_new_order (
no_w_id  integer   not null ,
no_d_id  integer   not null,
[detached from 1248.pts-0.obce-0000]
[root@obce-0000 run]# cat sql.common/tableCreates.sql
create table bmsql_config (
cfg_name    varchar(30) primary key,
cfg_value   varchar(50)
);

create tablegroup tpcc_group ;

create table bmsql_warehouse (
w_id        integer   not null,
w_ytd       decimal(12,2),
w_tax       decimal(4,4),
w_name      varchar(10),
w_street_1  varchar(20),
w_street_2  varchar(20),
w_city      varchar(20),
w_state     char(2),
w_zip       char(9),
primary key(w_id)
)tablegroup=tpcc_group;

create table bmsql_district (
d_w_id       integer       not null,
d_id         integer       not null,
d_ytd        decimal(12,2),
d_tax        decimal(4,4),
d_next_o_id  integer,
d_name       varchar(10),
d_street_1   varchar(20),
d_street_2   varchar(20),
d_city       varchar(20),
d_state      char(2),
d_zip        char(9),
PRIMARY KEY (d_w_id, d_id)
)tablegroup=tpcc_group ;

create table bmsql_customer (
c_w_id         integer        not null,
c_d_id         integer        not null,
c_id           integer        not null,
c_discount     decimal(4,4),
c_credit       char(2),
```

```
c_last         varchar(16),
c_first        varchar(16),
c_credit_lim   decimal(12,2),
c_balance      decimal(12,2),
c_ytd_payment  decimal(12,2),
c_payment_cnt  integer,
c_delivery_cnt integer,
c_street_1     varchar(20),
c_street_2     varchar(20),
c_city         varchar(20),
c_state        char(2),
c_zip          char(9),
c_phone        char(16),
c_since        timestamp,
c_middle       char(2),
c_data         varchar(500),
PRIMARY KEY (c_w_id, c_d_id, c_id)
)tablegroup=tpcc_group ;


create table bmsql_history (
hist_id  integer,
h_c_id   integer,
h_c_d_id integer,
h_c_w_id integer,
h_d_id   integer,
h_w_id   integer,
h_date   timestamp,
h_amount decimal(6,2),
h_data   varchar(24)
)tablegroup=tpcc_group ;

create table bmsql_new_order (
no_w_id  integer   not null ,
no_d_id  integer   not null,
no_o_id  integer   not null,
PRIMARY KEY (no_w_id, no_d_id, no_o_id)
)tablegroup=tpcc_group ;

create table bmsql_oorder (
o_w_id       integer      not null,
o_d_id       integer      not null,
o_id         integer      not null,
o_c_id       integer,
o_carrier_id integer,
o_ol_cnt     integer,
o_all_local  integer,
o_entry_d    timestamp,
PRIMARY KEY (o_w_id, o_d_id, o_id)
)tablegroup=tpcc_group ;

create table bmsql_order_line (
ol_w_id         integer   not null,
ol_d_id         integer   not null,
ol_o_id         integer   not null,
ol_number       integer   not null,
ol_i_id         integer   not null,
```

```
ol_delivery_d   timestamp,
ol_amount       decimal(6,2),
ol_supply_w_id  integer,
ol_quantity     integer,
ol_dist_info    char(24),
PRIMARY KEY (ol_w_id, ol_d_id, ol_o_id, ol_number)
)tablegroup=tpcc_group ;

create table bmsql_item (
i_id     integer      not null,
i_name   varchar(24),
i_price  decimal(5,2),
i_data   varchar(50),
i_im_id  integer,
PRIMARY KEY (i_id)
)tablegroup=tpcc_group;

create table bmsql_stock (
s_w_id       integer       not null,
s_i_id       integer       not null,
s_quantity   integer,
s_ytd        integer,
s_order_cnt  integer,
s_remote_cnt integer,
s_data       varchar(50),
s_dist_01    char(24),
s_dist_02    char(24),
s_dist_03    char(24),
s_dist_04    char(24),
s_dist_05    char(24),
s_dist_06    char(24),
s_dist_07    char(24),
s_dist_08    char(24),
s_dist_09    char(24),
s_dist_10    char(24),
PRIMARY KEY (s_w_id, s_i_id)
)tablegroup=tpcc_group;
```

- 分区表语法

  分区表是一种水平拆分方案，大部分表按照仓库 ID 做 HASH 分区。分区数取决于要测试的数据规模和机器数。

  如果只有三台机器，分区数以 3 ~ 9 个为宜。如果是 5000 仓，9 台机器，则分区数可以调整到 99 或 100。通常来说 HASH 分区数没必要过 100。

```
[root@obce-0000 run]# cat sql.common/tableCreates_parts.sql
create table bmsql_config (
  cfg_name    varchar(30) primary key,
  cfg_value   varchar(50)
);

-- drop tablegroup tpcc_group;
create tablegroup tpcc_group partition by hash partitions 3;
```

```
create table bmsql_warehouse (
  w_id        integer   not null,
  w_ytd       decimal(12,2),
  w_tax       decimal(4,4),
  w_name      varchar(10),
  w_street_1  varchar(20),
  w_street_2  varchar(20),
  w_city      varchar(20),
  w_state     char(2),
  w_zip       char(9),
  primary key(w_id)
)tablegroup='tpcc_group' partition by hash(w_id) partitions 3;

create table bmsql_district (
  d_w_id       integer       not null,
  d_id         integer       not null,
  d_ytd        decimal(12,2),
  d_tax        decimal(4,4),
  d_next_o_id  integer,
  d_name       varchar(10),
  d_street_1   varchar(20),
  d_street_2   varchar(20),
  d_city       varchar(20),
  d_state      char(2),
  d_zip        char(9),
  PRIMARY KEY (d_w_id, d_id)
)tablegroup='tpcc_group' partition by hash(d_w_id) partitions 3;

create table bmsql_customer (
  c_w_id         integer        not null,
  c_d_id         integer        not null,
  c_id           integer        not null,
  c_discount     decimal(4,4),
  c_credit       char(2),
  c_last         varchar(16),
  c_first        varchar(16),
  c_credit_lim   decimal(12,2),
  c_balance      decimal(12,2),
  c_ytd_payment  decimal(12,2),
  c_payment_cnt  integer,
  c_delivery_cnt integer,
  c_street_1     varchar(20),
  c_street_2     varchar(20),
  c_city         varchar(20),
  c_state        char(2),
  c_zip          char(9),
  c_phone        char(16),
  c_since        timestamp,
  c_middle       char(2),
  c_data         varchar(500),
  PRIMARY KEY (c_w_id, c_d_id, c_id)
)tablegroup='tpcc_group' partition by hash(c_w_id) partitions 3;


create table bmsql_history (
  hist_id  integer,
  h_c_id   integer,
```

```
  h_c_d_id integer,
  h_c_w_id integer,
  h_d_id   integer,
  h_w_id   integer,
  h_date   timestamp,
  h_amount decimal(6,2),
  h_data   varchar(24)
)tablegroup='tpcc_group' partition by hash(h_w_id) partitions 3;

create table bmsql_new_order (
  no_w_id  integer   not null ,
  no_d_id  integer   not null,
  no_o_id  integer   not null,
  PRIMARY KEY (no_w_id, no_d_id, no_o_id)
)tablegroup='tpcc_group' partition by hash(no_w_id) partitions 3;

create table bmsql_oorder (
  o_w_id       integer      not null,
  o_d_id       integer      not null,
  o_id         integer      not null,
  o_c_id       integer,
  o_carrier_id integer,
  o_ol_cnt     integer,
  o_all_local  integer,
  o_entry_d    timestamp,
  PRIMARY KEY (o_w_id, o_d_id, o_id)
)tablegroup='tpcc_group' partition by hash(o_w_id) partitions 3;

create table bmsql_order_line (
  ol_w_id         integer   not null,
  ol_d_id         integer   not null,
  ol_o_id         integer   not null,
  ol_number       integer   not null,
  ol_i_id         integer   not null,
  ol_delivery_d   timestamp,
  ol_amount       decimal(6,2),
  ol_supply_w_id  integer,
  ol_quantity     integer,
  ol_dist_info    char(24),
  PRIMARY KEY (ol_w_id, ol_d_id, ol_o_id, ol_number)
)tablegroup='tpcc_group' partition by hash(ol_w_id) partitions 3;

create table bmsql_item (
  i_id     integer      not null,
  i_name   varchar(24),
  i_price  decimal(5,2),
  i_data   varchar(50),
  i_im_id  integer,
  PRIMARY KEY (i_id)
) duplicate_scope='cluster';

create table bmsql_stock (
  s_w_id       integer       not null,
  s_i_id       integer       not null,
  s_quantity   integer,
  s_ytd        integer,
  s_order_cnt  integer,
```

```
  s_remote_cnt integer,
  s_data       varchar(50),
  s_dist_01    char(24),
  s_dist_02    char(24),
  s_dist_03    char(24),
  s_dist_04    char(24),
  s_dist_05    char(24),
  s_dist_06    char(24),
  s_dist_07    char(24),
  s_dist_08    char(24),
  s_dist_09    char(24),
  s_dist_10    char(24),
  PRIMARY KEY (s_w_id, s_i_id)
)tablegroup='tpcc_group' use_bloom_filter=true partition by hash(s_w_id) partitions 3;
```

运行如下命令建表。

```
sh runSQL.sh props.ob sql.common/tableCreates_parts.sql

输出:
[root@obce-0000 run]# sh runSQL.sh props.ob sql.common/tableCreates_parts.sql
# ------------------------------------------------------------
# Loading SQL file sql.common/tableCreates_parts.sql
# ------------------------------------------------------------
.:../lib/oracle/*:../lib/*:../dist/*
-Dprop=props.ob -DcommandFile=sql.common/tableCreates_parts.sql

<.....>
```

加载数据

加载数据即数据初始化，仓库数越多加载时间越长。1000 仓可能要近 1 个小时，5000 仓可能要半天以上。具体性能取决于机器配置。

```
sh runLoader.sh props.ob

输出:
[root@obce-0000 run]# sh runLoader.sh props.ob
Starting BenchmarkSQL LoadData

driver=com.alipay.oceanbase.jdbc.Driver
conn=jdbc:oceanbase://172.xx.xxx.52:2883/tpccdb?useUnicode=true&characterEncoding=utf-8&rewriteBatchedS
tatements=true&allowMultiQueries=true
user=tpcc@test#obdemo
password=***********
warehouses=10
loadWorkers=2
fileLocation (not defined)
csvNullValue (not defined - using default 'NULL')
LoadStartW (not defined)
LoadStopW (not defined)

Worker 000: Loading ITEM
```

```
Worker 001: Loading Warehouse      1
Worker 000: Loading ITEM done
Worker 000: Loading Warehouse      2

<.....>
```

加载数据的 INSERT SQL 使用了 `batch insert` 特性，这点是在 `props.ob` 里的 JDBC URL 里指定的。该特性的写入性能最好。

创建索引

当数据初始化完后再补充两个索引。

```
[root@obce-0000 run]# cat sql.common/indexCreates.sql
create index bmsql_customer_idx1
  on  bmsql_customer (c_w_id, c_d_id, c_last, c_first) local;
create  index bmsql_oorder_idx1
  on  bmsql_oorder (o_w_id, o_d_id, o_carrier_id, o_id) local;
```

(可选)删除表

删除表就是删除所有的表和表分组，在需要修改表结构分区数的时候执行。

```
[root@obce-0000 run]# cat sql.common/tableDrops.sql
drop table bmsql_config;
drop table bmsql_new_order;
drop table bmsql_order_line;
drop table bmsql_oorder;
drop table bmsql_history;
drop table bmsql_customer;
drop table bmsql_stock;
drop table bmsql_item;
drop table bmsql_district;
drop table bmsql_warehouse;

purge recyclebin;

-- tpcc_group
drop tablegroup tpcc_group;
```

运行命令:

```
sh runSQL.sh props.ob sql.common/tableDrops.sql

输出:
[root@obce-0000 run]# sh runSQL.sh props.ob sql.common/tableDrops.sql
# ------------------------------------------------------------
# Loading SQL file sql.common/tableDrops.sql
# ------------------------------------------------------------
.:../lib/oracle/*:../lib/*:../dist/*
```

```
-Dprop=props.ob -DcommandFile=sql.common/tableDrops.sql
drop table bmsql_config;
pdrop table bmsql_new_order;
wdrop table bmsql_order_line;
drop table bmsql_oorder;
ddrop table bmsql_history;

drop table bmsql_customer;
drop table bmsql_stock;
drop table bmsql_item;
drop table bmsql_district;
drop table bmsql_warehouse;
purge recyclebin;
-- tpcc_group
drop tablegroup tpcc_group;
purge recyclebin;
```

性能测试

数据初始化没有错误（没有仓库数据报错）即可跑性能测试。建议跑性能测试之前先做一次集群合并（`major freeze`）。

```
sh runBenchmark.sh props.ob

输出:
[root@obce-0000 run]# sh runBenchmark.sh props.ob
10:21:44,894 [main] INFO   jTPCC : Term-00,
10:21:44,896 [main] INFO   jTPCC : Term-00, +-------------------------------------------------------
----+
10:21:44,896 [main] INFO   jTPCC : Term-00,      BenchmarkSQL v5.0
10:21:44,896 [main] INFO   jTPCC : Term-00, +-------------------------------------------------------
----+
10:21:44,896 [main] INFO   jTPCC : Term-00,  (c) 2003, Raul Barbosa
10:21:44,896 [main] INFO   jTPCC : Term-00,  (c) 2004-2016, Denis Lussier
10:21:44,898 [main] INFO   jTPCC : Term-00,  (c) 2016, Jan Wieck
10:21:44,898 [main] INFO   jTPCC : Term-00, +-------------------------------------------------------
----+
10:21:44,898 [main] INFO   jTPCC : Term-00,
10:21:44,898 [main] INFO   jTPCC : Term-00, db=oracle
10:21:44,898 [main] INFO   jTPCC : Term-00, driver=com.alipay.oceanbase.jdbc.Driver
10:21:44,898 [main] INFO   jTPCC : Term-00, conn=jdbc:oceanbase://172.xx.xxx.52:2883/tpccdb?useUnicode=
true&characterEncoding=utf-8&rewriteBatchedStatements=true&allowMultiQueries=true
10:21:44,898 [main] INFO   jTPCC : Term-00, user=tpcc@test#obdemo
10:21:44,898 [main] INFO   jTPCC : Term-00,
10:21:44,899 [main] INFO   jTPCC : Term-00, warehouses=10
10:21:44,899 [main] INFO   jTPCC : Term-00, terminals=10
10:21:44,900 [main] INFO   jTPCC : Term-00, runMins=10
10:21:44,900 [main] INFO   jTPCC : Term-00, limitTxnsPerMin=0
10:21:44,900 [main] INFO   jTPCC : Term-00, terminalWarehouseFixed=true
10:21:44,900 [main] INFO   jTPCC : Term-00,
10:21:44,900 [main] INFO   jTPCC : Term-00, newOrderWeight=45
10:21:44,900 [main] INFO   jTPCC : Term-00, paymentWeight=43
10:21:44,900 [main] INFO   jTPCC : Term-00, orderStatusWeight=4
10:21:44,901 [main] INFO   jTPCC : Term-00, deliveryWeight=4
```

```
10:21:44,901 [main] INFO   jTPCC : Term-00, stockLevelWeight=4
10:21:44,901 [main] INFO   jTPCC : Term-00,
10:21:44,901 [main] INFO   jTPCC : Term-00, resultDirectory=my_result_%tY-%tm-%td_%tH%tM%tS
10:21:44,901 [main] INFO   jTPCC : Term-00, osCollectorScript=./misc/os_collector_linux.py
10:21:44,901 [main] INFO   jTPCC : Term-00,
10:21:44,925 [main] INFO   jTPCC : Term-00, copied props.ob to my_result_2021-10-01_102144/run.propert
ies
10:21:44,926 [main] INFO   jTPCC : Term-00, created my_result_2021-10-01_102144/data/runInfo.csv for r
unID 326
10:21:44,926 [main] INFO   jTPCC : Term-00, writing per transaction results to my_result_2021-10-01_10
2144/data/result.csv
10:21:44,926 [main] INFO   jTPCC : Term-00, osCollectorScript=./misc/os_collector_linux.py
10:21:44,926 [main] INFO   jTPCC : Term-00, osCollectorInterval=1
10:21:44,927 [main] INFO   jTPCC : Term-00, osCollectorSSHAddr=null
10:21:44,927 [main] INFO   jTPCC : Term-00, osCollectorDevices=null
10:21:45,031 [main] INFO   jTPCC : Term-00,
10:21:45,611 [main] INFO   jTPCC : Term-00, C value for C_LAST during load: 144
10:21:45,613 [main] INFO   jTPCC : Term-00, C value for C_LAST this run:    215
10:21:45,614 [main] INFO   jTPCC : Term-00,

0:21:45,611 [main] INFO   jTPCC : Term-00, C value for C_LAST during load: 14Term-00, Running Average
tpmTOTAL: 5376.32    Current tpmTOTAL: 356448    Memory Usage: 44MB / 366MB                         10:
31:46,707 [Thread-7] INFO   jTPCC : Term-00
,
                                                    10:31:46,707 [Thread-7] INFO   jTPCC : Term-00
,
                                                    10:31:46,707 [Thread-7] INFO   jTPCC : Term-00, M
easured tpmC (NewOrders) = 2410.28
10:31:46,708 [Thread-7] INFO   jTPCC : Term-00, Measured tpmTOTAL = 5376.33
10:31:46,708 [Thread-7] INFO   jTPCC : Term-00, Session Start     = 2021-10-01 10:21:46
10:31:46,708 [Thread-7] INFO   jTPCC : Term-00, Session End       = 2021-10-01 10:31:46
10:31:46,708 [Thread-7] INFO   jTPCC : Term-00, Transaction Count = 53776
[root@obce-0000 run]#
```

TPC-C 用 `tpmC` 值（Transactions per Minute）来衡量系统最大有效吞吐量。其中 Transactions 以 NewOrder Transaction 为准，即最终衡量单位为每分钟处理的订单数。

性能调优

性能测试的结果取决于租户资源大小、内存转储的设置、客户端的性能（JVM 的大小）等等。

转储与合并

对比项	合并转储前	合并后	提升率
bmsql（1000 仓、600 并发 tpmc）	230211.65	264401.03	11.30%

调整 PRIMARY_ZONE

对比项	集中式部署（zone1）	random 部署

bmsql	88592.24	264401.03

使用 TABLEGROUP

使用 `table_group` 可以减少分布式查询和事务，提升性能。

对比项	未使用 pg	使用 pg
bmsql	60383.72	250249.36

常见问题

- `Could not find the main class: ExecJDBC. Program will exit.`

  解决方法:

  - 升级 JDK
  - 升级 JDBC 版本
  - 确认 oceanbase-client-x.x.x.jar 是否在指定路径下

- `Invalid number of terminals!`
  - 原因：这种情况是由 `my_oracle.properties` 中设置的 `terminals` 值不对而造成的。
  - 解决方法：填写正确范围内的 `terminals` 值 `numTerminals <= 0 || numTerminals > 10*numWarehouses`。

- 事务超时报错 `transaction timeout`

  解决方法：需增大超时时间，建议为 `36000000000`。

6.5 如何跑 TPC-H 测试

TPC-H 简介

TPC-H 基准测试是由 TPC-D（由 TPC 组织于 1994 年制定的标准，用于决策支持系统方面的测试基准）发展而来的。TPC-H 用 3NF 实现了一个数据仓库，共包含 8 个基本关系。TPC-H 的主要评价指标是各个查询的响应时间，即从提交查询到结果返回所需时间。

TPC-H 基准测试的度量单位是每小时执行的查询数（QphH@size），其中 `H` 表示每小时系统执行复杂查询的平均次数，`size` 表示数据库规模的大小，它能够反映出系统处理查询的能力。

TPC-H 是根据真实的生产运行环境来建模的，这使得它可以评估一些其他测试所不能评估的关键性能参数。

总而言之，TPC 组织颁布的 TPC-H 标准满足了数据仓库领域的测试需求，并且促使各个厂商以及研究机构将该项技术推向极限。详细信息请参考 http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.3.pdf。

测试准备

软件安装

- 下载软件

  软件由 TPC-H Tool 官方提供，下载地址为：http://tpc.org/TPC_Documents_Current_Versions/download_programs/tools-download-request5.asp?bm_type=TPC-H&bm_vers=3.0.0&mode=CURRENT-ONLY。

  文件名：`TPC-H_Tools_v3.0.0.zip`

```
cd /root/TPC-H_Tools_v3.0.0/dbgen
cp makefile.suite Makefile
```

- 修改 `Makefile` 文件中的 `CC`、`DATABASE`、`MACHINE` 和 `WORKLOAD` 等参数定义。

```
vim Makefile +103
CC      = gcc
# Current values for DATABASE are: INFORMIX, DB2, TDAT (Teradata)
#                                  SQLSERVER, SYBASE, ORACLE, VECTORWISE
# Current values for MACHINE are:  ATT, DOS, HP, IBM, ICL, MVS,
#                                  SGI, SUN, U2200, VMS, LINUX, WIN32
# Current values for WORKLOAD are:  TPCH
DATABASE= MYSQL
MACHINE = LINUX
WORKLOAD = TPCH
```

- 修改 `tpcd.h` 文件，并添加新的宏定义。

```
vim tpcd.h +$
#ifdef MYSQL
#define GEN_QUERY_PLAN ""
#define START_TRAN "START TRANSACTION"
#define END_TRAN "COMMIT"
#define SET_OUTPUT ""
#define SET_ROWCOUNT "limit %d;\n"
#define SET_DBASE "use %s;\n"
#endif
```

- 对文件进行编译。

```
make
```

编译完成后该目录下会生成以下两个可执行文件:

- dbgen：数据生成工具。在使用 InfiniDB 官方测试脚本进行测试时，需要用该工具生成 tpch 相关表数据。
- qgen：SQL 生成工具。生成初始化测试查询，由于不同的 seed 生成的查询不同，为了结果的可重复性，请使用附件提供的 22 个查询。

数据文件准备

`dbgen` 命令可以生成指定大小的数据，生成环境测试建议不少于 1000G 。本文以 `10G` 为例。

```
./dbgen -s 10

输出:

TPC-H Population Generator (Version 3.0.0)
Copyright Transaction Processing Performance Council 1994 - 2010

mkdir tpch10
mv *.tbl tpch10/

[root@obce-0000 dbgen]# ls -lrth tpch10/
总用量 11G
-rw-r--r-- 1 root root  14M 10月  1 16:04 supplier.tbl
-rw-r--r-- 1 root root  389 10月  1 16:04 region.tbl
-rw-r--r-- 1 root root 233M 10月  1 16:04 part.tbl
-rw-r--r-- 1 root root 1.2G 10月  1 16:04 partsupp.tbl
-rw-r--r-- 1 root root 1.7G 10月  1 16:04 orders.tbl
-rw-r--r-- 1 root root 2.2K 10月  1 16:04 nation.tbl
-rw-r--r-- 1 root root 7.3G 10月  1 16:04 lineitem.tbl
-rw-r--r-- 1 root root 234M 10月  1 16:04 customer.tbl
```

查询语句准备

```
cp qgen queries/
cp dists.dss queries/
cd queries/

for i in `seq 22`; do echo $i; ./qgen -d $i -s 10 > db"$i".sql; done

dos2unix *.sql
```

去掉生成的 SQL 文件中的 limit -xx（OceanBase 不支持 limit 负数的语法），去掉 day 后面的 `(3)`，并且加上 `parallel(96)` 并发，以 `db1.sql` 为例。

- SQL 1

```
select /*+   TPCH_Q1 parallel(96) */      ---增加 parallel 并发执行
        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
        sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
from
        lineitem
where
        l_shipdate <= date '1998-12-01' - interval '90' day (3)     ---去掉(3)
group by
        l_returnflag,
        l_linestatus
order by
        l_returnflag,
        l_linestatus;
limit -1;              ---   去掉这行
```

- SQL 2

```
select /*+   TPCH_Q2 parallel(96) */
        s_acctbal,
        s_name,
        n_name,
        p_partkey,
        p_mfgr,
        s_address,
        s_phone,
        s_comment
from
        part,
        supplier,
        partsupp,
        nation,
        region
where
```

```
        p_partkey = ps_partkey
        and s_suppkey = ps_suppkey
        and p_size = 15
        and p_type like '%BRASS'
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = 'EUROPE'
        and ps_supplycost = (
                select
                        min(ps_supplycost)
                from
                        partsupp,
                        supplier,
                        nation,
                        region
                where
                        p_partkey = ps_partkey
                        and s_suppkey = ps_suppkey
                        and s_nationkey = n_nationkey
                        and n_regionkey = r_regionkey
                        and r_name = 'EUROPE'
        )
order by
        s_acctbal desc,
        n_name,
        s_name,
        p_partkey;
```

其他 SQL 不再重复说明。

参数准备

租户参数

```
set global autocommit=ON;
set global ob_query_timeout=36000000000;
set global ob_trx_timeout=36000000000;
set global max_allowed_packet=67108864;
set global ob_sql_work_area_percentage=80;
/*
parallel_max_servers 和 parallel_servers_target 的值
推荐设置为测试租户分配的 resource unit cpu 数的 10 倍
如测试租户使用的 unit 配置为: create resource unit $unit_name max_cpu 26
那么该值设置为 260
*/
set global parallel_max_servers=260;
set global parallel_servers_target=260;
```

建表

```
create tablegroup tpch_tg_10g_lineitem_order_group binding true partition by key 1 partitions 9;
create tablegroup tpch_tg_10g_partsupp_part binding true partition by key 1 partitions 9;
```

```
drop database if exists tpch_10g_part;
create database tpch_10g_part;
use tpch_10g_part;

CREATE TABLE lineitem (
    l_orderkey bigint NOT NULL,
    l_partkey bigint NOT NULL,
    l_suppkey bigint NOT NULL,
    l_linenumber bigint NOT NULL,
    l_quantity bigint NOT NULL,
    l_extendedprice bigint NOT NULL,
    l_discount bigint NOT NULL,
    l_tax bigint NOT NULL,
    l_returnflag char(1) DEFAULT NULL,
    l_linestatus char(1) DEFAULT NULL,
    l_shipdate date NOT NULL,
    l_commitdate date DEFAULT NULL,
    l_receiptdate date DEFAULT NULL,
    l_shipinstruct char(25) DEFAULT NULL,
    l_shipmode char(10) DEFAULT NULL,
    l_comment varchar(44) DEFAULT NULL,
    primary key(l_orderkey, l_linenumber)
)    tablegroup = tpch_tg_10g_lineitem_order_group
    partition by key (l_orderkey) partitions 9;
create index I_L_ORDERKEY on lineitem(l_orderkey) local;
create index I_L_SHIPDATE on lineitem(l_shipdate) local;

CREATE TABLE orders (
    o_orderkey bigint NOT NULL,
    o_custkey bigint NOT NULL,
    o_orderstatus char(1) DEFAULT NULL,
    o_totalprice bigint DEFAULT NULL,
    o_orderdate date NOT NULL,
    o_orderpriority char(15) DEFAULT NULL,
    o_clerk char(15) DEFAULT NULL,
    o_shippriority bigint DEFAULT NULL,
    o_comment varchar(79) DEFAULT NULL,
    PRIMARY KEY (o_orderkey))
    tablegroup = tpch_tg_10g_lineitem_order_group
    partition by key(o_orderkey) partitions 9;
    create index I_O_ORDERDATE on orders(o_orderdate) local;


CREATE TABLE partsupp (
    ps_partkey bigint NOT NULL,
    ps_suppkey bigint NOT NULL,
    ps_availqty bigint DEFAULT NULL,
    ps_supplycost bigint DEFAULT NULL,
    ps_comment varchar(199) DEFAULT NULL,
    PRIMARY KEY (ps_partkey, ps_suppkey))
    tablegroup tpch_tg_10g_partsupp_part
    partition by key(ps_partkey) partitions 9;


CREATE TABLE part (
  p_partkey bigint NOT NULL,
```

```
  p_name varchar(55) DEFAULT NULL,
  p_mfgr char(25) DEFAULT NULL,
  p_brand char(10) DEFAULT NULL,
  p_type varchar(25) DEFAULT NULL,
  p_size bigint DEFAULT NULL,
  p_container char(10) DEFAULT NULL,
  p_retailprice bigint DEFAULT NULL,
  p_comment varchar(23) DEFAULT NULL,
  PRIMARY KEY (p_partkey))
  tablegroup tpch_tg_10g_partsupp_part
  partition by key(p_partkey) partitions 9;

CREATE TABLE customer (
  c_custkey bigint NOT NULL,
  c_name varchar(25) DEFAULT NULL,
  c_address varchar(40) DEFAULT NULL,
  c_nationkey bigint DEFAULT NULL,
  c_phone char(15) DEFAULT NULL,
  c_acctbal bigint DEFAULT NULL,
  c_mktsegment char(10) DEFAULT NULL,
  c_comment varchar(117) DEFAULT NULL,
  PRIMARY KEY (c_custkey))
  partition by key(c_custkey) partitions 9;

CREATE TABLE supplier (
  s_suppkey bigint NOT NULL,
  s_name char(25) DEFAULT NULL,
  s_address varchar(40) DEFAULT NULL,
  s_nationkey bigint DEFAULT NULL,
  s_phone char(15) DEFAULT NULL,
  s_acctbal bigint DEFAULT NULL,
  s_comment varchar(101) DEFAULT NULL,
  PRIMARY KEY (s_suppkey)
) partition by key(s_suppkey) partitions 9;

CREATE TABLE nation (
  n_nationkey bigint NOT NULL,
  n_name char(25) DEFAULT NULL,
  n_regionkey bigint DEFAULT NULL,
  n_comment varchar(152) DEFAULT NULL,
  PRIMARY KEY (n_nationkey));

CREATE TABLE region (
  r_regionkey bigint NOT NULL,
  r_name char(25) DEFAULT NULL,
  r_comment varchar(152) DEFAULT NULL,
  PRIMARY KEY (r_regionkey));
```

创建结束后查看表和表分组。

```
MySQL [tpch_10g_part]> show tables;
+-------------------------+
| Tables_in_tpch_10g_part |
+-------------------------+
| customer                |
| lineitem                |
```

```
| nation                   |
| orders                   |
| part                     |
| partsupp                 |
| region                   |
| supplier                 |
+--------------------------+
8 rows in set (0.365 sec)

MySQL [tpch_10g_part]> show tablegroups;
+---------------------------------+------------------+---------------+
| Tablegroup_name                 | Table_name       | Database_name |
+---------------------------------+------------------+---------------+
| oceanbase                       | NULL             | NULL          |
| tpch_tg_10g_lineitem_order_group | lineitem         | tpch_10g_part |
| tpch_tg_10g_lineitem_order_group | orders           | tpch_10g_part |
| tpch_tg_10g_partsupp_part       | part             | tpch_10g_part |
| tpch_tg_10g_partsupp_part       | partsupp         | tpch_10g_part |
+---------------------------------+------------------+---------------+
5 rows in set (0.068 sec)
```

加载数据

您可使用 OceanBase 数据库自带的 LOAD 命令逐个加载数据，也可以编写一个脚本批量加载数据，如下所示。

- 创建加载脚本目录

```
cd /root/TPC-H_Tools_v3.0.0/dbgen
mkdir load
cd load
cp ../dss.ri  ../dss.ddl ./
```

- 编写加载脚本

```
#/usr/bin/evn python
#-*- encoding:utf-8 -*-
import os
import sys
import time
import commands
hostname='172.xx.xxx.52'  #注意！！请填写某个 observer 所在服务器的 ip 地址
port='2881'               #端口号
tenant='test'              #组户名
user='tpch'               #用户名
password='******'           #密码
data_path='/tmp/tpch10'          #注意！！请填写某个 observer 所在服务器下 tbl 所在目录
db_name='tpch_10g_part'              #数据库名
cmd_str='mysql --default-auth=mysql_native_password,db_name -h%s -P%s -u%s@%s   -c -A %s  -e "sh
ow tables;" '%(hostname,port,user,tenant,db_name)
print cmd_str
result = commands.getstatusoutput(cmd_str)
print result
cmd_str=""" mysql --default-auth=mysql_native_password,db_name -h%s -P%s -u%s@%s  -c -A %s  -e "
load data /*+ parallel(8) */ infile '%s/customer.tbl' into table customer fields terminated by '
```

```
|';" """ %(hostname,port,user,tenant,db_name,data_path)
print cmd_str
result = commands.getstatusoutput(cmd_str)
print result
cmd_str=""" mysql --default-auth=mysql_native_password,db_name -h%s -P%s -u%s@%s  -c -A %s  -e "
load data /*+ parallel(8) */ infile '%s/lineitem.tbl' into table lineitem fields terminated by '
|';" """ %(hostname,port,user,tenant,db_name,data_path)
result = commands.getstatusoutput(cmd_str)
print result
cmd_str=""" mysql --default-auth=mysql_native_password,db_name -h%s -P%s -u%s@%s  -c -A %s  -e "
load data /*+ parallel(8) */ infile '%s/nation.tbl' into table nation fields terminated by '|';
" """ %(hostname,port,user,tenant,db_name,data_path)
result = commands.getstatusoutput(cmd_str)
print result
cmd_str=""" mysql --default-auth=mysql_native_password,db_name -h%s -P%s -u%s@%s  -c -A %s  -e "
load data /*+ parallel(8) */ infile '%s/orders.tbl' into table orders fields terminated by '|';
" """ %(hostname,port,user,tenant,db_name,data_path)
result = commands.getstatusoutput(cmd_str)
print result
cmd_str=""" mysql --default-auth=mysql_native_password,db_name -h%s -P%s -u%s@%s  -c -A %s  -e "
load data /*+ parallel(8) */ infile '%s/partsupp.tbl' into table partsupp fields terminated by '
|';" """ %(hostname,port,user,tenant,db_name,data_path)
result = commands.getstatusoutput(cmd_str)
print result
cmd_str=""" mysql --default-auth=mysql_native_password,db_name -h%s -P%s -u%s@%s  -c -A %s  -e "
load data /*+ parallel(8) */ infile '%s/part.tbl' into table part fields terminated by '|';" ""
" %(hostname,port,user,tenant,db_name,data_path)
result = commands.getstatusoutput(cmd_str)
print result
cmd_str=""" mysql --default-auth=mysql_native_password,db_name -h%s -P%s -u%s@%s  -c -A %s  -e "
load data /*+ parallel(8) */ infile '%s/region.tbl' into table region fields terminated by '|';
" """ %(hostname,port,user,tenant,db_name,data_path)
result = commands.getstatusoutput(cmd_str)
print result
cmd_str=""" mysql --default-auth=mysql_native_password,db_name -h%s -P%s -u%s@%s  -c -A %s  -e "
load data /*+ parallel(8) */ infile '%s/supplier.tbl' into table supplier fields terminated by '
|';" """ %(hostname,port,user,tenant,db_name,data_path)
```

```
export MYSQL_PWD=******
python load.py
```

性能调优

转储与合并

对比项	合并转储前	合并后	提升率
tpch_10g（总用时）	57.26	23.28	59.30%
Q1	2.43s	2.02s	
Q2	0.86s	0.32s	

Q3	2.08s	1.49s	
Q4	0.45s	0.34s	
Q5	0.96s	1.58s	
Q6	1.29s	0.83s	
Q7	2.63s	1.41s	
Q8	3.60s	1.39s	
Q9	1.53s	1.85s	
Q10	4.23s	2.38s	
Q11	0.59s	0.47s	
Q12	2.01s	1.00s	
Q13	2.34s	0.85s	
Q14	0.51s	0.21s	
Q15	1.09s	0.57s	
Q16	1.18s	0.41s	
Q17	**10.54s**	0.94s	
Q18	1.94s	0.98s	
Q19	2.32s	1.02s	
Q20	**10.46s**	1.25s	
Q21	2.49s	1.42s	
Q22	1.73s	0.55s	

调整 PRIMARY_ZONE

对比项	集中式部署（zone1）	random 部署
tpch	**590.42**	**149.13**
Q1	**50.74s**	**15.14s**
Q2	2.25s	0.47s
Q3	**37.69s**	**13.42s**
Q4	**42.78s**	2.57s
Q5	6.94s	2.52s

Q6	**26.62s**	7.38s
Q7	0.35s	0.23s
Q8	**67.92s**	**11.69s**
Q9	**20.19s**	6.17s
Q10	**54.79s**	**24.00s**
Q11	0.39s	0.19s
Q12	**40.71s**	8.60s
Q13	**20.99s**	7.82s
Q14	**22.99s**	1.52s
Q15	**46.28s**	2.95s
Q16	7.41s	2.88s
Q17	**32.04s**	8.64s
Q18	**26.64s**	8.29s
Q19	**31.25s**	9.42s
Q20	**38.14s**	**10.74s**
Q21	0.24s	0.15s
Q22	**13.07s**	4.34s

使用 TABLEGROUP

对比项	未使用 pg	使用 pg
tpch	25.3	23.28
Q1	2.23s	2.02s
Q2	0.59s	0.32s
Q3	1.59s	1.49s
Q4	0.46s	0.34s
Q5	1.65s	1.58s
Q6	0.80s	0.83s
Q7	1.58s	1.41s
Q8	1.52s	1.39s

Q9	1.83s	1.85s
Q10	2.41s	2.38s
Q11	0.54s	0.47s
Q12	1.14s	1.00s
Q13	0.93s	0.85s
Q14	0.23s	0.21s
Q15	0.63s	0.57s
Q16	0.46s	0.41s
Q17	0.98s	0.94s
Q18	1.18s	0.98s
Q19	1.03s	1.02s
Q20	1.34s	1.25s
Q21	1.55s	1.42s
Q22	0.63s	0.55s

6.6 如何使用 JMeter 跑业务场景测试

前提条件

使用 JMeter 测试 OceanBase 数据库性能的时候需要加载 OceanBase 数据库的 Java 驱动。您可以从官网的下载文件中获取，地址：https://help.aliyun.com/document_detail/212815.html。

下载的 `oceanbase-client-1.x.x.jar` 需要放到 `Jmeter` 的 `lib` 文件夹中。

业务场景定义

建表语句

请在 OceanBase MySQL 租户下业务账户执行如下 SQL。

```
$obclient -h127.1 -utpcc@test#obdemo -P2883 -p***** -c -A tpcc

CREATE TABLE account(id bigint NOT NULL AUTO_INCREMENT PRIMARY KEY
 , name varchar(50) NOT NULL UNIQUE
 , value bigint NOT NULL
 , gmt_create timestamp DEFAULT current_timestamp NOT NULL
 , gmt_modified timestamp DEFAULT current_timestamp NOT NULL  );
```

场景 SQL

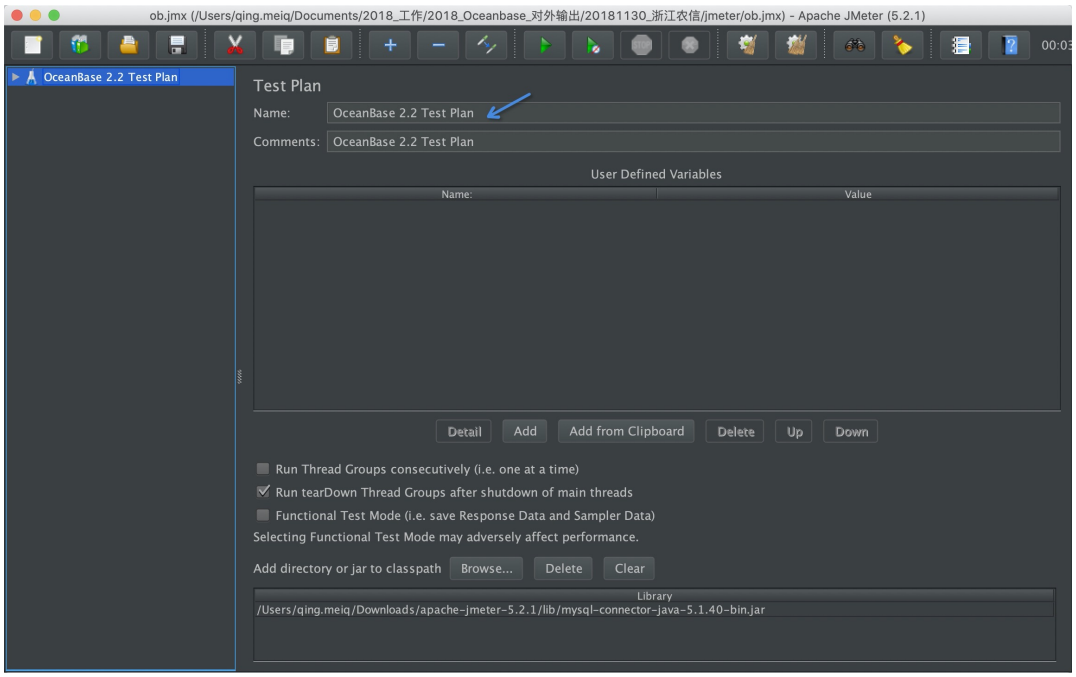
此次测试模拟一个分布式事务，SQL 示例如下：

```
-- session A
begin ;
select value from account where id = 174 for update ;
update account set  value = value - 7 , gmt_modified = current_timestamp where id = 174 ;
update account set  value = value + 7 , gmt_modified = current_timestamp where id = 165 ;
-- commit or rollback;
commit;
```

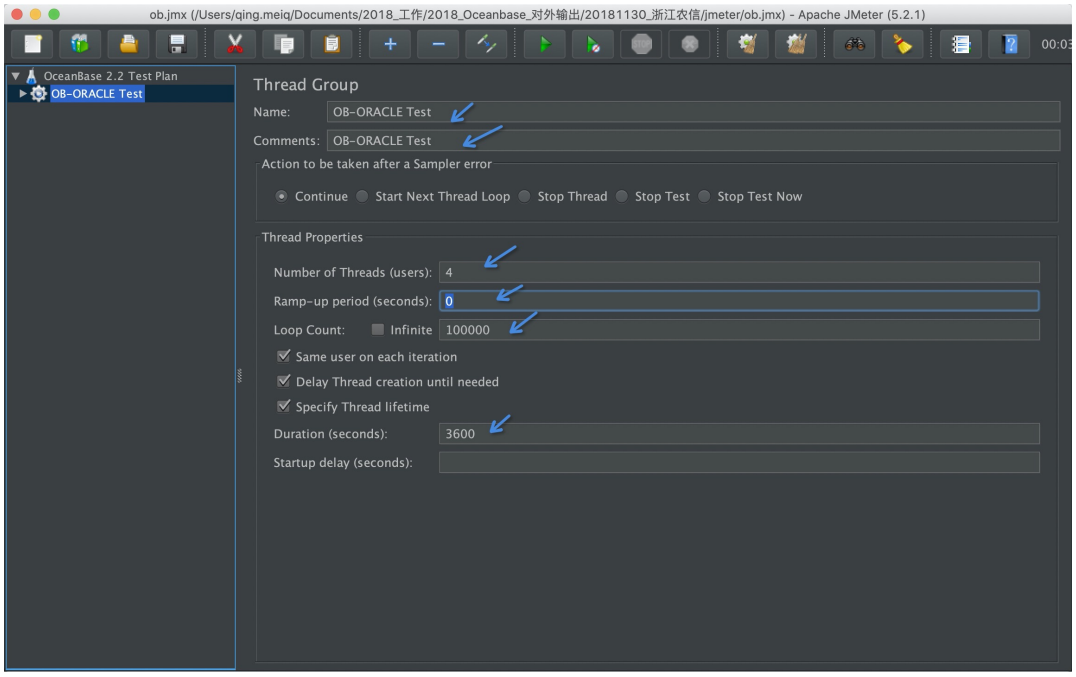
新建 JMeter 测试计划

JMeter 能在命令行下运行，也可在图形界面运行，这里以图形界面为例。

测试计划属性



新建 `Thread-Group`

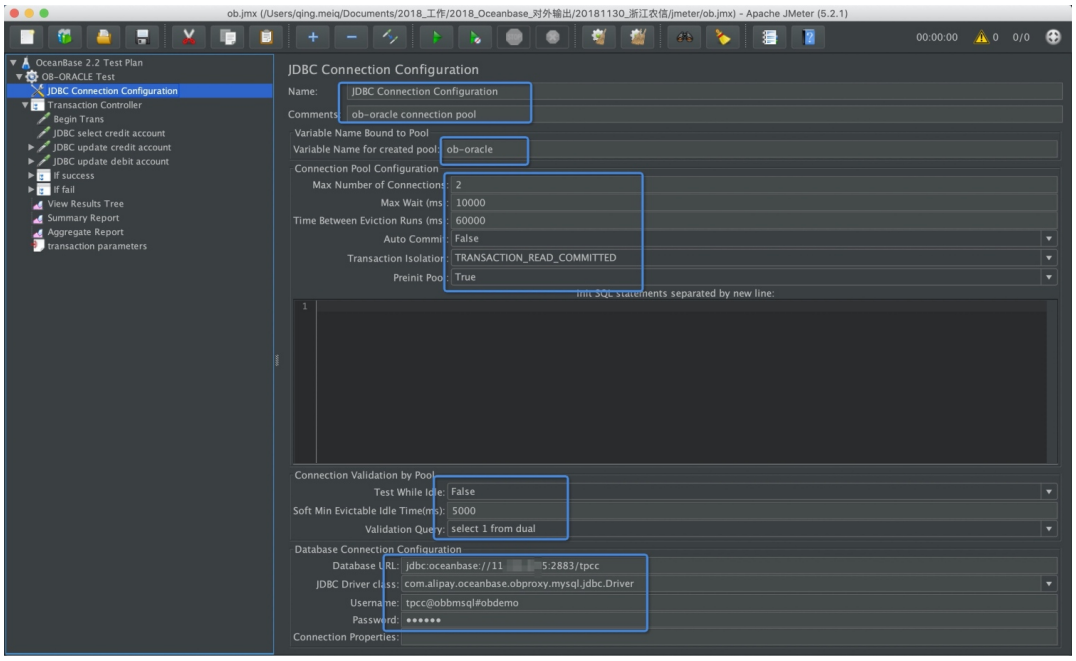


此处有很多与多线程运行有关的 JMeter 参数，具体说明请参考 JMeter 官网文档。

注意

此处的 `bigint of Threads` 指压测的客户端线程数。

JDBC 连接属性



上图中有很多属性，都是一个连接池常具备的参数。详情参考网上关于 Java 连接池配置的经验。

您需注意以下参数:

- `Max bigint of Connections`：指连接池里的最大连接数。

  如果压测线程数远高于这个值，那么压测线程可能需要等待这个连接池创建或返还数据库连接（即到 OceanBase 数据库的连接）给它，如果等不到可能会报错。在这个环节，客户端压测线程获取不到连接，不一定跟 OceanBase 数据库有直接关系。在 Java 应用里面也同理。

- `Transaction Isolation`：数据库连接使用的事务隔离级别。

  OceanBase 数据库支持两种事务隔离级别：读已提交（`Read-Committed`）和序列化（`Serializable`）。

  前者很常见容易理解，后者是 Oracle 特有的隔离级别，OceanBase 数据库也兼容。详情参考《OceanBase 事务引擎特性和应用实践分享》，理解序列化隔离级别的特点和场景可以加深对数据库事务的理解。

- `Test While Idle`：连接探活(`keepalive`)设置。

  这个设置对应用很有必要。有时候应用会提示数据库在一个关闭的连接上执行 SQL，进而报错，这是由于连接池中的数据库连接因为其他原因断开了。所以，数据库连接池通常都需要探活机制。此处因为压测场景基本无闲置连接，所以可以设置为 `False`。

- `Database URL`：数据库连接 URL 格式，例如 `jdbc:oceanbase://11.***.***.5:2883/tpcc`。

- `JDBC Driver Class`：数据库驱动中的 Main 类名。

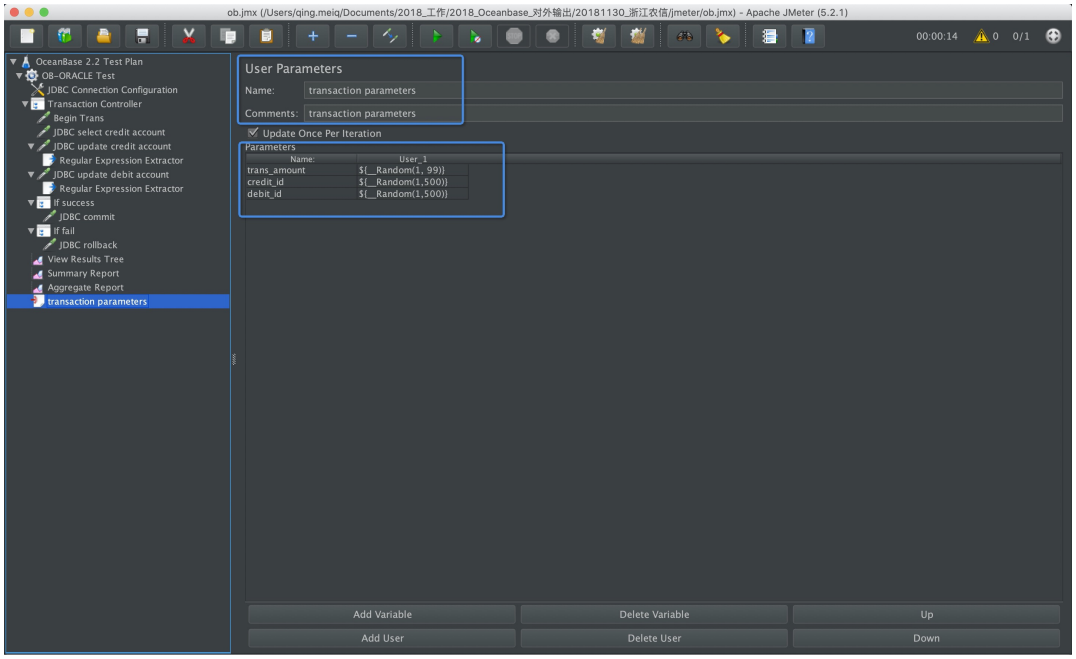
  Main 类名需要按 OceanBase 格式填写，例如`com.alipay.oceanbase.obproxy.mysql.jdbc.Driver`。

- `Username`：用户格式。

  OceanBase 数据库的用户名格式比较特别，格式为 `租户里用户名@租户名#集群名` 或 `集群名:租户名:租户里用户名`，如 `tpcc@test#obdemo`。

事务参数(变量)

在本测试中，有三个变量：账户 A，账户 B，转账金额。所以需要设置参数。

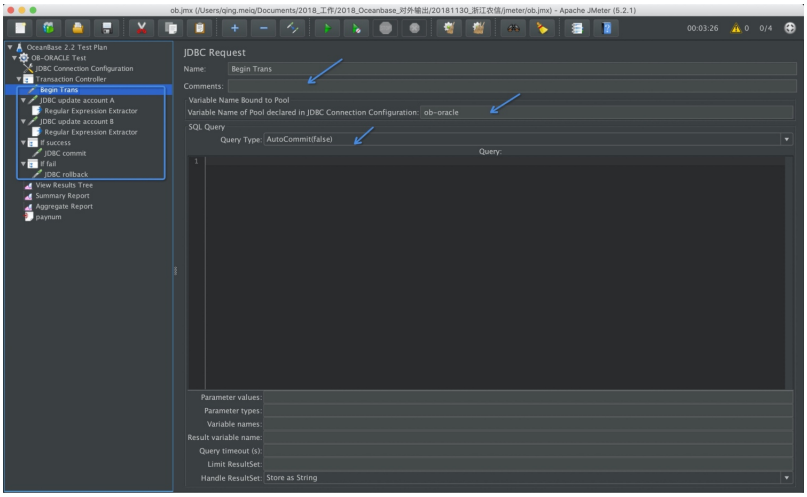


账户参数和金额采取随机数，随机数的值不可超出测试数据的实际范围。

事务控制器

此处是维护每个事务的逻辑。事务由一组 `JDBC` 请求组成。

- 开启事务



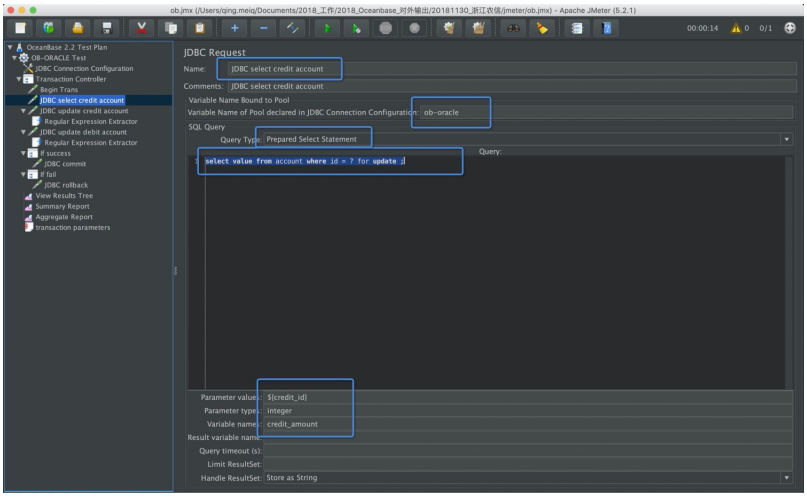
选择 `Autocommit(false)`，开启显式事务。

- 查询账户 A 的余额

查询账户 A 记录的同时会锁住这笔记录，即常用的悲观锁技术。此步骤是否进行取决于测试需要，不是必需

的。

```
select value from account where id = ? for update ;
```



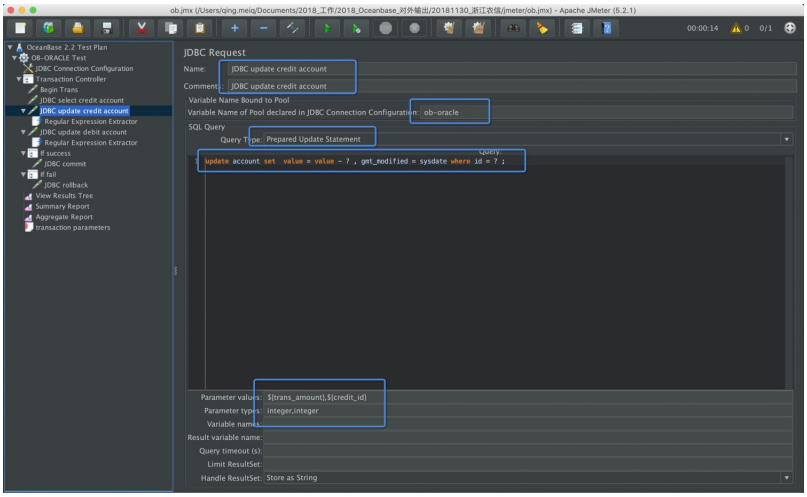
注意 所有参数都使用 `Prepared Statement`，以下相同。

此步骤操作之后，按业务设计应该检查返回值是否大于要转账的值，如果不满足将会提示“转账余额不足”。此处省略研究 JMeter 如何根据查询返回值进行逻辑判断。

- 扣减账户 A 的余额

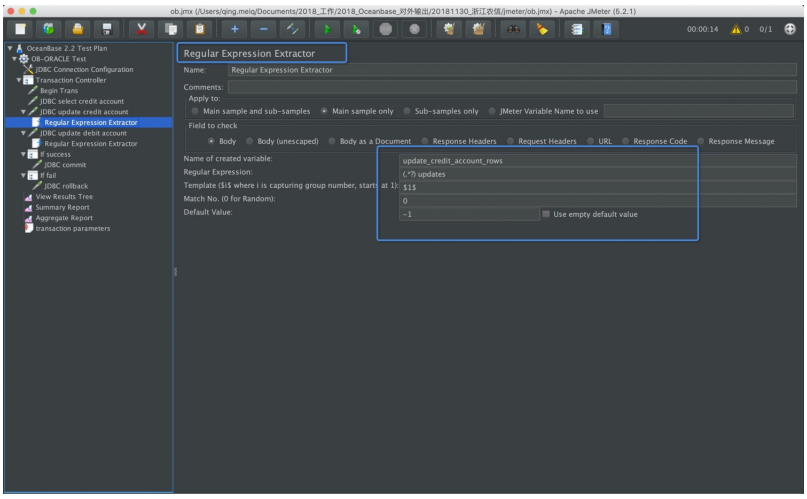
  示例 SQL。

```
update account set  value = value - ? , gmt_modified = current_timestamp where id = ? ;
```



多个绑定参数使用逗号 `,` 分隔。

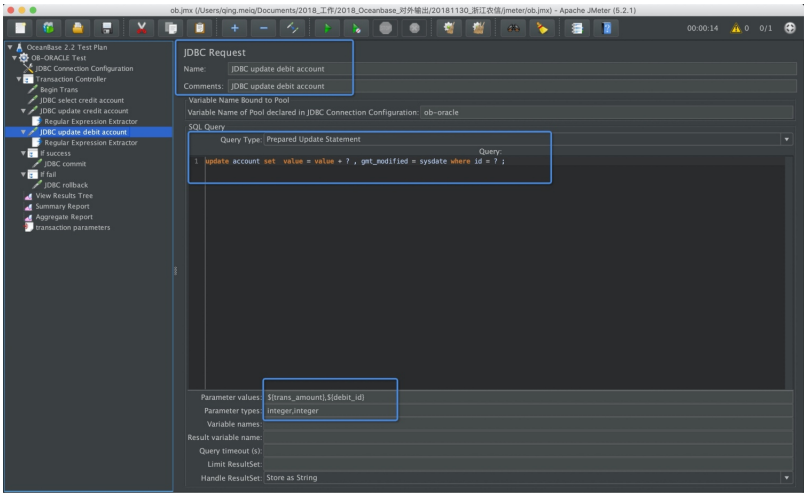
需要新增一个 Post 处理逻辑，获取更新返回值。



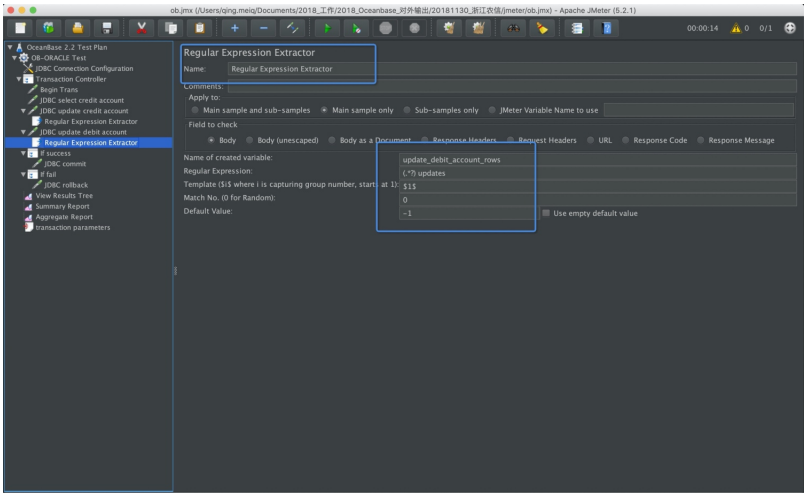
- 增加账户 B 的余额

  示例 SQL:

```
update account set  value = value + ? , gmt_modified = current_timestamp where id = ? ;
```



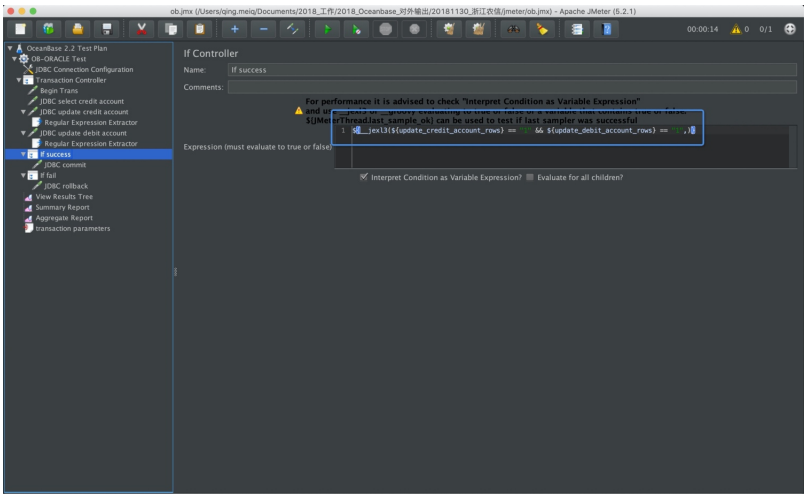
同样，需要增加一个 Post 处理器。



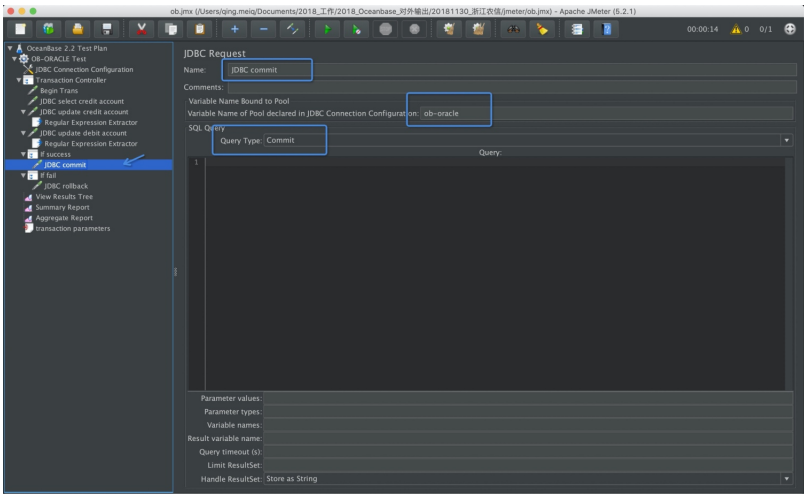
判断逻辑 - 成功流程

如果上面两笔账户更新成功，则提交事务。

- 新增判断控制 `IF`



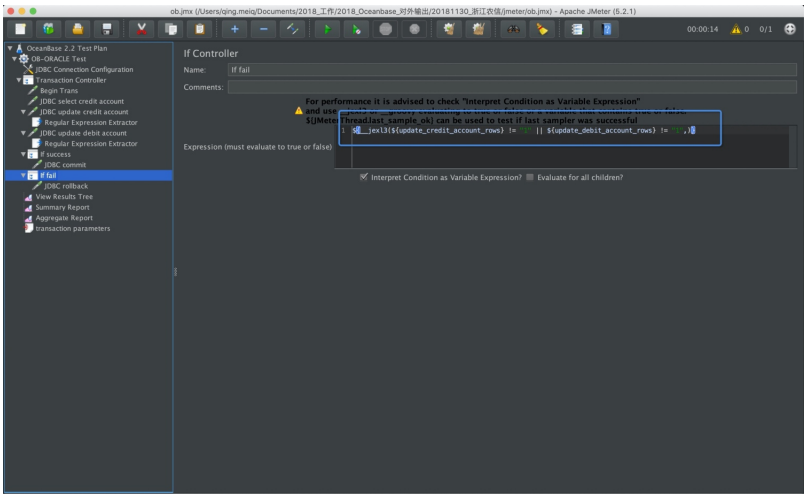
- 新增动作



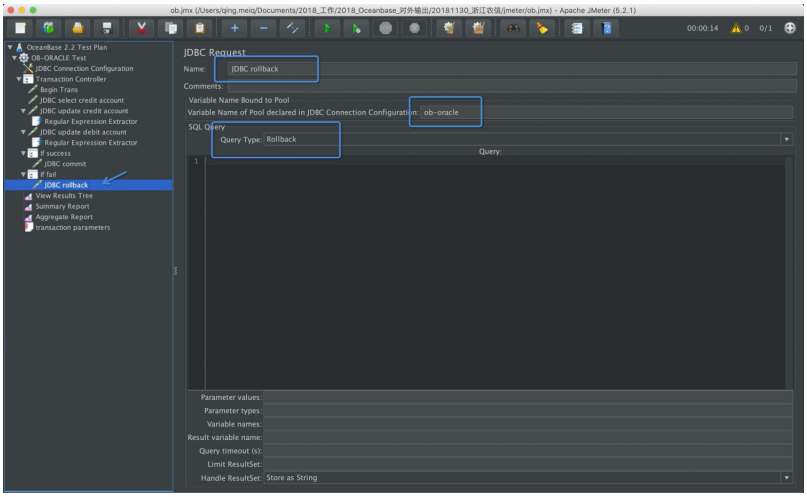
判断逻辑 - 失败流程

如果上面两笔账户的更新有一笔失败，则回滚事务。

- 新增判断控制 `IF`

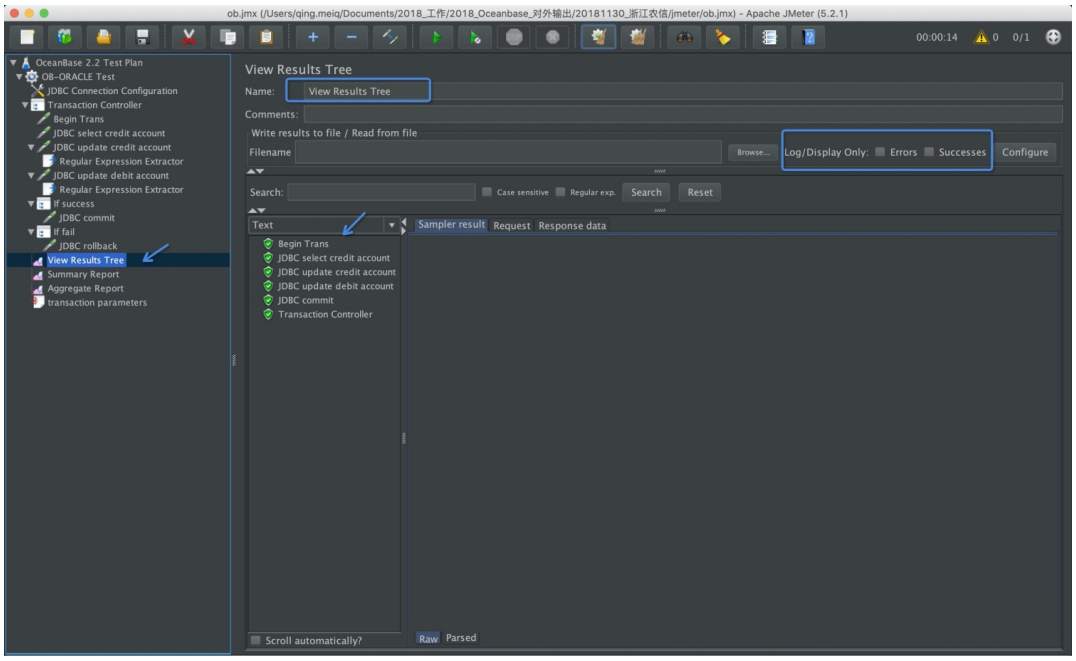


- 新增动作

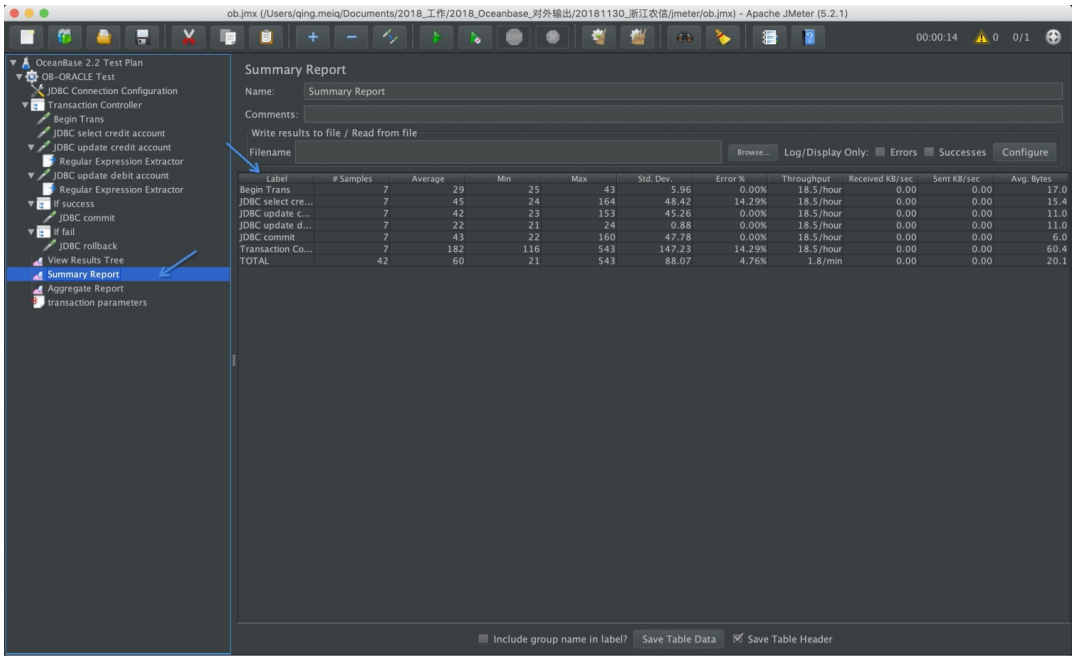


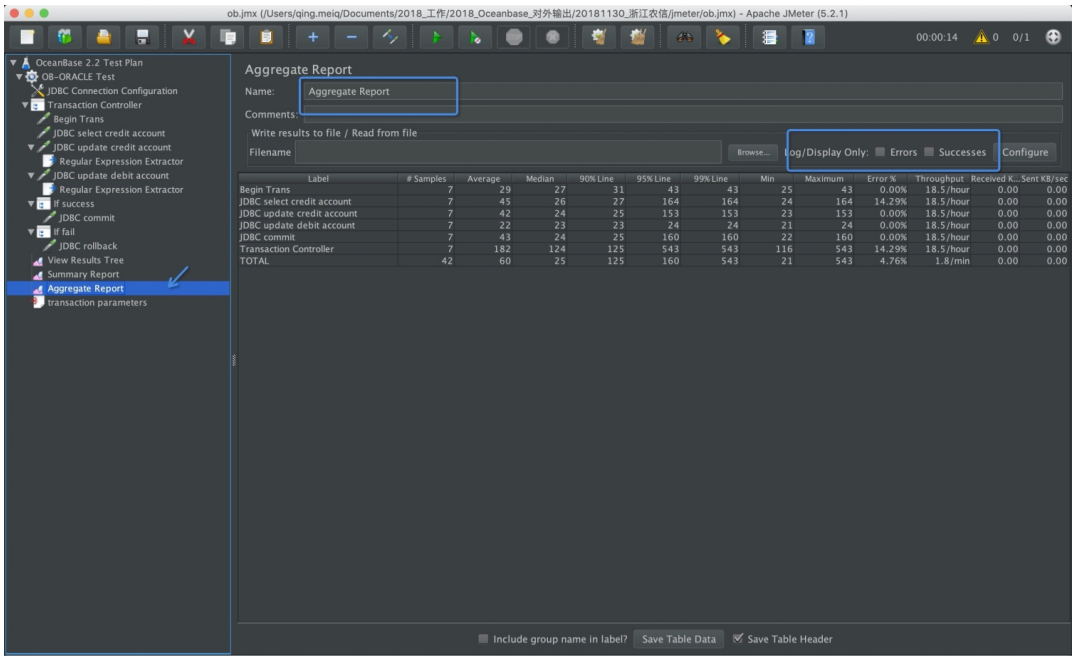
查看结果

可以查看成功、失败的结果。


ob.jmx (/Users/qing.meiq/Documents/2018_工作/2018_Oceanbase_对外输出/20181130_浙江农信/jmeter/ob.jmx) - Apache JMeter (5.2.1)
00:00:14
0
0/1
OceanBase 2.2 Test Plan
OB-ORACLE Test
JDBC Connection Configuration
Transaction Controller
Begin Trans
JDBC select credit account
JDBC update credit account
Regular Expression Extractor
JDBC update debit account
Regular Expression Extractor
If success
JDBC commit
If fail
JDBC rollback
View Results Tree
Summary Report
Aggregate Report
transaction parameters
View Results Tree
Name:
View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only:
Errors
Successes
Configure
Search:
Case sensitive
Regular exp.
Search
Reset
Text
Sampler result
Request
Response data
Begin Trans
JDBC select credit account
JDBC update credit account
JDBC update debit account
JDBC commit
Transaction Controller
Scroll automatically?
Raw
Parsed

查看汇总报告。


ob.jmx (/Users/qing.meiq/Documents/2018_工作/2018_Oceanbase_对外输出/20181130_浙江农信/jmeter/ob.jmx) - Apache JMeter (5.2.1)
00:00:14
0
0/1
OceanBase 2.2 Test Plan
OB-ORACLE Test
JDBC Connection Configuration
Transaction Controller
Begin Trans
JDBC select credit account
JDBC update credit account
Regular Expression Extractor
JDBC update debit account
Regular Expression Extractor
If success
JDBC commit
If fail
JDBC rollback
View Results Tree
Summary Report
Aggregate Report
transaction parameters
Summary Report
Name:
Summary Report
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only:
Errors
Successes
Configure
Label
Samples
Average
Min
Max
Std. Dev.
Error %
Throughput
Received KB/sec
Sent KB/sec
Avg. Bytes
Begin Trans 7 29 25 43 5.96 0.00% 18.5/hour 0.00 0.00 17.0
JDBC select cre... 7 45 24 164 48.42 14.29% 18.5/hour 0.00 0.00 15.4
JDBC update c... 7 42 23 153 45.26 0.00% 18.5/hour 0.00 0.00 11.0
JDBC update d... 7 22 21 24 0.88 0.00% 18.5/hour 0.00 0.00 11.0
JDBC commit 7 43 22 160 47.78 0.00% 18.5/hour 0.00 0.00 6.0
Transaction Co... 7 182 116 543 147.23 14.29% 18.5/hour 0.00 0.00 60.4
TOTAL 42 60 21 543 88.07 4.76% 1.8/min 0.00 0.00 20.1
Include group name in label?
Save Table Data
Save Table Header


ob.jmx (/Users/qing.meiq/Documents/2018_工作/2018_Oceanbase_对外输出/20181130_浙江农信/jmeter/ob.jmx) - Apache JMeter (5.2.1)
00:00:14
0
0/1
OceanBase 2.2 Test Plan
OB-ORACLE Test
JDBC Connection Configuration
Transaction Controller
Begin Trans
JDBC select credit account
JDBC update credit account
Regular Expression Extractor
JDBC update debit account
Regular Expression Extractor
If success
JDBC commit
If fail
JDBC rollback
View Results Tree
Summary Report
Aggregate Report
transaction parameters
Aggregate Report
Name:
Aggregate Report
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only:
Errors
Successes
Configure
Label
Samples
Average
Median
90% Line
95% Line
99% Line
Min
Maximum
Error %
Throughput
Received K...
Sent KB/sec
Begin Trans 7 29 27 31 43 43 25 43 0.00% 18.5/hour 0.00 0.00
JDBC select credit account 7 45 26 27 164 164 24 164 14.29% 18.5/hour 0.00 0.00
JDBC update credit account 7 42 24 25 153 153 23 153 0.00% 18.5/hour 0.00 0.00
JDBC update debit account 7 22 23 23 24 24 21 24 0.00% 18.5/hour 0.00 0.00
JDBC commit 7 43 24 25 160 160 22 160 0.00% 18.5/hour 0.00 0.00
Transaction Controller 7 182 124 125 543 543 116 543 14.29% 18.5/hour 0.00 0.00
TOTAL 42 60 25 125 160 543 21 543 4.76% 1.8/min 0.00 0.00
Include group name in label?
Save Table Data
Save Table Header

第 7 章：OceanBase 数据库性能诊断和调优

本章主要介绍 OceanBase 性能诊断和调优技巧以及部分原理。

本章目录

7.1 性能诊断调优概述

客户端 SQL 执行过程分为以下步骤:

- 客户端将 SQL 发往 OBProxy 节点。
- OBProxy 将 SQL 发往后端 OBServer 节点。
- OBServer 解析 SQL，执行 SQL，返回结果给 OBProxy 节点。
- OBProxy 将数据返回给客户端。

OceanBase 的 SQL 性能诊断是围绕这个链路上的几个关键环节进行，如 OBProxy 的路由、OBServer 的 SQL 引擎等。

SQL 引擎简介

OBServer 进程包含 SQL 引擎、事务引擎和存储引擎。其中，SQL 引擎的设计跟 Oracle SQL 引擎设计理念一致，都有 SQL 语法解析、执行计划缓存、软解析和 outline 等技术。

OBProxy 进程并不包含 SQL 引擎，只能做简单的 SQL 分析，判断该 SQL 通过 OBProxy 到后端哪个 OBServer 节点。未来 OBProxy 可能会包含 OBServer 的 SQL 引擎能力。

SQL 执行计划简介

SQL 执行计划即 SQL 执行的路径或者算法，影响 SQL 的执行性能。

OceanBase SQL 引擎支持 SQL 解析，解析后的执行计划会缓存，下次可以复用。跟传统数据库一样，SQL 文本不一样，执行计划不会复用。此外，如果 SQL 文本一样，但是执行环境不一样，执行计划也不会复用。

目前已知的执行环境包括运行节点 IP、会话级别的并行参数。当然可能还有其他情况，您可以从这个角度判断所有数据库的 SQL 执行引擎的成熟度。

OceanBase 缓存执行计划的两个视图分别是 `gv$plan_cache_plan_stat` 和 `gv$plan_cache_plan_explain`，这是实际运行使用的执行计划，更为准确。不过初学的时候看这个比较麻烦。所以本章先探讨用 EXPLAIN 命令解析的执行计划。

先介绍执行计划的特点:

- 没有绝对正确的执行计划，尤其是 SQL 比较复杂的时候。
- 同样的 SQL，不同 OBServer 版本，执行计划可能会有变化。
- 同样的 SQL，执行计划也不可能一成不变。影响执行计划变化的因素可能有数据量（或者统计信息）、会话参数（或变量）、数据分布特点、SQL 被路由到的节点等。后两者是 OceanBase 特有的。
- EXPLAIN 解析的执行计划可能跟实际运行的执行计划不一样，受 OBProxy 和 OBServer 的 LOCATION

CACHE 和 SQL 路由策略影响。

- SQL 执行计划可以通过 SQL 注释干预，这个需要改 SQL。如果 SQL 不可以改，可以通过 OUTLINE 在线干预。干预的结果可能成功也可能失败。

SQL HINTS

OceanBase 支持在 SQL 里使用 HINTS 来改变执行计划或控制其他行为。

语句级别的 hint

```
FROZEN_VERSION
QUERY_TIMEOUT
READ_CONSISTENCY
LOG_LEVEL
QB_NAME
ACTIVATE_BURIED_POINT
TRACE_LOG
MAX_CONCURRENT
```

说明:

- `/*+ QUERY_TIMEOUT(100000000) */`: 对 SQL 超时时间进行控制。
- `/*+ READ_CONSISTENCY(weak) */`: 用来设置弱一致性读。
- `/*+ MAX_CONCURRENT(8) */`: 用来对 SQL 的并发进行限制的。

计划相关的 hint

```
FULL
INDEX
LEADING
USE_MERGE
USE_HASH
USE_NL
ORDERED
NO_REWRITE
```

说明:

- `/*+ FULL(表名) */`: 强制对表进行全表扫描。
- `/*+ LEADING(表名) */`: 强制某表为驱动表。可以间接改变连接算法。
- `/*+ USE_NL(表a,表b,表c) */`: 设置表 a、表 b 和表 c 的连接算法为嵌套循环。
- `/*+ ORDERED */`: 按照 `from` 子句后的表顺序进行表连接。

OceanBase 性能诊断思路

OceanBase 性能问题通常包括:

- 大表查询性能，大批量数据导出性能等。
- 大事务性能，大批量数据加载性能等。
- 普通的读写性能。

有些性能问题往往是集群出现稳定性问题之后出现的。比如，网络延时增大或者节点时间误差增大导致的节点状态不稳定，进而影响了性能；磁盘故障或者磁盘空间满导致的合并异常问题，也可能间接影响性能。

由于 OceanBase 可靠性很高，即使集群不稳定，也不会出现不可用的状态。所以，稳定性问题隐藏的有点深，不为业务所见。业务更多的时候感知的是性能问题。当然，运维使用一些监控手段，也能够及时发现稳定性问题。

性能诊断的方向如下（暂不考虑稳定性问题带来的性能问题）：

- OBProxy 节点的负载。包括 CPU、网络、内存等。
- OBServer 节点的负载。包括 CPU、磁盘、内存、网络等。
- SQL 的执行类型分布（本地 SQL 和远程 SQL 的比例）。
- SQL 的执行计划。
- 集群和租户的内存管理（内存参数、转储和合并进度等）。
- 业务事务的具体 SQL 逻辑。

7.2 OBProxy SQL 路由原理

OBProxy 的主要功能是提供 SQL 路由。所以要先了解数据的位置，再了解 SQL 路由策略。

查看租户资源单元位置

租户的资源通常分布在每个 ZONE 的机器上。租户的数据就在租户资源所在的机器上。集群规模很大的时候，每个租户可能只是在部分机器上。OBProxy 首先需要知道租户的位置（`LOCATION CACHE`）。

如下有 2 种方法可以确认租户的位置:

- sys 租户直接查询

```
select t1.name resource_pool_name, t2.`name` unit_config_name, t2.max_cpu, t2.min_cpu, round(t2.
max_memory/1024/1024/1024) max_mem_gb, round(t2.min_memory/1024/1024/1024) min_mem_gb, t3.unit_i
d, t3.zone, concat(t3.svr_ip,':',t3.`svr_port`) observer,t4.tenant_id, t4.tenant_name
from __all_resource_pool t1 join __all_unit_config t2 on (t1.unit_config_id=t2.unit_config_id)
    join __all_unit t3 on (t1.`resource_pool_id` = t3.`resource_pool_id`)
    left join __all_tenant t4 on (t1.tenant_id=t4.tenant_id)
order by t1.`resource_pool_id`, t2.`unit_config_id`, t3.unit_id
;
```

输出:

```
+--------------------+------------------+---------+---------+------------+------------+---------
| resource_pool_name | unit_config_name | max_cpu | min_cpu | max_mem_gb | min_mem_gb | unit_id
+--------------------+------------------+---------+---------+------------+------------+---------
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       1
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       2
| sys_pool           | sys_unit_config  |       5 |       5 |          2 |          2 |       3
| my_pool_zone1      | unit1            |       9 |       9 |         19 |         19 |    1001
| my_pool_zone2      | unit1            |       9 |       9 |         19 |         19 |    1002
| my_pool_zone3      | unit1            |       9 |       9 |         19 |         19 |    1003
+--------------------+------------------+---------+---------+------------+------------+---------
6 rows in set (0.008 sec)
```

- 业务租户直接查询

```
select tenant_name, unit_id, unit_config_name, resource_pool_name, zone, svr_ip ,max_cpu,min_cpu
,round(max_memory/1024/1024/1024) max_mem_gb,round(min_memory/1024/1024/1024) min_mem_gb
from gv$unit
WHERE tenant_id=1002;
```

输出:

```
+-------------+---------+------------------+--------------------+-------+---------------+-------
--+---------+------------+------------+
| tenant_name | unit_id | unit_config_name | resource_pool_name | zone  | svr_ip        | max_cp
u | min_cpu | max_mem_gb | min_mem_gb |
```

```
+-------------+---------+-----------------+--------------------+-------+--------------+-------
--+---------+-----------+------------+
| obmysql     |    1001 | unit1           | my_pool_zone1      | zone1 | 172.20.249.52 |
9 |       9 |         19 |         19 |
| obmysql     |    1002 | unit1           | my_pool_zone2      | zone2 | 172.20.249.49 |
9 |       9 |         19 |         19 |
| obmysql     |    1003 | unit1           | my_pool_zone3      | zone3 | 172.20.249.51 |
9 |       9 |         19 |         19 |
+-------------+---------+-----------------+--------------------+-------+--------------+-------
--+---------+-----------+------------+
3 rows in set (0.031 sec)
```

在业务租户里，条件 `tenant_id=1002` 也可以不用，因为每个业务租户只能查看自己的租户资源单元信息。

说明

直连 OBServer 时，如果该 OBServer 节点上没有这个租户的资源单元，连接会报错。

查看分区副本位置

每个租户的分区都是从租户的资源单元中分配的。以下 SQL 可以在 sys 租户里查看租户中所有数据库的主副本位置。

```
SELECT  a.tenant_name, t.table_name, d.database_name,  tg.tablegroup_name , t.part_num , t2.part
 , IF(t.primary_zone = '' OR t.primary_zone IS NULL, a.primary_zone, t.primary_zone) primary_zon
FROM oceanbase.__all_tenant AS a
 JOIN oceanbase.__all_virtual_database AS d ON ( a.tenant_id = d.tenant_id  )
 JOIN oceanbase.__all_virtual_table AS t  ON (t.tenant_id = d.tenant_id AND t.database_id = d.da
 JOIN oceanbase.__all_virtual_meta_table t2 ON (t.tenant_id = t2.tenant_id AND (t.table_id=t2.ta
 LEFT JOIN oceanbase.__all_virtual_tablegroup AS tg ON (t.tenant_id = tg.tenant_id and  t.tableg
WHERE a.tenant_id IN (1002 ) AND  t.table_type IN (3)
 AND d.database_name IN ( 'tpccdb')
ORDER BY t.tenant_id, tg.tablegroup_name, d.database_name, t.table_name, t2.partition_id
;
```

输出:

```
+-------------+-----------------+---------------+-----------------+----------+--------------+--
| tenant_name | table_name      | database_name | tablegroup_name | part_num | partition_id | Z
+-------------+-----------------+---------------+-----------------+----------+--------------+--
| obmysql     | bmsql_config    | tpccdb        | NULL            |        1 |            0 | z
| obmysql     | bmsql_item      | tpccdb        | NULL            |        1 |            0 | z
| obmysql     | bmsql_customer  | tpccdb        | tpcc_group      |        3 |            0 | z
| obmysql     | bmsql_customer  | tpccdb        | tpcc_group      |        3 |            1 | z
| obmysql     | bmsql_customer  | tpccdb        | tpcc_group      |        3 |            2 | z
| obmysql     | bmsql_district  | tpccdb        | tpcc_group      |        3 |            0 | z
| obmysql     | bmsql_district  | tpccdb        | tpcc_group      |        3 |            1 | z
| obmysql     | bmsql_district  | tpccdb        | tpcc_group      |        3 |            2 | z
| obmysql     | bmsql_history   | tpccdb        | tpcc_group      |        3 |            0 | z
| obmysql     | bmsql_history   | tpccdb        | tpcc_group      |        3 |            1 | z
| obmysql     | bmsql_history   | tpccdb        | tpcc_group      |        3 |            2 | z
| obmysql     | bmsql_new_order | tpccdb        | tpcc_group      |        3 |            0 | z
```

```
| obmysql     | bmsql_new_order  | tpccdb        | tpcc_group      |         3 |            1 | z
| obmysql     | bmsql_new_order  | tpccdb        | tpcc_group      |         3 |            2 | z
| obmysql     | bmsql_oorder     | tpccdb        | tpcc_group      |         3 |            0 | z
| obmysql     | bmsql_oorder     | tpccdb        | tpcc_group      |         3 |            1 | z
| obmysql     | bmsql_oorder     | tpccdb        | tpcc_group      |         3 |            2 | z
| obmysql     | bmsql_order_line | tpccdb        | tpcc_group      |         3 |            0 | z
| obmysql     | bmsql_order_line | tpccdb        | tpcc_group      |         3 |            1 | z
| obmysql     | bmsql_order_line | tpccdb        | tpcc_group      |         3 |            2 | z
| obmysql     | bmsql_stock      | tpccdb        | tpcc_group      |         3 |            0 | z
| obmysql     | bmsql_stock      | tpccdb        | tpcc_group      |         3 |            1 | z
| obmysql     | bmsql_stock      | tpccdb        | tpcc_group      |         3 |            2 | z
| obmysql     | bmsql_warehouse  | tpccdb        | tpcc_group      |         3 |            0 | z
| obmysql     | bmsql_warehouse  | tpccdb        | tpcc_group      |         3 |            1 | z
| obmysql     | bmsql_warehouse  | tpccdb        | tpcc_group      |         3 |            2 | z
+-------------+------------------+---------------+-----------------+-----------+--------------+--
26 rows in set (0.095 sec)
```

`role` 是副本角色。`1` 表示主副本，`2` 表示备副本。

下面 SQL 是 OBProxy 查看一个具体的分区副本位置。视图 `__all_virtual_proxy_schema` 的访问要求非常严格，必须提供下面 4 个条件。通常不建议运维人员自己查询这个视图。

```
select table_name,partition_id, part_num, svr_ip,role, replica_num, table_type
from __all_virtual_proxy_schema
where tenant_name = 'obmysql' and database_name = 'tpccdb' and table_name = 'bmsql_oorder' and partiti
on_id = 1 ;
```

输出:

```
+--------------+--------------+----------+---------------+------+-------------+------------+
| table_name   | partition_id | part_num | svr_ip        | role | replica_num | table_type |
+--------------+--------------+----------+---------------+------+-------------+------------+
| bmsql_oorder |            1 |        3 | 172.20.249.49 |    2 |           3 |          3 |
| bmsql_oorder |            1 |        3 | 172.20.249.51 |    2 |           3 |          3 |
| bmsql_oorder |            1 |        3 | 172.20.249.52 |    1 |           3 |          3 |
+--------------+--------------+----------+---------------+------+-------------+------------+
3 rows in set (0.028 sec)
```

OBProxy 获取到每个分区的位置缓存在进程内部。如果后端 OBServer 节点宕机，发生部分分区主备故障切换，OBProxy 不一定能立即感知到，而是在 SQL 发到老的主副本节点时报错后，OBProxy 重新获取该分区新的主副本位置，然后更新自己的分区位置缓存。

调整 PRIMARY_ZONE 设置

PRIMARY_ZONE 设置主副本分布策略，通常可以在租户级别设置。MySQL 租户还可以在数据库级别再设置，数据库设置覆盖租户默认设置。此外，也可以在表分组或表级别设置，表分组和表的 PRIMARY_ZONE 设置会覆盖数据库或者租户的默认设置。

当调整主副本分布策略后，OceanBase 会很快发起在线主备切换。

- 查看租户的 PRIMARY_ZONE 和 LOCALITY 设置。

```
select tenant_id,tenant_name,primary_zone,locality from gv$tenant;

# 输出:
+-----------+-------------+------------------+-------------------------------------------+
| tenant_id | tenant_name | primary_zone     | locality                                  |
+-----------+-------------+------------------+-------------------------------------------+
|         1 | sys         | zone1;zone2,zone3 | FULL{1}@zone1, FULL{1}@zone2, FULL{1}@zone3 |
|      1002 | obmysql     | RANDOM           | FULL{1}@zone1, FULL{1}@zone2, FULL{1}@zone3 |
+-----------+-------------+------------------+-------------------------------------------+
2 rows in set (0.011 sec)
```

  PRIMARY_ZONE 的取值可以有很多组合:

  - `zone1`：等同于 `zone1; zone2, zone3`。即主副本优先分布在 `zone1` 的节点上。如果 `zone1` 的节点不可用，会选举 `zone2` 或 `zone3` 中的节点。
  - `zone1; zone2; zone3`：即主副本优先分布在 `zone1` 的节点上。如果 `zone1` 的节点不可用，优先考虑 `zone2` 的节点，其次才是考虑 `zone3` 的节点。
  - `RANDOM`：即主副本随机分布。赋值语句： `set primary_zone =RANDOM;`，不需要单引号。
  - `DEFAULT`：即清空 PRIMARY_ZONE 设置，遵守默认值，只能设置数据库和表分组、表的 PRIMARY_ZONE 为 `DEFAULT`。

  LOCALITY 设置是描述每个 ZONE 里的副本类型，默认为全功能副本（FULL）。其他副本类型还有：日志副本（LOGONLY）、只读副本（READONLY）。

- 查看数据库的 PRIMARY_ZONE 设置

  可以通过查看数据库的定义获取。

```
MySQL [test]> alter database test primary_zone='zone2';
Query OK, 0 rows affected (0.310 sec)

MySQL [test]> show create database test;
+----------+-----------------------------------------------------------------------------------------
| Database | Create Database
+----------+-----------------------------------------------------------------------------------------
| test     | CREATE DATABASE `test` DEFAULT CHARACTER SET = utf8mb4 REPLICA_NUM = 3 PRIMARY_ZONE
+----------+-----------------------------------------------------------------------------------------
1 row in set (0.016 sec)
```

- 查看表分组和表的 PRIMARY_ZONE 设置

```
MySQL [test]> alter tablegroup tpcc_group primary_zone=RANDOM;
Query OK, 0 rows affected (0.427 sec)

MySQL [test]> show create tablegroup tpcc_group\G
*************************** 1. row ***************************
       Tablegroup: tpcc_group
```

```
Create Tablegroup: CREATE TABLEGROUP IF NOT EXISTS `tpcc_group`  PRIMARY_ZONE = RANDOM BINDING =
  partition by hash partitions 3

1 row in set (0.010 sec)

MySQL [test]> alter table t1 primary_zone='zone3;zone2;zone1';
Query OK, 0 rows affected (0.405 sec)

MySQL [test]> show create table t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `c1` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) AUTO_INCREMENT = 1000001 DEFAULT CHARSET = utf8mb4 ROW_FORMAT = COMPACT COMPRESSION = 'zstd_1.
1 row in set (0.082 sec)
```

- 查看分区主备副本切换事件

  调整 PRIMARY_ZONE 可能引起分区副本主备角色切换，这个是在线切换，立即生效，对业务影响非常小。可以通过以下 SQL 查看切换历史:

```
select t.table_name, h.partition_idx, h.gmt_create, h.svr_ip, h.event, h.leader, h.info
from `__all_virtual_election_event_History` h join __all_virtual_table t  on (h.table_id=t.table
_id)
where t.table_id >> 40 = 1002
order by h.gmt_create desc
limit 10;
```

  输出:

```
+------------------+--------------+----------------------------+---------------+---------------
| table_name       | partition_idx | gmt_create                 | svr_ip        | event
+------------------+--------------+----------------------------+---------------+---------------
| bmsql_order_line |             1 | 2021-10-03 16:31:05.822946 | 172.20.249.52 | leader takevoe
| bmsql_order_line |             1 | 2021-10-03 16:31:05.822150 | 172.20.249.49 | leader takevoe
| bmsql_order_line |             1 | 2021-10-03 16:31:05.821211 | 172.20.249.51 | leader revoke
| bmsql_order_line |             1 | 2021-10-03 16:31:05.821211 | 172.20.249.51 | leader takevoe
| nation           |             0 | 2021-10-03 16:31:05.817790 | 172.20.249.51 | leader takevoe
| sbtest7          |             0 | 2021-10-03 16:31:05.817790 | 172.20.249.51 | leader takevoe
| sbtest1          |             0 | 2021-10-03 16:31:05.817790 | 172.20.249.51 | leader takevoe
| supplier         |             3 | 2021-10-03 16:31:05.817790 | 172.20.249.51 | leader takevoe
| sbtest8          |             0 | 2021-10-03 16:31:05.817790 | 172.20.249.51 | leader takevoe
| customer         |             0 | 2021-10-03 16:31:05.817790 | 172.20.249.51 | leader takevoe
+------------------+--------------+----------------------------+---------------+---------------
10 rows in set (0.107 sec)
```

LDC 设置

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。但当真正多地多中心部署时，任何数据库都会面临异地路由延迟问题。

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。您可以为 OceanBase 集群的每个 ZONE 设置 Region 属性和 IDC 属性，并为 OBProxy 指定 IDC 名称配置项。当 OBProxy 随机发送时，会优先考虑同一个 IDC 或者同一个 Region 的可用节点。

OceanBase 集群节点添加 Region 和 IDC 属性

```
ALTER SYSTEM MODIFY zone zone1 SET region = SHANGHAI;
ALTER SYSTEM MODIFY zone zone1 SET idc = SH_IDC1;

ALTER SYSTEM MODIFY zone zone2 SET region = SHANGHAI;
ALTER SYSTEM MODIFY zone zone2 SET idc = SH_IDC2;

ALTER SYSTEM MODIFY zone zone3 SET region = HANGZHOU;
ALTER SYSTEM MODIFY zone zone3 SET idc = SH_IDC3;
```

一个集群可以有若干 Region，一个 Region 有若干个 ZONE，每个 ZONE 对应一个 IDC（可以重复，如同机房三副本）。Region 和 IDC 不能随意设置，需要保证租户的 PRIMARY_ZONE 里至少有两个 FULL 副本。

```
MySQL [oceanbase]> select * from __all_zone where name in ('region','idc');
+----------------------------+----------------------------+-------+--------+-------+---------+
| gmt_create                 | gmt_modified               | zone  | name   | value | info    |
+----------------------------+----------------------------+-------+--------+-------+---------+
| 2021-09-25 08:19:05.067944 | 2021-10-04 14:28:03.262961 | zone1 | idc    |     0 | idc1    |
| 2021-09-25 08:19:05.067944 | 2021-10-04 14:29:17.004134 | zone1 | region |     0 | region1 |
| 2021-09-25 08:19:05.068993 | 2021-10-04 14:28:12.236866 | zone2 | idc    |     0 | idc2    |
| 2021-09-25 08:19:05.068993 | 2021-10-04 14:29:23.306688 | zone2 | region |     0 | region2 |
| 2021-09-25 08:19:05.070055 | 2021-10-04 14:28:19.560596 | zone3 | idc    |     0 | idc3    |
| 2021-09-25 08:19:05.070055 | 2021-10-04 14:29:30.604319 | zone3 | region |     0 | region3 |
+----------------------------+----------------------------+-------+--------+-------+---------+
6 rows in set (0.012 sec)
```

例如，上述设置就有问题，租户的 PRIMARY_ZONE 设置为任意一个 ZONE 都会报错。

```
MySQL [oceanbase]> alter tenant sys primary_zone='zone1';
ERROR 4179 (HY000): primary zone F type replica not enough in its region not allowed

MySQL [oceanbase]> alter system modify zone zone2 set region = region1;
Query OK, 0 rows affected (0.004 sec)


MySQL [oceanbase]> alter tenant sys primary_zone='zone1';
Query OK, 0 rows affected (0.095 sec)

MySQL [oceanbase]> select tenant_name,primary_zone,locality from __all_tenant;
+-------------+--------------+--------------------------------------------+
| tenant_name | primary_zone | locality                                   |
+-------------+--------------+--------------------------------------------+
| sys         | zone1;zone2  | FULL{1}@zone1, FULL{1}@zone2, FULL{1}@zone3 |
```

```
| obmysql     | RANDOM       | FULL{1}@zone1, FULL{1}@zone2, FULL{1}@zone3 |
+-------------+--------------+---------------------------------------------+
2 rows in set (0.003 sec)
```

将 ZONE2 的 Region 属性和 ZONE1 的 Region 属性调整一致后，即可设置 PRIMARY_ZONE 为 `zone1` 或 `zone2`。并且 `zone3` 不再出现在候选里，因为 `zone3` 的 `region3` 里只有一个 FULL 副本，不具备设置为 PRIMARY_ZONE 的资格。

开启 LDC 后，PRIMARY_ZONE 设置为 `RANDOM` 还可能会有限制。

```
MySQL [oceanbase]> alter tenant obmysql primary_zone=RANDOM;
ERROR 1235 (0A000): tenant primary zone span regions when GTS is on not supported
```

登录业务租户查看。

```
MySQL [test]> show global variables like 'ob_timestamp_service';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| ob_timestamp_service | GTS   |
+----------------------+-------+
1 row in set (0.008 sec)

MySQL [test]> set global ob_timestamp_service = LTS;
Query OK, 0 rows affected (0.120 sec)
```

再次登录 sys 租户修改。

```
MySQL [oceanbase]> alter tenant obmysql primary_zone=RANDOM;
Query OK, 0 rows affected (0.078 sec)

MySQL [oceanbase]> select tenant_name,primary_zone,locality from __all_tenant;
+-------------+------------------+---------------------------------------------+
| tenant_name | primary_zone     | locality                                    |
+-------------+------------------+---------------------------------------------+
| sys         | zone1;zone2,zone3 | FULL{1}@zone1, FULL{1}@zone2, FULL{1}@zone3 |
| obmysql     | RANDOM           | FULL{1}@zone1, FULL{1}@zone2, FULL{1}@zone3 |
+-------------+------------------+---------------------------------------------+
2 rows in set (0.014 sec)
```

此时虽然设置成功，不过由于时间服务改用了 LTS，不支持跨节点的一致性读快照，会遇到以下问题。

```
MySQL [tpccdb]> select /*+ read_consistency(strong) query_timeout(10000000) */ count(*) from
ERROR 1235 (0A000): strong consistency across distributed node not supported
MySQL [tpccdb]>
MySQL [tpccdb]> select /*+ read_consistency(weak) */ count(*) from bmsql_warehouse;
+----------+
| count(*) |
+----------+
|       10 |
+----------+
```

```
1 row in set (0.269 sec)
```

LTS 服务下不支持跨节点的分布式查询，这个是业务不能接受的。所以反过来，当 OceanBase 集群有多个 Region 的时候，就不能设置 PRIMARY_ZONE 为 `RANDOM`。

OBProxy 的 IDC 属性

OBProxy 启动时可以指定参数 `proxy_idc_name` 为具体的 IDC，也可以启动后修改参数值。

```
MySQL [(none)]> show proxyconfig like 'proxy_idc_name';
+----------------+-------+--------------------------------------------------------------------------
| name           | value | info
+----------------+-------+--------------------------------------------------------------------------
| proxy_idc_name |       | idc name for proxy ldc route. If is empty or invalid, treat as do not
+----------------+-------+--------------------------------------------------------------------------
1 row in set (0.008 sec)

MySQL [(none)]> alter proxyconfig set proxy_idc_name = 'idc1';
Query OK, 0 rows affected (0.044 sec)

MySQL [(none)]> show proxyconfig like 'proxy_idc_name';
+----------------+-------+--------------------------------------------------------------------------
| name           | value | info
+----------------+-------+--------------------------------------------------------------------------
| proxy_idc_name | idc1  | idc name for proxy ldc route. If is empty or invalid, treat as do not
+----------------+-------+--------------------------------------------------------------------------
1 row in set (0.051 sec)

MySQL [(none)]> show proxyinfo idc;
+-----------------+--------------+----------------+----------------+--------------+-------------
| global_idc_name | cluster_name | match_type     | regions_name   | same_idc     | same_region
+-----------------+--------------+----------------+----------------+--------------+-------------
| idc1            | obce-3zones  | MATCHED_BY_IDC | [[0]"region1"] | [[0]"zone1"] | [[0]"zone2",
+-----------------+--------------+----------------+----------------+--------------+-------------
1 row in set (0.010 sec)
```

OBProxy 路由策略简介

说明

OBProxy 的路由策略非常丰富，本节只做大概的介绍。您仅需了解每条 OBProxy 的路由策略即可。

弱一致性读路由策略

OceanBase 数据库默认是强一致性读，即写后读立即可见（READ AFTER WRITER）。使用强一致性读策略时，OBProxy 会优先路由到访问表的分区的主副本节点上。

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。弱一致性读也可以路由到分区的主副本

和备副本节点，通常有三副本所在节点可以选。

但是开启弱一致性读后，如果 OBServer 和 OBProxy 都开启了 LDC 特性，那么弱一致性读语句的路由策略会改变，弱一致性读语句将按下述顺序进行路由：

1. 同一个机房或者同一个 Region 状态不是合并中（merging）的节点。
2. 合并中的节点。
3. 其他 Region 的不在合并的或在合并的节点。

即 OBProxy 会尽力避开合并中的节点。不过若 OceanBase 集群关闭了轮转合并（参数 `enable_merge_by_turn` 设置为 `false`），合并（major freeze）则是所有节点都开始合并，那么 OBProxy 也就无法避开合并中的节点。

还有一些 SQL 不是访问数据，而是查看或者设置变量值等。如：

```
set @@autocommit=off
show variables like 'autocommit';
```

这类 SQL 的路由策略则是随机路由。在随机路由策略中，如果 OBServer 和 OBProxy 开启了 LDC 设置，也会按照上文的路由顺序进行路由。

弱一致性读通常用在读写分离场景中。不过当租户 PRIMARY_ZONE 为 `RANDOM` 时，租户的所有分区的主副本也是分散在所有 ZONE 下，这时弱一致性读备副本的意义也不是很大。

但是，如果使用了只读副本，只读副本设置为独立的 IDC，然后单独的 OBProxy 设置为同一个 IDC，则这个 OBProxy 可以用于只读副本的路由。

强一致性读路由策略

路由策略很多，这里针对部分路由策略由简单到复杂进行列举。

- 强一致性读路由策略将 SQL 路由到访问表的分区的主副本所在节点。这一条理解起来比较简单，但实际 SQL 情形很复杂。
- 如果 SQL 访问了两个表，会依据第一个表及其条件判断出该分区主副本节点。如果无法判断，就随机发。所以 SQL 里多表连接时，表的前后顺序对路由策略是有影响的，间接对性能有影响。

  注意

  如果要判断的表是分区表，会判断条件是否是分区键等值条件。如果不是，则不能确定是哪个分区，就随机发到该表的所有分区所在的节点任意一个。

- 如果开启事务了，则事务里开启事务的 SQL 的路由节点会作为事务后面其他 SQL 路由的目标节点，直到事务结束（提交或者回滚）为止。
- 当 SQL 被 OBProxy 路由分配到一个节点上时
  - 如果要访问的数据分区的主副本恰好在那个节点上，SQL 就在该节点执行，这个 SQL 的执行类型是本地

SQL（`plan_type` 为 `1`）。

- 如果要访问的数据分区的主副本不在这个节点上，SQL 会被 OBServer 再次转发。这个 SQL 的执行类型是远程 SQL（`plan_type` 为 `2`）。
- 如果 SQL 执行计划要访问的数据分区是跨越多个节点，则这个 SQL 的执行类型是分布式 SQL（`plan_type` 为 `3`）。

- 如果事务中有复制表的读 SQL，只要 SQL 被路由到的节点上有该复制表的备副本，则该 SQL 可以读取本地备副本，因为复制表的所有备副本跟主副本是强一致的。这个 SQL 的执行类型是本地 SQL。

实际 SQL 类型很复杂，OBProxy 的路由策略也变得很复杂，有时候会出现路由不准的情形。如果不符合设计预期就会产生 BUG，但很可能也是设计如此（`BY DESIGN`）。毕竟当前版本的 OBProxy 只能做简单的 SQL 解析，不像 OBServer 那样做完整的执行计划解析。

当业务 SQL 很多很复杂时，远程 SQL 和分布式 SQL 将会无法避免，这时需主要观察远程 SQL 和分布式 SQL 比例。如果比例很高，整体上业务 SQL 性能都不会很好。此时应尽可能地减少远程 SQL 和分布式 SQL。比如，通过表分组、复制表和 PRIMARY_ZONE 设置等。

7.3 如何管理 OceanBase 数据库连接

由于执行计划跟连接方法也有关，此处先介绍 OceanBase 数据库的连接原理。

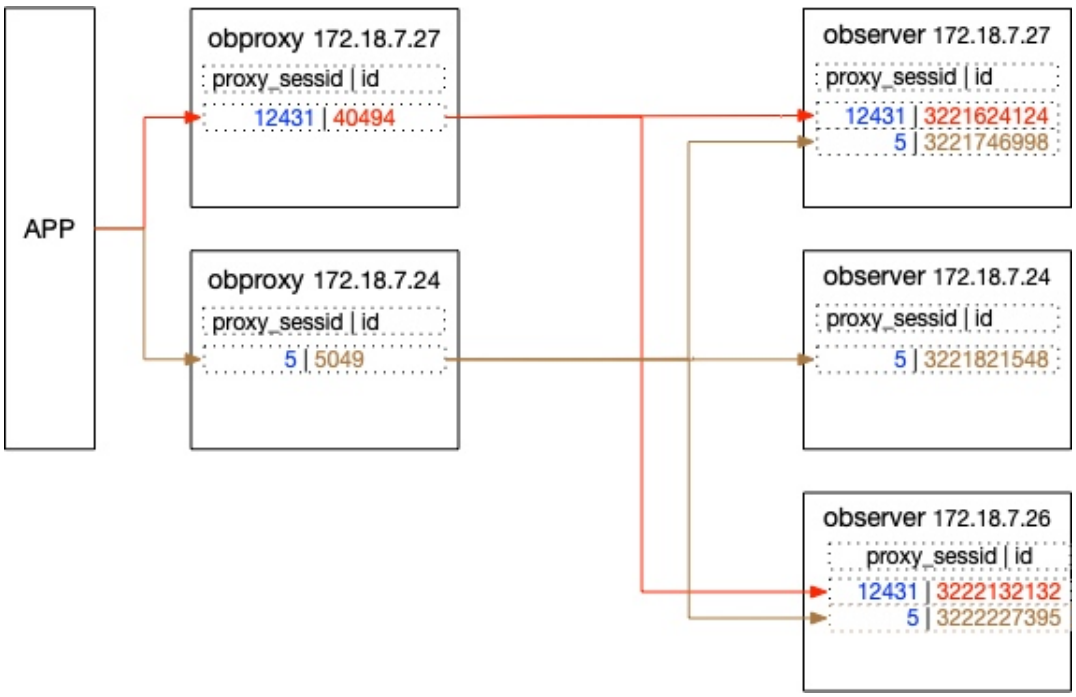
连接 OceanBase 集群有两种方法:

- 直连 OceanBase 集群节点的 2881 端口。任意一个 OBServer 节点，只要没有脱离集群（掉线），都可以用来连接 OceanBase 集群。业务连接 OceanBase 集群不适合这个方法，因为节点比较多，且可能会变化。
- 通过 OBProxy 的 2883 端口连接 OceanBase 集群。OBProxy 和 OceanBase 集群保持联系，能感知集群节点状态变化。您可以使用多个 OBProxy 连接 OceanBase 集群。OBProxy 之间彼此独立工作，互不影响。

OBProxy 连接原理

OBProxy 是个单进程程序，默认监听端口 2883，实现了 MySQL 连接协议。客户端和 OBProxy 建立连接后，访问 OceanBase 集群数据时，OBProxy 会自动判断要访问的数据的主副本在哪个 OBServer 节点上，然后自动和该 OBServer 建立一个长连接。

通常我们把客户端和 OBProxy 建立的连接称为前端连接，把相应的 OBProxy 和后端 OBServer 节点之间的连接称之为后端连接。每个前端连接可能对应 1-N 个后端连接。N 是 OceanBase 租户所在的节点总数（包括备副本所在 OBServer 节点）。不同前端连接对应的后端连接是不复用的。



OBProxy 前端连接数受 OBProxy 参数 `max_connections` 和 `client_max_connections` 限制。

```
MySQL [(none)]> show proxyconfig like '%connection%';
+----------------------------------------+-------+--------------------------------------------
| name                                   | value | info
```

```
+---------------------------------------+-------+-------------------------------------------
| max_connections                       | 60000 | max fd proxy could use
| client_max_connections                | 8192  | client max connections for one obproxy, [0,
| enable_client_connection_lru_disconnect | False | if client connections reach throttle, true i
+---------------------------------------+-------+-------------------------------------------
3 rows in set (0.004 sec)

MySQL [(none)]> alter proxyconfig set client_max_connections=20000;
Query OK, 0 rows affected (0.005 sec)
```

如何管理连接

直连 OBProxy 管理连接

- 查看并直接 KILL 前端连接

  `show processlist` 只能查看当前客户端连接，`show proxysession` 只能查看当前 OBProxy 的连接。

```
MySQL [(none)]> show processlist;
+---------+----------+------+---------------------+------------+-------------+------------------
| Id      | Tenant   | User | Host                | db         | trans_count | svr_session_count
+---------+----------+------+---------------------+------------+-------------+------------------
|       5 | proxysys | root | 172.20.249.50:32108 | NULL       |           0 |                 0
|  524299 | sys      | root | 172.20.249.50:32082 | oceanbase  |           0 |                 1
| 1048583 | obmysql  | root | 172.20.249.50:32120 | sysbenchdb |           0 |                 2
+---------+----------+------+---------------------+------------+-------------+------------------
3 rows in set (0.007 sec)

MySQL [(none)]> show proxysession;
+--------------+---------+-------------+----------+------+---------------------+------------+---
| proxy_sessid | Id      | Cluster     | Tenant   | User | Host                | db         | tr
+--------------+---------+-------------+----------+------+---------------------+------------+---
|            0 |       5 | obce-3zones | proxysys | root | 172.20.249.50:32108 | NULL       |
|           15 |  524299 | obce-3zones | sys      | root | 172.20.249.50:32082 | oceanbase  |
|           19 | 1048583 | obce-3zones | obmysql  | root | 172.20.249.50:32120 | sysbenchdb |
+--------------+---------+-------------+----------+------+---------------------+------------+---
3 rows in set (0.005 sec)

MySQL [(none)]> kill proxysession 1048583;
Query OK, 0 rows affected (0.009 sec)

MySQL [(none)]> show processlist;
+---------+----------+------+---------------------+------------+-------------+------------------
| Id      | Tenant   | User | Host                | db         | trans_count | svr_session_count
+---------+----------+------+---------------------+------------+-------------+------------------
|       5 | proxysys | root | 172.20.249.50:32108 | NULL       |           0 |                 0
|  524299 | sys      | root | 172.20.249.50:32082 | oceanbase  |           0 |                 1
| 1048584 | obmysql  | root | 172.20.249.50:32124 | sysbenchdb |           0 |                 2
+---------+----------+------+---------------------+------------+-------------+------------------
3 rows in set (0.046 sec)

MySQL [(none)]> show proxysession;
+--------------+---------+-------------+----------+------+---------------------+------------+---
```

```
| proxy_sessid | Id      | Cluster     | Tenant   | User | Host                | db         | tr
+--------------+---------+-------------+----------+------+---------------------+------------+---
|            0 |       5 | obce-3zones | proxysys | root | 172.20.249.50:32108 | NULL       |
|           15 |  524299 | obce-3zones | sys      | root | 172.20.249.50:32082 | oceanbase  |
|           20 | 1048584 | obce-3zones | obmysql  | root | 172.20.249.50:32124 | sysbenchdb |
+--------------+---------+-------------+----------+------+---------------------+------------+---
3 rows in set (0.045 sec)

MySQL [(none)]> kill 1048584;
Query OK, 0 rows affected (0.030 sec)
```

`show proxysession` 里的 ID 就是 `show processlist` 中的 ID，可以使用 `kill [id]` 或 `kill proxysession [id]` 杀前端连接。当前端连接被杀后，对应的后端连接也一并中断。

不支持 `kill query` 语法。

- 查看并直接 KILL 后端连接

```
MySQL [(none)]> show processlist;
+---------+----------+------+--------------------+------------+-------------+------------------
| Id      | Tenant   | User | Host               | db         | trans_count | svr_session_count
+---------+----------+------+--------------------+------------+-------------+------------------
|       5 | proxysys | root | 172.20.249.50:32108 | NULL       |           0 |                 0
|  524299 | sys      | root | 172.20.249.50:32082 | oceanbase  |           0 |                 1
| 1048585 | obmysql  | root | 172.20.249.50:32130 | sysbenchdb |           0 |                 3
+---------+----------+------+--------------------+------------+-------------+------------------
3 rows in set (0.074 sec)

MySQL [(none)]> show proxysession ;
+--------------+---------+-------------+----------+------+---------------------+------------+---
| proxy_sessid | Id      | Cluster     | Tenant   | User | Host                | db         | tr
+--------------+---------+-------------+----------+------+---------------------+------------+---
|            0 |       5 | obce-3zones | proxysys | root | 172.20.249.50:32108 | NULL       |
|           15 |  524299 | obce-3zones | sys      | root | 172.20.249.50:32082 | oceanbase  |
|           21 | 1048585 | obce-3zones | obmysql  | root | 172.20.249.50:32130 | sysbenchdb |
+--------------+---------+-------------+----------+------+---------------------+------------+---
3 rows in set (0.200 sec)

MySQL [(none)]> show proxysession attribute 1048585 ;
+---------------------------------+---------------------+----------------+
| attribute_name                  | value               | info           |
+---------------------------------+---------------------+----------------+
| proxy_sessid                    | 21                  | cs common      |
| cs_id                           | 1048585             | cs common      |
| cluster                         | obce-3zones         | cs common      |
| tenant                          | obmysql             | cs common      |
| user                            | root                | cs common      |
| host_ip                         | 172.20.249.50       | cs common      |
| host_port                       | 32130               | cs common      |
| db                              | sysbenchdb          | cs common      |
| total_trans_cnt                 | 0                   | cs common      |
| svr_session_cnt                 | 3                   | cs common      |
| active                          | false               | cs common      |
| read_state                      | MCS_ACTIVE_READER   | cs common      |
| tid                             | 49652               | cs common      |
| pid                             | 49646               | cs common      |
```

```
| idc_name                          |                    | cs common      |
| modified_time                     | 0                  | cs stat        |
| reported_time                     | 0                  | cs stat        |
| hot_sys_var_version               | 1                  | cs var version |
| sys_var_version                   | 3                  | cs var version |
| user_var_version                  | 0                  | cs var version |
| last_insert_id_version            | 0                  | cs var version |
| db_name_version                   | 1                  | cs var version |
| server_ip                         | 172.20.249.52      | last used ss   |
| server_port                       | 2881               | last used ss   |
| server_sessid                     | 3221634194         | last used ss   |
| ss_id                             | 34                 | last used ss   |
| state                             | MSS_KA_CLIENT_SLAVE | last used ss   |
| transact_count                    | 2                  | last used ss   |
| server_trans_stat                 | 0                  | last used ss   |
| hot_sys_var_version               | 1                  | last used ss   |
| sys_var_version                   | 3                  | last used ss   |
| user_var_version                  | 0                  | last used ss   |
| last_insert_id_version            | 0                  | last used ss   |
| db_name_version                   | 1                  | last used ss   |
| is_checksum_supported             | 1                  | last used ss   |
| is_safe_read_weak_supported       | 0                  | last used ss   |
| is_checksum_switch_supported      | 1                  | last used ss   |
| checksum_switch                   | 1                  | last used ss   |
| enable_extra_ok_packet_for_stats  | 1                  | last used ss   |
| server_ip                         | 172.20.249.51      | ss pool [0]    |
| server_port                       | 2881               | ss pool [0]    |
| server_sessid                     | 3222242117         | ss pool [0]    |
| ss_id                             | 36                 | ss pool [0]    |
| state                             | MSS_KA_SHARED      | ss pool [0]    |
| transact_count                    | 2                  | ss pool [0]    |
| server_trans_stat                 | 0                  | ss pool [0]    |
| hot_sys_var_version               | 1                  | ss pool [0]    |
| sys_var_version                   | 2                  | ss pool [0]    |
| user_var_version                  | 0                  | ss pool [0]    |
| last_insert_id_version            | 0                  | ss pool [0]    |
| db_name_version                   | 1                  | ss pool [0]    |
| is_checksum_supported             | 1                  | ss pool [0]    |
| is_safe_read_weak_supported       | 0                  | ss pool [0]    |
| is_checksum_switch_supported      | 1                  | ss pool [0]    |
| checksum_switch                   | 1                  | ss pool [0]    |
| enable_extra_ok_packet_for_stats  | 1                  | ss pool [0]    |
| server_ip                         | 172.20.249.49      | ss pool [1]    |
| server_port                       | 2881               | ss pool [1]    |
| server_sessid                     | 3221895002         | ss pool [1]    |
| ss_id                             | 35                 | ss pool [1]    |
| state                             | MSS_KA_SHARED      | ss pool [1]    |
| transact_count                    | 1                  | ss pool [1]    |
| server_trans_stat                 | 0                  | ss pool [1]    |
| hot_sys_var_version               | 1                  | ss pool [1]    |
| sys_var_version                   | 2                  | ss pool [1]    |
| user_var_version                  | 0                  | ss pool [1]    |
| last_insert_id_version            | 0                  | ss pool [1]    |
| db_name_version                   | 1                  | ss pool [1]    |
| is_checksum_supported             | 1                  | ss pool [1]    |
| is_safe_read_weak_supported       | 0                  | ss pool [1]    |
| is_checksum_switch_supported      | 1                  | ss pool [1]    |
```

```
| checksum_switch                  | 1                   | ss pool [1]    |
| enable_extra_ok_packet_for_stats | 1                   | ss pool [1]    |
+----------------------------------+---------------------+----------------+
73 rows in set (0.081 sec)
```

通过 `show proxysession attribute [id]` 可以查看前端连接对应的后端连接。

- {`proxy_sessid`、`cs_id`、`host_ip`、`host_port`} 为前端连接元组。其中 `cs_id` 是 OBProxy 内部标识的前端连接（客户端连接）的 ID 号，跟 `show processlist` 的 ID 列一致。
- {`server_ip`、`server_port`、`server_sessid`、`ss_id`} 组成后端连接元组。其中 `ss_id` 是 OBProxy 内部标识的后端连接（OBProxy 跟 OBServer 连接）的 ID 号。`server_sessid` 是 OBServer 上的客户端连接 ID。

以下示例指定了前端连接标识和后端连接标识，可以针对性的杀后端连接。

```
MySQL [(none)]> kill proxysession 1048585 35;
Query OK, 0 rows affected (0.002 sec)

MySQL [(none)]> show proxysession attribute 1048585 ;
+----------------------------------+---------------------+----------------+
| attribute_name                   | value               | info           |
+----------------------------------+---------------------+----------------+
| proxy_sessid                     | 21                  | cs common      |
| cs_id                            | 1048585             | cs common      |
| cluster                          | obce-3zones         | cs common      |
| tenant                           | obmysql             | cs common      |
| user                             | root                | cs common      |
| host_ip                          | 172.20.249.50       | cs common      |
| host_port                        | 32130               | cs common      |
| db                               | sysbenchdb          | cs common      |
| total_trans_cnt                  | 0                   | cs common      |
| svr_session_cnt                  | 2                   | cs common      |
| active                           | false               | cs common      |
| read_state                       | MCS_ACTIVE_READER   | cs common      |
| tid                              | 49652               | cs common      |
| pid                              | 49646               | cs common      |
| idc_name                         |                     | cs common      |
| modified_time                    | 0                   | cs stat        |
| reported_time                    | 0                   | cs stat        |
| hot_sys_var_version              | 1                   | cs var version |
| sys_var_version                  | 3                   | cs var version |
| user_var_version                 | 0                   | cs var version |
| last_insert_id_version           | 0                   | cs var version |
| db_name_version                  | 1                   | cs var version |
| server_ip                        | 172.20.249.52       | last used ss   |
| server_port                      | 2881                | last used ss   |
| server_sessid                    | 3221634194          | last used ss   |
| ss_id                            | 34                  | last used ss   |
| state                            | MSS_KA_CLIENT_SLAVE | last used ss   |
| transact_count                   | 2                   | last used ss   |
| server_trans_stat                | 0                   | last used ss   |
| hot_sys_var_version              | 1                   | last used ss   |
| sys_var_version                  | 3                   | last used ss   |
| user_var_version                 | 0                   | last used ss   |
```

```
| last_insert_id_version           | 0                  | last used ss   |
| db_name_version                  | 1                  | last used ss   |
| is_checksum_supported            | 1                  | last used ss   |
| is_safe_read_weak_supported      | 0                  | last used ss   |
| is_checksum_switch_supported     | 1                  | last used ss   |
| checksum_switch                  | 1                  | last used ss   |
| enable_extra_ok_packet_for_stats | 1                  | last used ss   |
| server_ip                        | 172.20.249.51      | ss pool [0]    |
| server_port                      | 2881               | ss pool [0]    |
| server_sessid                    | 3222242117         | ss pool [0]    |
| ss_id                            | 36                 | ss pool [0]    |
| state                            | MSS_KA_SHARED      | ss pool [0]    |
| transact_count                   | 2                  | ss pool [0]    |
| server_trans_stat                | 0                  | ss pool [0]    |
| hot_sys_var_version              | 1                  | ss pool [0]    |
| sys_var_version                  | 2                  | ss pool [0]    |
| user_var_version                 | 0                  | ss pool [0]    |
| last_insert_id_version           | 0                  | ss pool [0]    |
| db_name_version                  | 1                  | ss pool [0]    |
| is_checksum_supported            | 1                  | ss pool [0]    |
| is_safe_read_weak_supported      | 0                  | ss pool [0]    |
| is_checksum_switch_supported     | 1                  | ss pool [0]    |
| checksum_switch                  | 1                  | ss pool [0]    |
| enable_extra_ok_packet_for_stats | 1                  | ss pool [0]    |
+----------------------------------+--------------------+----------------+
56 rows in set (0.026 sec)
```

通过 OBProxy 连接 OceanBase 集群管理连接

使用 OBProxy 连接 OceanBase 集群。

```
obclient -h172.20.249.52 -uroot@sys#obce-3zones -P2883 -p0******d -c -A oceanbase
```

通过 OBProxy 连接时，`show processlist` 和 `show proxysession` 只能查看当前 OBProxy 的客户端连接，`show proxysession` 能看到其他 OceanBase 集群的连接（如果这个 OBProxy 还可以为其他集群提供路由服务），`show full processlist` 能看到 OceanBase 集群的全部后端连接，包括通过其他 OBProxy 的连接。

在这个连接里可以针对性的 KILL 当前 OBProxy 的后端连接（不能 KILL 其他 OBProxy 的后端连接），但是前端连接只能 KILL 自己，不能 KILL 其他连接。支持 KILL CONNECTION 和 KILL QUERY。

```
MySQL [oceanbase]> show proxysession;
+--------------+---------+-------------+----------+------+---------------------+------------+---
| proxy_sessid | Id      | Cluster     | Tenant   | User | Host                | db         | tr
+--------------+---------+-------------+----------+------+---------------------+------------+---
|           23 |  524301 | obce-3zones | sys      | root | 172.20.249.50:32214 | oceanbase  |
|           21 | 1048585 | obce-3zones | obmysql  | root | 172.20.249.50:32130 | sysbenchdb |
|            0 |       6 | obce-3zones | proxysys | root | 172.20.249.50:32192 | NULL       |
+--------------+---------+-------------+----------+------+---------------------+------------+---
3 rows in set (0.001 sec)

MySQL [oceanbase]> show full processlist;
+------------+---------+---------+---------------------+------------+---------+------+--------+-
```

```
| Id         | User    | Tenant  | Host                | db         | Command | Time | State  |
+------------+---------+---------+---------------------+------------+---------+------+--------+-
| 3222242175 | root    | obmysql | 172.20.249.52:12714 | sysbenchdb | Sleep   |  204 | SLEEP  |
| 3222242191 | root    | sys     | 172.20.249.52:12736 | oceanbase  | Query   |    0 | ACTIVE |
| 3221895066 | root    | obmysql | 172.20.249.52:4454  | sysbenchdb | Sleep   |  200 | SLEEP  |
| 3221611294 | proxyro | sys     | 172.20.249.52:55222 | oceanbase  | Sleep   |   13 | SLEEP  |
| 3221634194 | root    | obmysql | 172.20.249.52:55346 | sysbenchdb | Sleep   |  219 | SLEEP  |
+------------+---------+---------+---------------------+------------+---------+------+--------+-
5 rows in set (0.026 sec)

MySQL [oceanbase]> kill 3222242191;
ERROR 1317 (70100): Query execution was interrupted
MySQL [oceanbase]> show full processlist;
ERROR 2013 (HY000): Lost connection to MySQL server during query
MySQL [oceanbase]> show full processlist;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    524302
Current database: oceanbase

+------------+---------+---------+---------------------+------------+---------+------+--------+-
| Id         | User    | Tenant  | Host                | db         | Command | Time | State  |
+------------+---------+---------+---------------------+------------+---------+------+--------+-
| 3221659683 | root    | sys     | 172.20.249.52:55448 | oceanbase  | Query   |    0 | ACTIVE |
| 3221611294 | proxyro | sys     | 172.20.249.52:55222 | oceanbase  | Sleep   |    5 | SLEEP  |
| 3221634194 | root    | obmysql | 172.20.249.52:55346 | sysbenchdb | Sleep   |  231 | SLEEP  |
| 3221895066 | root    | obmysql | 172.20.249.52:4454  | sysbenchdb | Sleep   |  211 | SLEEP  |
| 3222242175 | root    | obmysql | 172.20.249.52:12714 | sysbenchdb | Sleep   |  216 | SLEEP  |
+------------+---------+---------+---------------------+------------+---------+------+--------+-
5 rows in set (0.060 sec)

MySQL [oceanbase]> kill 3221634194;
ERROR 1094 (HY000): sqln thread id: 3221634194
```

如果要 KILL 特定后端连接，需先通过后端 ID 找到 `proxysess_id`，然后找到对应的前端连接 ID，再通过命令 `show proxysession attribute [id]` 找到后端连接的 `ss_id`。

```
MySQL [oceanbase]> show proxysession;
+--------------+---------+-------------+----------+------+---------------------+------------+---
| proxy_sessid | Id      | Cluster     | Tenant   | User | Host                | db         | tr
+--------------+---------+-------------+----------+------+---------------------+------------+---
|           24 |  524302 | obce-3zones | sys      | root | 172.20.249.50:32218 | oceanbase  |
|           21 | 1048585 | obce-3zones | obmysql  | root | 172.20.249.50:32130 | sysbenchdb |
|            0 |       6 | obce-3zones | proxysys | root | 172.20.249.50:32192 | NULL       |
+--------------+---------+-------------+----------+------+---------------------+------------+---
3 rows in set (0.019 sec)

MySQL [oceanbase]> show proxysession attribute 1048585;
+---------------------------------+---------------------+----------------+
| attribute_name                  | value               | info           |
+---------------------------------+---------------------+----------------+
| proxy_sessid                    | 21                  | cs common      |
| cs_id                           | 1048585             | cs common      |
| cluster                         | obce-3zones         | cs common      |
| tenant                          | obmysql             | cs common      |
| user                            | root                | cs common      |
```

```
| host_ip                          | 172.20.249.50       | cs common      |
| host_port                        | 32130               | cs common      |
| db                               | sysbenchdb          | cs common      |
| total_trans_cnt                  | 0                   | cs common      |
| svr_session_cnt                  | 3                   | cs common      |
| active                           | false               | cs common      |
| read_state                       | MCS_ACTIVE_READER   | cs common      |
| tid                              | 49652               | cs common      |
| pid                              | 49646               | cs common      |
| idc_name                         |                     | cs common      |
| modified_time                    | 0                   | cs stat        |
| reported_time                    | 0                   | cs stat        |
| hot_sys_var_version              | 1                   | cs var version |
| sys_var_version                  | 3                   | cs var version |
| user_var_version                 | 0                   | cs var version |
| last_insert_id_version           | 0                   | cs var version |
| db_name_version                  | 1                   | cs var version |
| server_ip                        | 172.20.249.49       | last used ss   |
| server_port                      | 2881                | last used ss   |
| server_sessid                    | 3221895066          | last used ss   |
| ss_id                            | 38                  | last used ss   |
| state                            | MSS_KA_CLIENT_SLAVE | last used ss   |
| transact_count                   | 1                   | last used ss   |
| server_trans_stat                | 0                   | last used ss   |
| hot_sys_var_version              | 1                   | last used ss   |
| sys_var_version                  | 3                   | last used ss   |
| user_var_version                 | 0                   | last used ss   |
| last_insert_id_version           | 0                   | last used ss   |
| db_name_version                  | 1                   | last used ss   |
| is_checksum_supported            | 1                   | last used ss   |
| is_safe_read_weak_supported      | 0                   | last used ss   |
| is_checksum_switch_supported     | 1                   | last used ss   |
| checksum_switch                  | 1                   | last used ss   |
| enable_extra_ok_packet_for_stats | 1                   | last used ss   |
| server_ip                        | 172.20.249.51       | ss pool [0]    |
| server_port                      | 2881                | ss pool [0]    |
| server_sessid                    | 3222242175          | ss pool [0]    |
| ss_id                            | 37                  | ss pool [0]    |
| state                            | MSS_KA_SHARED       | ss pool [0]    |
| transact_count                   | 2                   | ss pool [0]    |
| server_trans_stat                | 0                   | ss pool [0]    |
| hot_sys_var_version              | 1                   | ss pool [0]    |
| sys_var_version                  | 3                   | ss pool [0]    |
| user_var_version                 | 0                   | ss pool [0]    |
| last_insert_id_version           | 0                   | ss pool [0]    |
| db_name_version                  | 1                   | ss pool [0]    |
| is_checksum_supported            | 1                   | ss pool [0]    |
| is_safe_read_weak_supported      | 0                   | ss pool [0]    |
| is_checksum_switch_supported     | 1                   | ss pool [0]    |
| checksum_switch                  | 1                   | ss pool [0]    |
| enable_extra_ok_packet_for_stats | 1                   | ss pool [0]    |
| server_ip                        | 172.20.249.52       | ss pool [1]    |
| server_port                      | 2881                | ss pool [1]    |
| server_sessid                    | 3221634194          | ss pool [1]    |
| ss_id                            | 34                  | ss pool [1]    |
| state                            | MSS_KA_SHARED       | ss pool [1]    |
| transact_count                   | 3                   | ss pool [1]    |
```

```
| server_trans_stat                | 0                   | ss pool [1]    |
| hot_sys_var_version              | 1                   | ss pool [1]    |
| sys_var_version                  | 3                   | ss pool [1]    |
| user_var_version                 | 0                   | ss pool [1]    |
| last_insert_id_version           | 0                   | ss pool [1]    |
| db_name_version                  | 1                   | ss pool [1]    |
| is_checksum_supported            | 1                   | ss pool [1]    |
| is_safe_read_weak_supported      | 0                   | ss pool [1]    |
| is_checksum_switch_supported     | 1                   | ss pool [1]    |
| checksum_switch                  | 1                   | ss pool [1]    |
| enable_extra_ok_packet_for_stats | 1                   | ss pool [1]    |
+----------------------------------+---------------------+----------------+
73 rows in set (0.001 sec)

MySQL [oceanbase]> kill proxysession 1048585 34 ;
Query OK, 0 rows affected (0.010 sec)
```

客户端感知不到后端连接被 KILL。再次查询时 OBProxy 会自动建立新的后端连接。

直连 OBServer 管理连接

在 OBProxy 的连接里查看后端连接的命令是 `show full processlist`，输出的 `Id` 列就是 OBServer 里客户端连接 ID。可以在 OBServer 内部直接使用 KILL QUERY 或 KILL CONNECTION。

```
obclient -h172.20.249.52 -uroot@sys#obce-3zones -P2883 -p0******d -c -A oceanbase

MySQL [oceanbase]> show full processlist;
+------------+---------+---------+---------------------+------------+---------+------+--------+-
| Id         | User    | Tenant  | Host                | db         | Command | Time | State  |
+------------+---------+---------+---------------------+------------+---------+------+--------+-
| 3221659683 | root    | sys     | 172.20.249.52:55448 | oceanbase  | Query   |    0 | ACTIVE |
| 3221611294 | proxyro | sys     | 172.20.249.52:55222 | oceanbase  | Sleep   |   16 | SLEEP  |
| 3221666993 | root    | obmysql | 127.0.0.1:45726     | sysbenchdb | Sleep   |    3 | SLEEP  |
| 3221663293 | root    | obmysql | 172.20.249.52:55458 | sysbenchdb | Sleep   |  281 | SLEEP  |
| 3221895066 | root    | obmysql | 172.20.249.52:4454  | sysbenchdb | Sleep   |  291 | SLEEP  |
| 3222242175 | root    | obmysql | 172.20.249.52:12714 | sysbenchdb | Sleep   |  285 | SLEEP  |
+------------+---------+---------+---------------------+------------+---------+------+--------+-
6 rows in set (0.010 sec)
```

说明:

- `Id`：后端连接的客户端 ID。整个 OceanBase 集群内部唯一。
- `Host`：前端连接的客户端。

  `IP:PORT` 发起数据库连接的客户端信息。
- `Ip`：后端连接的服务端 IP。即 OBServer 的 IP。
- `Proxy_sessid`：前端连接的 OBProxy 内部 ID，该 ID 在 OBProxy 内部唯一，不同 OBProxy 的 `Proxy_sessid` 会重复。如果为 NULL，表示是直连 OBServer 建立的连接。

如果您直连到 OBServer，执行以上命令，查询结果会有点不一样。

```
mysql -h127.1 -P2881 -uroot@obmysql -p1******6 -c -A sysbenchdb

mysql> show processlist;
+------------+------+---------------------+------------+---------+------+--------+--------------
| Id         | User | Host                | db         | Command | Time | State  | Info
+------------+------+---------------------+------------+---------+------+--------+--------------
| 3222242175 | root | 172.20.249.52:12714 | sysbenchdb | Sleep   |  312 | SLEEP  | NULL
| 3221895066 | root | 172.20.249.52:4454  | sysbenchdb | Sleep   |  318 | SLEEP  | NULL
| 3221666993 | root | 127.0.0.1:45726     | sysbenchdb | Query   |    0 | ACTIVE | show processl
| 3221663293 | root | 172.20.249.52:55458 | sysbenchdb | Sleep   |  308 | SLEEP  | NULL
+------------+------+---------------------+------------+---------+------+--------+--------------
4 rows in set (0.05 sec)
```

说明：`Host` 客户端连接信息（`IP:PORT`），通常为 OBProxy 地址或者发起连接的客户端。

```
mysql> show full processlist;
+------------+------+---------+---------------------+------------+---------+------+--------+----
| Id         | User | Tenant  | Host                | db         | Command | Time | State  | Inf
+------------+------+---------+---------------------+------------+---------+------+--------+----
| 3221666993 | root | obmysql | 127.0.0.1:45726     | sysbenchdb | Query   |    0 | ACTIVE | sho
| 3221663293 | root | obmysql | 172.20.249.52:55458 | sysbenchdb | Sleep   |  312 | SLEEP  | NUL
| 3221895066 | root | obmysql | 172.20.249.52:4454  | sysbenchdb | Sleep   |  321 | SLEEP  | NUL
| 3222242175 | root | obmysql | 172.20.249.52:12714 | sysbenchdb | Sleep   |  315 | SLEEP  | NUL
+------------+------+---------+---------------------+------------+---------+------+--------+----
4 rows in set (0.00 sec)

mysql> show proxysession ;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to y
```

说明：

- `Host`：前端连接的客户端，即发起连接的客户端信息（`IP:PORT`）。通常为 OBProxy 地址或者发起连接的客户端。
- `Ip`：后端连接的服务端地址，通常为 OBServer 地址。
- `Id`：后端连接的 ID 标识，整个 OceanBase 集群内部唯一。可以直接 KILL QUERY 或 KILL CONNECTION。

直连 OBServer 不支持 OBProxy 的管理命令。

```
mysql> kill 3221663293;
Query OK, 0 rows affected (0.02 sec)

mysql> kill 3221895066;
Query OK, 0 rows affected (0.14 sec)

mysql> kill 3222242175;
Query OK, 0 rows affected (0.02 sec)
```

跟前面一样，如果后端连接被 KILL，客户端重新查询时，OBProxy 将自动和 OBServer 建立后端连接。

视图 __all_virtual_processlist

视图 `__all_virtual_processlist` 能查看到 OceanBase 集群的所有连接，包括通过 OBProxy 连接的后端连接以及直连集群 OBServer 节点的连接。

```
SELECT id, host, command, sql_id, time, state, info, svr_ip, proxy_sessid, user_client_ip, trans_id ,t
hread_id, trace_id
FROM __all_virtual_processlist
where 1=1
;
```

输出:

```
+------------+--------------------+---------+----------------------------------+------+--------
| id         | host               | command | sql_id                           | time | state
+------------+--------------------+---------+----------------------------------+------+--------
| 3222242518 | 172.20.249.52:19440 | Query   | 17115A49B6D58958854D9B2E58CB821A |    0 | ACTIVE
FROM __all_virtual_processlist
where 1=1 | 172.20.249.51 |           26 | 172.20.249.50  |                   0 |     19240 | Y
| 3222242521 | 172.20.249.52:19442 | Sleep   |                                  |  458 | SLEEP
| 3222242532 | 172.20.249.49:48664 | Sleep   |                                  | 1167 | SLEEP
| 3221895453 | 172.20.249.52:11250 | Sleep   |                                  |   13 | SLEEP
| 3221895423 | 172.20.249.52:11218 | Sleep   |                                  | 1024 | SLEEP
| 3221611294 | 172.20.249.52:55222 | Sleep   |                                  |    9 | SLEEP
| 3221591229 | 172.20.249.49:11510 | Sleep   |                                  | 1226 | SLEEP
| 3221594049 | 172.20.249.52:62192 | Sleep   |                                  |  998 | SLEEP
| 3221583881 | 172.20.249.52:62154 | Sleep   |                                  |  384 | SLEEP
| 3221591227 | 172.20.249.49:11512 | Sleep   |                                  |    4 | SLEEP
| 3221603549 | 127.0.0.1:52488     | Sleep   |                                  |  324 | SLEEP
+------------+--------------------+---------+----------------------------------+------+--------
11 rows in set (0.011 sec)
```

这个视图的结果跟命令 `show full processlist` 的结果基本一致。视图说明如下:

- `proxy_sessid`: 如果为 `NULL`，表示直连 OBServer 的连接；如果不为 NULL，表示通过 OBProxy 建立的连接。
- `host`: 连接的客户端。通常是 OBProxy 地址，如果是直连，则为发起连接的实际客户端地址。
- `svr_ip`: 连接的服务端。通常是 OBServer 地址。
- `user_client_ip`: 是连接的实际客户端地址，即发起连接的客户端地址。
- `state`: 后端连接的状态，通常状态是 `Sleep`。如果连接正在执行 SQL，状态是 `Active`。
- `time`: 后端连接当前状态持续时间，单位为秒。如果状态发生变化，就重新计时。
- `info`: 连接正在执行的 SQL。如果 SQL 很快，则很难捕捉到。
- `sql_id`: 连接正在执行的 SQL 的 SQLID，可以根据 SQLID 去查执行计划。

- `trans_id`：连接的事务的 TRANS_ID，这个不准确，仅供参考。如果连接开启了事务，这个是真正的事务 ID。事务提交之后，在开启新事务之前，连接的 TRANS_ID 不一定会变化。
- `trace_id`：连接使用的 TRACE_ID，可以在 OBServer 的运行日志中根据 TRACE_ID 追踪相关日志。

分析连接的事务超时问题

事务有两个超时时间，分别通过参数 `ob_trx_idle_timeout` 和 `ob_trx_timeout` 控制。前者是事务空闲超时，后者是事务未提交超时。

为了研究这两个参数的影响，以下示例有意把两个参数的值差异拉大。

事务空闲超时

客户端 1 查看空闲超时时间。事务空闲超时设置为 120 秒，事务未提交超时设置为 1000 秒，前者先超时。事务空闲实际超时时间在 [120 s, 120+10 s) 。

```
set global ob_trx_idle_timeout=120000000;
set global ob_trx_timeout=1000000000;

show global variables where variable_name in ('ob_trx_idle_timeout','ob_trx_timeout');
+---------------------+------------+
| Variable_name       | Value      |
+---------------------+------------+
| ob_trx_idle_timeout | 120000000  |
| ob_trx_timeout      | 1000000000 |
+---------------------+------------+
2 rows in set (0.040 sec)
```

客户端 2 新建连接开启事务。

```
MySQL [test]> begin; update t1 set c1=now() where id=3;
Query OK, 0 rows affected (0.002 sec)

Query OK, 1 row affected (0.023 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MySQL [test]> select * from t1;
+----+---------------------+
| id | c1                  |
+----+---------------------+
|  1 | 2021-10-03 09:22:20 |
|  2 | 2021-09-29 21:16:05 |
|  3 | 2021-10-03 10:41:59 |
|  4 | 2021-09-29 21:16:06 |
|  5 | 2021-09-29 21:16:06 |
|  6 | 2021-09-29 21:16:18 |
|  7 | 2021-10-03 09:35:18 |
|  8 | 2021-09-29 21:16:19 |
+----+---------------------+
8 rows in set (0.029 sec)
```

客户端 2 停止操作一段时间。

客户端 1 查看连接状态。

```
MySQL [oceanbase]> SELECT id, host, command, sql_id, time, state, info, svr_ip, proxy_sessid, user_cli
ent_ip, trans_id ,thread_id, trace_id   FROM __all_virtual_processlist  where 1=1 and id=3222242958 or
der by id;
+------------+--------------------+---------+--------+------+-------+------+---------------+---------
-----+---------------+--------------------+-----------+----------+
| id         | host               | command | sql_id | time | state | info | svr_ip        | proxy_se
ssid | user_client_ip | trans_id           | thread_id | trace_id |
+------------+--------------------+---------+--------+------+-------+------+---------------+---------
-----+---------------+--------------------+-----------+----------+
| 3222243128 | 172.20.249.52:19954 | Sleep   |        |  121 | SLEEP | NULL | 172.20.249.51
|           45 | 172.20.249.50  | 6678987957559799424 |         0 | NULL     |
+------------+--------------------+---------+--------+------+-------+------+---------------+---------
-----+---------------+--------------------+-----------+----------+
1 row in set (0.007 sec)
```

`time` 列是连接空闲时间。

观察客户端 2 连接。

```
MySQL [test]> select * from t1;
ERROR 6002 (25000): transaction needs rollback
MySQL [test]>
```

可见，OceanBase 3.1 版本后对于事务空闲超时的处理行为变为不中断连接，而是提示事务要回滚。此时不管使用 COMMIT 还是 ROLLBACK，都可以清理事务状态。

示例:

```
MySQL [test]> commit;
ERROR 6002 (40000): Transaction rollbacked
MySQL [test]> select * from t1;
+----+---------------------+
| id | c1                  |
+----+---------------------+
|  1 | 2021-10-03 09:22:20 |
|  2 | 2021-09-29 21:16:05 |
|  3 | 2021-09-29 21:16:05 |
|  4 | 2021-09-29 21:16:06 |
|  5 | 2021-09-29 21:16:06 |
|  6 | 2021-09-29 21:16:18 |
|  7 | 2021-10-03 09:35:18 |
|  8 | 2021-09-29 21:16:19 |
+----+---------------------+
8 rows in set (0.003 sec)
```

注意

不管是哪种超时，连接事务所持有的锁都会释放，只是事务的状态需要客户端主动清理一下。对于应用而言，就是需要捕捉事务超时报错，然后发起 ROLLBACK 。

事务未提交超时

客户端 1 查看事务未提交超时时间。

未提交超时设置为 100 秒，事务空闲超时设置为 1200 秒。前者先超时，事务未提交实际超时时间会在一个范围内 [100s, 100+10 s) 。

```
set global ob_trx_idle_timeout=1200000000;
set global ob_trx_timeout=100000000;

show variables where variable_name in ('ob_trx_idle_timeout','ob_trx_timeout');
+---------------------+------------+
| Variable_name       | Value      |
+---------------------+------------+
| ob_trx_idle_timeout | 1200000000 |
| ob_trx_timeout      | 100000000  |
+---------------------+------------+
2 rows in set (0.003 sec)
```

客户端 2 新建连接开启事务。

```
MySQL [test]> begin; update t1 set c1=now() where id=3;
Query OK, 0 rows affected (0.001 sec)

Query OK, 1 row affected (0.005 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

客户端 2 停止操作一段时间。

```
MySQL [test]> select * from t1;
ERROR 4012 (25000): Transaction is timeout
```

此时连接处于事务超时状态。需要用 COMMIT 或者 ROLLBACK 清除状态。下面示例使用 COMMIT 清除状态，但是事务修改早已经回滚了。

```
MySQL [test]> commit;
ERROR 4012 (25000): Transaction is timeout
MySQL [test]> select * from t1;
+----+---------------------+
| id | c1                  |
+----+---------------------+
|  1 | 2021-10-03 09:22:20 |
|  2 | 2021-09-29 21:16:05 |
|  3 | 2021-09-29 21:16:05 |
|  4 | 2021-09-29 21:16:06 |
|  5 | 2021-09-29 21:16:06 |
|  6 | 2021-09-29 21:16:18 |
```

```
|  7 | 2021-10-03 09:35:18 |
|  8 | 2021-09-29 21:16:19 |
+----+---------------------+
8 rows in set (0.014 sec)
```

注意

不管是哪种超时，连接事务所持有的锁都会释放，只是事务的状态需要客户端主动清理。对于应用而言，就是需要捕捉事务超时报错，然后发起 ROLLBACK。

7.4 如何分析 SQL 审计视图

SQL 审计视图可以查看在 OceanBase 里执行过的所有 SQL，不管是成功的还是失败的。这对开发同学了解自己的业务 SQL 和定位问题细节非常有帮助。

SQL 审计视图概述

SQL 审计视图 `gv$sql_audit` 是虚拟表，是内存中一个 FIFO 队列。

功能的开启和数据大小是通过下面的 OceanBase 集群参数控制的。

参数名	参数值	参数含义
enable_sql_audit	TRUE	指定是否开启 SQL 审计。默认 `TRUE` 是开启。`FALSE` 是关闭。
sql_audit_queue_size	10000000	SQL 审计视图里最大记录数。
sql_audit_memory_limit	3G	SQL 审计视图内存最大大小。

SQL 审计能保留的数据大小跟租户内存资源的大小也有关系，通常不会特别大。建议自行实时将 SQL 审计视图的数据抽取走，然后做二次分析。

您可以在租户里开启或关闭 SQL 审计功能。

```
set global ob_enable_sql_audit = on;
```

视图列定义如下:

列名	含义
SVR_IP	SQL 执行的 OBServer 节点 IP
SVR_PORT	SQL 执行的 OBServer 节点端口
REQUEST_ID	唯一标识
TRACE_ID	该 SQL 的 TRACE_ID 信息，在 OBServer 节点日志里可以关联查询相关日志
SID	SQL 执行的 OBServer 节点上的会话 ID
CLIENT_IP	该 SQL 执行的客户端 IP，通常是 OBProxy IP
TENANT_ID	该 SQL 执行的租户 ID
TENANT_NAME	该 SQL 执行的租户名称
USER_NAME	该 SQL 执行的租户内部用户名
	该 SQL 执行的实际客户端 IP

USER_CLIENT_IP	
DB_ID	该 SQL 执行的数据库 ID
DB_NAME	该 SQL 执行的数据库名称
SQL_ID	该 SQL 的 SQL_ID
QUERY_SQL	该 SQL 的文本，如果太长会截断
PLAN_ID	该 SQL 的执行计划 ID
AFFECTED_ROWS	该 SQL 的写影响行数
RETURN_ROWS	该 SQL 的返回行数
PARTITION_CNT	该 SQL 访问的分区数量
RET_CODE	该 SQL 的返回代码
EVENT	该 SQL 的主要等待事件
PLAN_TYPE	该 SQL 的执行计划类型 1：本地 SQL；2：远程 SQL；3：分布式 SQL
IS_HIT_PLAN	是否命中执行计划
REQUEST_TIME	该 SQL 执行时间点（时间戳类型，可通过 usec_to_time 转换为可读时间格式）
ELAPSED_TIME	该 SQL 执行总耗时
NET_TIME	该 SQL 执行网络消耗时间
QUEUE_TIME	该 SQL 执行内部排队时间
DECODE_TIME	出队列后 decode 时间
GET_PLAN_TIME	该 SQL 执行计划生成时间
EXECUTE_TIME	该 SQL 实际内部执行时间（不包括 CPU 排队时间）

如何查看 SQL 审计视图

以下 SQL 可以在 sys 租户和业务租户里执行。业务租户只能查看属于自己租户的 SQL 数据。

- 查看近期所有 SQL

```
SELECT /*+ read_consistency(weak) ob_querytimeout(100000000) */  substr(usec_to_time(request_tim
e),1,19) request_time_, s.svr_ip, s.client_Ip, s.sid,s.tenant_id, s.tenant_name, s.user_name, s.
db_name, s.query_sql, s.affected_rows, s.return_rows, s.ret_code, s.event, s.elapsed_time, s.que
ue_time, s.execute_time, round(s.request_memory_used/1024/1024/1024,2) req_mem_gb, plan_type, is
_executor_rpc, is_inner_sql, TRANSACTION_HASH, trace_id
FROM gv$sql_audit s
WHERE 1=1  and s.tenant_id = 1002
```

```
and user_name='u_tpcc'
and request_time >= time_to_usec(DATE_SUB(current_timestamp, INTERVAL 30 MINUTE) )
ORDER BY request_time DESC
LIMIT 100;
```

`request_time` 是时间戳，可通过函数 `usec_to_time` 和 `time_to_usec` 与微秒数转换。

- 分析统计近期所有 SQL

  根据 `sql_id` 统计平均总耗时、平均执行时间等。

```
SELECT sql_id, count(*), round(avg(elapsed_time)) avg_elapsed_time, round(avg(execute_time)) avg
_exec_time
FROM gv$sql_audit s
WHERE 1=1  and s.tenant_id = 1002
and user_name='u_tpcc'
and request_time >= time_to_usec(DATE_SUB(current_timestamp, INTERVAL 30 MINUTE) )
GROUP BY sql_id
order by avg_elapsed_time desc
;
```

- 查看报错的 SQL

  `ret_code` 是 SQL 执行报错时的错误码。正常是 `0`，错误码为负数。

```
SELECT /*+ read_consistency(weak) ob_querytimeout(100000000) */  substr(usec_to_time(request_tim
e),1,19) request_time_, s.svr_ip, s.client_Ip, s.sid,s.tenant_id, s.tenant_name, s.user_name, s.
db_name, s.sql_id,  s.query_sql, s.affected_rows, s.return_rows, s.ret_code, s.event, s.elapsed_
time, s.queue_time, s.execute_time, round(s.request_memory_used/1024/1024/1024,2) req_mem_mb, pl
an_type, is_executor_rpc, is_inner_sql, TRANSACTION_HASH, trace_id
FROM gv$sql_audit s
WHERE 1=1  and s.tenant_id = 1002
and user_name='u_tpcc'
and ret_code < 0
and request_time >= time_to_usec(DATE_SUB(current_timestamp, INTERVAL 30 MINUTE) )
ORDER BY request_time DESC
LIMIT 500;
```

- 查看远程 SQL 和分布式 SQL

  `plan_type` 的值有三个：`1` 表示本地 SQL；`2` 表示远程 SQL；`3` 表示分布式 SQL。

```
SELECT /*+ read_consistency(weak) ob_querytimeout(100000000) */  substr(usec_to_time(request_tim
e),1,19) request_time_, s.svr_ip, s.client_Ip, s.sid,s.tenant_id, s.tenant_name, s.user_name, s.
db_name, s.sql_id,  s.query_sql, s.affected_rows, s.return_rows, s.ret_code, s.event, s.elapsed_
time, s.queue_time, s.execute_time, round(s.request_memory_used/1024/1024/1024,2) req_mem_mb, pl
an_type, is_executor_rpc, is_inner_sql, TRANSACTION_HASH, trace_id
FROM gv$sql_audit s
WHERE 1=1  and s.tenant_id = 1002
and user_name='u_tpcc'
and plan_type > 1
and request_time >= time_to_usec(DATE_SUB(current_timestamp, INTERVAL 30 MINUTE) )
ORDER BY request_time DESC
LIMIT 500;
```

远程 SQL 的出现需要结合事务的业务逻辑分析。

7.5 如何诊断和调优 OceanBase SQL 执行计划

如何使用 EXPLAIN 查看理论执行计划

查看执行计划需要一个好的客户端，这样能将注意力集中在执行计划上。推荐使用以下客户端:

- 命令行 OBClient 命令。OBClient 的好处是格式化显示结果，当一行结果太长的时候格式会不美观，这个时候可以让 SQL 以 `\G` 替换 `;` 结尾，结果集会列传行展示，提高可读性。
- OceanBase 官方客户端 ODC。直接选中 SQL，点击右上角的执行计划，会以表格形式展示执行计划。
- 开源的 DBeaver 客户端工具。由于 DBeaver 和 OceanBase 还没有紧密适配，所以只能以 EXPLAIN SQL 的方式查看执行计划，目前这种方式体验最好。

EXPLAIN 说明

EXPLAIN 命令完整的语法是:

```
{EXPLAIN | DESCRIBE | DESC}
[BASIC | OUTLINE | EXTENDED | EXTENDED_NOADDR | PARTITIONS | FORMAT = {TRADITIONAL| JSON}]
{SELECT statement | DELETE statement | INSERT statement | REPLACE statement| UPDATE statement}
```

学习遵从由简入繁原则，先从默认的 `BASIC` 格式入手，不用指定。FORMAT 有 TRADITIONAL 和 JSON 两种格式，默认是 TRADITIONAL 格式，可读性更好，JSON 格式对程序解析比较友好。

先看一个简单的 SQL 执行计划格式:

```
EXPLAIN
SELECT count(*) FROM BMSQL_ITEM;
```

```
obclient> EXPLAIN
    -> SELECT count(*) FROM BMSQL_ITEM \G
*************************** 1. row ***************************
Query Plan: ==============================================
|ID|OPERATOR       |NAME      |EST. ROWS|COST |
----------------------------------------------
|0 |SCALAR GROUP BY|          |1        |78754|
|1 | TABLE SCAN    |BMSQL_ITEM|99995    |59653|
==============================================

Outputs & filters:
-------------------------------------
  0 - output([T_FUN_COUNT(*)]), filter(nil),
      group(nil), agg_func([T_FUN_COUNT(*)])
  1 - output([1]), filter(nil),
      access([BMSQL_ITEM.I_ID]), partitions(p0)
```

```
1 row in set (0.01 sec)
```

整个 EXPLAIN 结果是一行。

在 OceanBase 内部，这个结果是以 JSON 格式存储。示例如下:

```
{
  "ID": 1,
  "OPERATOR": "GROUP BY",
  "NAME": "GROUP BY",
  "EST.ROWS": 1,
  "COST": 4352454,
  "output": [
    "T_FUN_COUNT(*)"
  ],
  "CHILD_1": {
    "ID": 0,
    "OPERATOR": "TABLE SCAN",
    "NAME": "TABLE SCAN",
    "EST.ROWS": 10000000,
    "COST": 2442330,
    "output": [
      "1"
    ]
  }
}
```

这个 JSON 内容描述的是一个树，对普通用户可读性不好。

传统格式的展示分为两部分:

- 第一部分是用表格形式展示执行计划这棵树。每行是一个独立的操作，操作符是 `OPERATOR`，操作有 ID。操作展示可能会缩进。缩进表示是内部操作，可以嵌套。执行顺序遵循由内到外，由上到下。操作符支持的内容也是 SQL 引擎成熟度的一个体现。
  - `OPERATOR`: 表示操作算子的名称，`TABLE SCAN` 是常用操作算子，表示扫描。
  - `NAME`: 表示算子操作的对象。可以是表名、索引名、内部临时视图名。需要注意的是，如果扫描主键，依然展示表名。因为 OceanBase 里的表和索引的本质都是索引组织表，表数据跟主键索引是一个概念。
  - `EST. ROWS`: 执行当前算子输出的行数，跟统计信息有关。OceanBase 里表的统计信息目前只有在集群合并的时候才更新。
  - `COST`: 执行当前算子预估的成本。COST 计算比较复杂，暂时先不深入。
- 第二部分的内容跟第一部分有关，主要是描述第一部分算子的具体信息。
  - `output`: 表示当前算子输出的表达式（包含列）。
  - `filter`: 表示当前算子的过滤表达式，`nil` 表示无。如果当前算子是访问存储层，这个过滤表达式可以下推（push）。

常见执行计划算子

TABLE GET

表示主键直接等值访问，后面接表名。OceanBase 里主键就是表数据。

```
EXPLAIN
SELECT i_id, i_name, I_PRICE FROM BMSQL_ITEM WHERE i_id=10
\G

*************************** 1. row ***************************
Query Plan: ========================================
|ID|OPERATOR |NAME      |EST. ROWS|COST|
----------------------------------------
|0 |TABLE GET|BMSQL_ITEM|1        |53  |
========================================

Outputs & filters:
-------------------------------------
  0 - output([BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_PRICE]), filter(nil),
      access([BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_PRICE]), partitions(p0)

1 row in set (0.00 sec)
```

TABLE SCAN

表示全表扫描、主键扫描或索引扫描。具体看后面的操作对象名是表还是索引。

注意

扫描主键也是表名。

```
CREATE UNIQUE INDEX idx_item_uk ON bmsql_item(i_name);

EXPLAIN extended_noaddr
SELECT i_id, i_name, I_PRICE FROM BMSQL_ITEM WHERE i_name = 'w2uw7BJj5tG5BTlSdfT'
\G

*************************** 1. row ***************************
Query Plan: ========================================================
|ID|OPERATOR  |NAME                   |EST. ROWS|COST|
--------------------------------------------------------
|0 |TABLE SCAN|BMSQL_ITEM(IDX_ITEM_UK)|1        |88  |
========================================================

Outputs & filters:
-------------------------------------
  0 - output([BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_PRICE]), filter(nil),
      access([BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_PRICE]), partitions(p0),
      is_index_back=true,
      range_key([BMSQL_ITEM.I_NAME], [BMSQL_ITEM.shadow_pk_0]), range(w2uw7BJj5tG5BTlSdfT,MIN ;
      range_cond([BMSQL_ITEM.I_NAME = 'w2uw7BJj5tG5BTlSdfT'])
```

```
1 row in set (0.00 sec)
```

说明:

- `range_cond`: 条件中能用于索引的列。
- `range_key`: 索引中能用于过滤的列及其值域。

TOP-N SORT 和 LIMIT

`TOP-N SORT`: 常用的场景排序后可能只返回最大或最小的前 N 条记录。

`LIMIT`: 限制返回的行数。

```
MySQL [tpccdb]> explain extended_noaddr SELECT  i_id, i_name FROM BMSQL_ITEM WHERE i_name LIKE  'w2%'
ORDER BY I_PRICE  DESC limit 10 \G
```

输出:

```
*************************** 1. row ***************************
Query Plan: ==========================================================
|ID|OPERATOR    |NAME                    |EST. ROWS|COST|
----------------------------------------------------------
|0 |LIMIT       |                        |10       |479 |
|1 | TOP-N SORT |                        |10       |478 |
|2 |  TABLE SCAN|bmsql_item(idx_item_uk)|61       |406 |
==========================================================

Outputs & filters:
-------------------------------------
  0 - output([bmsql_item.i_id], [bmsql_item.i_name]), filter(nil), limit(10), offset(nil)
  1 - output([bmsql_item.i_id], [bmsql_item.i_name]), filter(nil), sort_keys([bmsql_item.i_price
  2 - output([bmsql_item.i_name], [bmsql_item.i_id], [bmsql_item.i_price]), filter(nil),
      access([bmsql_item.i_name], [bmsql_item.i_id], [bmsql_item.i_price]), partitions(p0),
      is_index_back=true,
      range_key([bmsql_item.i_name], [bmsql_item.shadow_pk_0]), range(w2,MIN ; w2
□□□□□□□□□□□□□□□□□□□□
      range_cond([(T_OP_LIKE, bmsql_item.i_name, 'w2%', '\\')])

1 row in set (0.002 sec)
```

NESTED-LOOP JOIN

```
EXPLAIN extended_noaddr
SELECT w.W_ID , w.W_NAME ,w.W_CITY ,d.D_ID ,d.D_NAME ,d.D_STATE
FROM BMSQL_WAREHOUSE w JOIN BMSQL_DISTRICT d ON (w.w_id = d.D_W_ID)
WHERE w.W_ID =1
ORDER BY D.D_name
;
```

输出:

```
==================================================
|ID|OPERATOR                   |NAME|EST. ROWS|COST|
--------------------------------------------------
|0 |SORT                       |    |10       |154 |
|1 | NESTED-LOOP JOIN CARTESIAN|    |10       |97  |
|2 |  TABLE GET                |W   |1        |53  |
|3 |  TABLE SCAN               |D   |10       |39  |
==================================================

Outputs & filters:
-------------------------------------
  0 - output([W.W_ID], [W.W_NAME], [W.W_CITY], [D.D_ID], [D.D_NAME], [D.D_STATE]), filter(nil),
  1 - output([W.W_ID], [W.W_NAME], [W.W_CITY], [D.D_ID], [D.D_NAME], [D.D_STATE]), filter(nil),
      conds(nil), nl_params_(nil), inner_get=false, self_join=false, batch_join=false
  2 - output([W.W_ID], [W.W_NAME], [W.W_CITY]), filter(nil),
      access([W.W_ID], [W.W_NAME], [W.W_CITY]), partitions(p0),
      is_index_back=false,
      range_key([W.W_ID]), range[1 ; 1],
      range_cond([W.W_ID = 1])
  3 - output([D.D_ID], [D.D_NAME], [D.D_STATE]), filter(nil),
      access([D.D_ID], [D.D_NAME], [D.D_STATE]), partitions(p0),
      is_index_back=false,
      range_key([D.D_W_ID], [D.D_ID]), range(1,MIN ; 1,MAX),
      range_cond([D.D_W_ID = 1])
```

说明:

- 这个算法的整体性能取决于外部表返回的记录数（循环的次数）和内部表的查询性能。
- 经验是小表作为外部表，大表作为内部表。不过实际并不是按照表的大小区分，而是由过滤条件应用后的结果集大小来定。可以对比下面 SQL 的执行计划。
- 两个表的访问都走了主键索引，主键就是数据，所以执行计划中每个算子后面的名称（`NAME`）都是表名。注意 2 和 3 里的 `is_index_back=false`。
- 通过 2 和 3 的 `range_cond` 可知，这里针对内部表和外部表扫描时都传入了条件 `W_ID=1`。所以这里的 `JOIN` 条件被消除，外层连接就变为笛卡儿积（`CARTESIAN`)。这也是常用的一个优化手段。

MERGE JOIN

`MERGE JOIN` 主要用于两个不是很小或很大的结果集的连接，它们没有有效的过滤条件或者这个条件上没有合适的索引。

`MERGE JOIN` 算法基本分两大阶段:

- 排序，将两个结果集分别按连接字段排序。
- 合并，分别从两个结果集里读取记录，进行比较、遍历等。

如果结果集本来就是有序的，那么第一阶段可以优化。`MERGE JOIN` 可以用于等值运算，也可以用于不等值运算

（小于、大于、小于等于、大于等于）。

`MERGE JOIN` 主要利用数据主键或者索引的有序，此时它的性能有可能会更好。数据量非常大的时候，`MERGE JOIN` 性能并不是很好，要设法规避。

```
EXPLAIN extended_noaddr
SELECT w.W_ID , w.W_NAME ,w.W_CITY ,d.D_ID ,d.D_NAME ,d.D_STATE
FROM BMSQL_WAREHOUSE w JOIN BMSQL_DISTRICT d ON (w.w_id = d.D_W_ID)
ORDER BY d.D_NAME
;
```

输出:

```
======================================
|ID|OPERATOR    |NAME|EST. ROWS|COST |
--------------------------------------
|0 |SORT        |    |1000     |10093|
|1 | MERGE JOIN |    |1000     |1312 |
|2 |  TABLE SCAN|W   |100      |59   |
|3 |  TABLE SCAN|D   |1000     |426  |
======================================

Outputs & filters:
-------------------------------------
  0 - output([W.W_ID], [W.W_NAME], [W.W_CITY], [D.D_ID], [D.D_NAME], [D.D_STATE]), filter(nil),
  1 - output([W.W_ID], [W.W_NAME], [W.W_CITY], [D.D_ID], [D.D_NAME], [D.D_STATE]), filter(nil),
      equal_conds([W.W_ID = D.D_W_ID]), other_conds(nil)
  2 - output([W.W_ID], [W.W_NAME], [W.W_CITY]), filter(nil),
      access([W.W_ID], [W.W_NAME], [W.W_CITY]), partitions(p0),
      is_index_back=false,
      range_key([W.W_ID]), range(MIN ; MAX)always true
  3 - output([D.D_W_ID], [D.D_ID], [D.D_NAME], [D.D_STATE]), filter(nil),
      access([D.D_W_ID], [D.D_ID], [D.D_NAME], [D.D_STATE]), partitions(p0),
      is_index_back=false,
      range_key([D.D_W_ID], [D.D_ID]), range(MIN,MIN ; MAX,MAX)always true
```

说明:

- 两个表在连接条件列上都有主键索引（即数据），所以返回的数据都是有序的，此处使用了 `MERGE JOIN`。
- 在外层再根据排序条件执行排序（`SORT`）。
- `MERGE JOIN` 可以临时排序的大小受集群隐含参数（`_sort_area_size`）限制。

HASH JOIN

`HASH JOIN` 用于两个比较大的结果集之间的连接，通常没有比较好的过滤条件或者过滤条件上没有索引。

```
EXPLAIN extended_noaddr
SELECT c.C_W_ID , c.C_D_ID , c.C_FIRST || ',' || c.C_LAST , c.C_PAYMENT_CNT , c.C_YTD_PAYMENT , o.O_W_
ID , o.O_D_ID ,o.O_OL_CNT
FROM BMSQL_CUSTOMER c JOIN BMSQL_OORDER o ON (c.C_ID = o.O_C_ID)
 WHERE c.C_W_ID = 1 AND c.C_D_ID = 5 AND (c.C_W_ID <> o.O_W_ID OR c.C_D_ID <> o.O_D_ID )
```

```
ORDER BY c.C_W_ID , c.C_D_ID , c.C_LAST , o.O_W_ID , o.O_D_ID
;
```

输出:

```
=======================================
|ID|OPERATOR    |NAME|EST. ROWS|COST    |
---------------------------------------
|0 |SORT        |    |2238162  |45593377|
|1 | HASH JOIN  |    |2238162  |3764643 |
|2 |  TABLE SCAN|C   |2973     |3515    |
|3 |  TABLE SCAN|O   |3019348  |785737  |
=======================================

Outputs & filters:
-------------------------------------
  0 - output([C.C_W_ID], [C.C_D_ID], [(T_OP_CNN, (T_OP_CNN, C.C_FIRST, ?), C.C_LAST)], [C.C_PAYM
  1 - output([C.C_W_ID], [C.C_D_ID], [C.C_FIRST], [C.C_LAST], [C.C_PAYMENT_CNT], [C.C_YTD_PAYMEN
      equal_conds([C.C_ID = O.O_C_ID]), other_conds([C.C_W_ID != O.O_W_ID OR C.C_D_ID != O.O_D_I
  2 - output([C.C_ID], [C.C_W_ID], [C.C_D_ID], [C.C_FIRST], [C.C_LAST], [C.C_PAYMENT_CNT], [C.C_
      access([C.C_ID], [C.C_W_ID], [C.C_D_ID], [C.C_FIRST], [C.C_LAST], [C.C_PAYMENT_CNT], [C.C_
      is_index_back=false,
      range_key([C.C_W_ID], [C.C_D_ID], [C.C_ID]), range(1,5,MIN ; 1,5,MAX),
      range_cond([C.C_W_ID = 1], [C.C_D_ID = 5])
  3 - output([O.O_C_ID], [O.O_W_ID], [O.O_D_ID], [O.O_OL_CNT]), filter(nil),
      access([O.O_C_ID], [O.O_W_ID], [O.O_D_ID], [O.O_OL_CNT]), partitions(p0),
      is_index_back=false,
      range_key([O.O_W_ID], [O.O_D_ID], [O.O_ID]), range(MIN,MIN,MIN ; MAX,MAX,MAX)always true
```

说明:

- 尽管条件 `(c.C_W_ID <> o.O_W_ID OR c.C_D_ID <> o.O_D_ID )` 不在 JOIN 的 ON 条件里，优化器还是能识别出来并自动应用，并且将条件分类为 `equal_conds` 和 `other_conds`。
- `HASH JOIN` 也分外部表和内部表。内部表是 `probe table`，外部表是 `hash table`。

  通常数据库会挑选结果集相对小的表作为外部表，并在连接条件上用哈希函数构建 `hash table`。然后循环遍历 `probe table`，对连接条件列用哈希函数，探测是否在 `hash table` 中存在。

  如果存在，则返回匹配的记录。这个跟 `NESTED-LOOP JOIN` 很类似，不同之处是 `HASH JOIN` 会在连接条件列上用哈希函数，并在内存中构建 `hash table`。
- OceanBase 优化器一次能构建的最大 `hash table` 受内部参数（`_hash_table_size`）限制。如果外部表的结果集比这个大，就需要分多次构建 `hash table`，这个也叫 `multiple pass`，会涉及到一些内存和文件数据交换，以及多次哈希探测，性能相对会下降一些。
- `HASH JOIN` 的细节比较复杂，此处不详细讨论。目前只要能识别出 `HASH JOIN`，以及掌握产生后如何规避 `HASH JOIN` 算法。

SUBPLAN SCAN 和 COUNT

算子 `SUBPLAN SCAN` 跟 `TABLE SCAN` 类似，不同的是:

- `SUBPLAN SCAN` 是从视图（包括内部临时生成的）里读取数据。
- `TABLE SCAN` 是从基表（或者索引）里扫描数据。

还是前面列举的分页查询场景例子。由于在 SELECT 后面增加了一列排序序号（计算出来的），执行计划就多了两步。

```
EXPLAIN extended_noaddr
SELECT i_id, i_name, i_price, i_data , rn
FROM (
 SELECT rownum rn, i_id, i_name, I_PRICE, I_DATA FROM (
  SELECT  i_id, i_name, i_price , i_data FROM BMSQL_ITEM
  WHERE i_name LIKE  'wu%'
  ORDER BY i_price  DESC
 ) t WHERE rownum <= 15
) WHERE rn > 10
;
```

输出:

```
==============================================================
|ID|OPERATOR      |NAME                     |EST. ROWS|COST|
--------------------------------------------------------------
|0 |COUNT         |                         |5        |332 |
|1 | SUBPLAN SCAN |T                        |5        |331 |
|2 |  LIMIT       |                         |5        |330 |
|3 |   TOP-N SORT |                         |15       |328 |
|4 |    TABLE SCAN|BMSQL_ITEM(BMSQL_ITEM_UK)|30       |245 |
==============================================================

Outputs & filters:
-------------------------------------
  0 - output([T.I_ID], [T.I_NAME], [T.I_PRICE], [T.I_DATA], [rownum() + ?]), filter(nil)
  1 - output([T.I_ID], [T.I_NAME], [T.I_PRICE], [T.I_DATA]), filter(nil),
      access([T.I_ID], [T.I_NAME], [T.I_PRICE], [T.I_DATA])
  2 - output([BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_PRICE], [BMSQL_ITEM.I_DATA]),
  3 - output([BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_PRICE], [BMSQL_ITEM.I_DATA]),
  4 - output([BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_PRICE], [BMSQL_ITEM.I_DATA]),
      access([BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_PRICE], [BMSQL_ITEM.I_DATA]),
      is_index_back=true,
      range_key([BMSQL_ITEM.I_NAME], [BMSQL_ITEM.shadow_pk_0]), range(wu,MIN ; wu
□□□□□□□□□□□□□□□□□□
      range_cond([(T_OP_LIKE, BMSQL_ITEM.I_NAME, ?, '\')])
```

上面 `COUNT` 算子是为了兼容 Oracle 的 rownum 功能，作用在于为 rownum 表达式进行自增操作。

在一般场景下，当 SQL 查询含有 rownum 时，SQL 优化器就会在生成计划的时候分配一个 `COUNT` 算子。如果 SELECT 里不涉及到 rownum 值的展示时，`COUNT` 算子也可能会被优化为 `LIMIT` 算子。

```
EXPLAIN extended_noaddr
SELECT * FROM (
 SELECT  i_id, i_name FROM BMSQL_ITEM WHERE i_name LIKE  'w2u%'
```

```
 ORDER BY I_PRICE  DESC
) WHERE rownum < 5
;
```

输出:

```
============================================================
|ID|OPERATOR    |NAME                      |EST. ROWS|COST|
------------------------------------------------------------
|0 |LIMIT       |                          |1        |91  |
|1 | TOP-N SORT |                          |1        |90  |
|2 |  TABLE SCAN|BMSQL_ITEM(BMSQL_ITEM_UK)|1        |88  |
============================================================

Outputs & filters:
-------------------------------------
  0 - output([BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_NAME]), filter(nil), limit(?), offset(nil)
  1 - output([BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_NAME]), filter(nil), sort_keys([BMSQL_ITEM.I_PRICE
  2 - output([BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_PRICE]), filter(nil),
      access([BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_PRICE]), partitions(p0),
      is_index_back=true,
      range_key([BMSQL_ITEM.I_NAME], [BMSQL_ITEM.shadow_pk_0]), range(w2u,MIN ; w2u
□□□□□□□□□□□□□□
      range_cond([(T_OP_LIKE, BMSQL_ITEM.I_NAME, ?, '\')])
```

EXCHANGE IN|OUT REMOTE

首先看要访问表的主副本节点，然后直连另外一个节点。人为构造一个远程执行计划。

- 主键扫描示例

```
explain extended_Noaddr select /*+ test20210405 */ * from bmsql_item where i_id=100\G

obclient: [Warning] Using a password on the command line interface can be insecure.
*************************** 1. row ***************************
Query Plan: ====================================================
|ID|OPERATOR            |NAME      |EST. ROWS|COST|
------------------------------------------------------
|0 |EXCHANGE IN REMOTE  |          |1        |54  |
|1 | EXCHANGE OUT REMOTE|          |1        |53  |
|2 |  TABLE GET         |BMSQL_ITEM|1        |53  |
======================================================

Outputs & filters:
-------------------------------------
0 - output([BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_PRICE], [BMSQL_ITEM.I_DATA], [B
1 - output([BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_PRICE], [BMSQL_ITEM.I_DATA], [B
2 - output([BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_PRICE], [BMSQL_ITEM.I_DATA], [B
access([BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_PRICE], [BMSQL_ITEM.I_DATA], [BMSQL
is_index_back=false,
range_key([BMSQL_ITEM.I_ID]), range[100 ; 100],
range_cond([BMSQL_ITEM.I_ID = 100])
```

  - `Exchange` 算子是分布式场景下，用于线程间进行数据交互的算子。它一般成对出现，数据源端有一个 `out` 算子，目的端会有一个 `in` 算子。
  - 算子 2 是实际取数据的执行计划，`TABLE GET` 是主键访问数据的算子。
  - 算子 1 的 `EXCHANGE OUT REMOTE` 在远端机器上负责读取并传输数据。
  - 算子 0 的 `EXCHANGE IN REMOTE` 在本地节点接收数据。
- 索引扫描示例

```
explain extended_Noaddr select /*+ test20210405 */ * from bmsql_item where i_name LIKE 'Ax%'\G

obclient: [Warning] Using a password on the command line interface can be insecure.
*************************** 1. row ***************************
Query Plan: ============================================================================
|ID|OPERATOR            |NAME                         |EST. ROWS|COST|
----------------------------------------------------------------------------
|0 |EXCHANGE IN REMOTE  |                             |33       |294 |
|1 | EXCHANGE OUT REMOTE|                             |33       |264 |
|2 |  TABLE SCAN        |BMSQL_ITEM(BMSQL_ITEM_IDX1)|33       |264 |
============================================================================

Outputs & filters:
-------------------------------------
0 - output([BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_PRICE], [BMSQL_ITEM.I_DATA], [B
1 - output([BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_PRICE], [BMSQL_ITEM.I_DATA], [B
2 - output([BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_PRICE], [BMSQL_ITEM.I_DATA], [B
    access([BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_ID], [BMSQL_ITEM.I_PRICE], [BMSQL_ITEM.I_DATA], [B
    is_index_back=true,
    range_key([BMSQL_ITEM.I_NAME], [BMSQL_ITEM.I_ID]), range(Ax,MIN ; Ax
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
    range_cond([(T_OP_LIKE, BMSQL_ITEM.I_NAME, ?, '\')])
```

实际上业务都是通过 OBProxy 连接，能正确路由到 OBServer 节点上，很大程度规避了远程执行计划，不过并不能从根本上避免。后面还会举例说明。

主备切换如何影响执行计划

- 当 SQL 包含多表连接时，如果多表的主副本都在同一个 OBServer 节点上或是当前连接的节点，这就是个本地 SQL 执行计划（`plan_type` = 1）。
- 当多表的主副本是分散在不同的 OBServer 节点上时，这就是分布式 SQL 执行计划（`plan_type=3`）。
- 当多表的主副本都在同一个 OBServer 节点上但不是当前连接的节点，这就是一个远程 SQL 执行计划（`plan_type=2`）。

通常我们不直接调整分区主副本的位置，而是通过调整表或表分组、数据库或者租户的 PRIMARY_ZONE 来间接改变表主副本的位置。理论上改变了主副本的位置后，执行计划也会变化。

实际情况下，由于分区主副本位置的变动信息 OBProxy 并不会立即获得，所以有时候观察 EXPLAIN 执行计划也

不会变，这点需要留意。

查看表的主副本位置。

```
SELECT  a.tenant_name, d.database_name,  tg.tablegroup_name ,t.table_name,  t.part_num , t2.part
FROM oceanbase.__all_tenant AS a
 JOIN oceanbase.__all_virtual_database AS d ON ( a.tenant_id = d.tenant_id  )
 JOIN oceanbase.__all_virtual_table AS t  ON (t.tenant_id = d.tenant_id AND t.database_id = d.da
 JOIN oceanbase.__all_virtual_meta_table t2 ON (t.tenant_id = t2.tenant_id AND (t.table_id=t2.ta
 LEFT JOIN oceanbase.__all_virtual_tablegroup AS tg ON (t.tenant_id = tg.tenant_id and  t.tableg
WHERE a.tenant_id IN (1002 ) AND  t.table_type IN (3)
 AND d.database_name = 'tpccdb' and table_name in ('bmsql_item','bmsql_oorder')
ORDER BY t.tenant_id, tg.tablegroup_name, d.database_name, t.table_name, t2.partition_id
;
```

输出:

```
+-------------+---------------+-----------------+--------------+----------+--------------+-------+----
-----------+----------+-------------+-----------+
| tenant_name | database_name | tablegroup_name | table_name   | part_num | partition_id | ZONE  | svr
_ip        | svr_port | data_size_mb | row_count |
+-------------+---------------+-----------------+--------------+----------+--------------+-------+----
-----------+----------+-------------+-----------+
| obmysql     | tpccdb        | NULL            | bmsql_item   |        1 |            0 | zone3 | 172
.20.249.51 |     2882 |            7 |    100000 |
| obmysql     | tpccdb        | tpcc_group      | bmsql_oorder |        3 |            0 | zone3 | 172
.20.249.51 |     2882 |            1 |    139802 |
| obmysql     | tpccdb        | tpcc_group      | bmsql_oorder |        3 |            1 | zone2 | 172
.20.249.49 |     2882 |            2 |    203152 |
| obmysql     | tpccdb        | tpcc_group      | bmsql_oorder |        3 |            2 | zone1 | 172
.20.249.52 |     2882 |            2 |    150858 |
+-------------+---------------+-----------------+--------------+----------+--------------+-------+----
-----------+----------+-------------+-----------+
4 rows in set (0.053 sec)
```

从结果可以看出，表 `bmsql_item` 是个单分区的普通表，主副本在节点 `172.20.249.51` 上。 表 `bmsql_oorder` 有 3 个分区的表，主副本分别在三个节点上。

查看以下 SQL 的执行计划。

```
MySQL [tpccdb]> explain extended_noaddr select o.o_w_id , o.o_d_id ,o.o_id , i.i_name ,i.i_price ,o.o_
c_id  from bmsql_oorder o , bmsql_item i  where o.o_id = i.i_id  and o.o_w_id  = 3 limit 10 \G
```

输出:

```
*************************** 1. row ***************************
Query Plan: ========================================
|ID|OPERATOR         |NAME|EST. ROWS|COST|
----------------------------------------
|0 |LIMIT            |    |10       |407 |
|1 | NESTED-LOOP JOIN|    |10       |406 |
|2 |  TABLE SCAN     |o   |10       |39  |
|3 |  TABLE GET      |i   |1        |36  |
```

```
========================================

Outputs & filters:
-------------------------------------
  0 - output([o.o_w_id], [o.o_d_id], [o.o_id], [i.i_name], [i.i_price], [o.o_c_id]), filter(nil)
  1 - output([o.o_w_id], [o.o_d_id], [o.o_id], [i.i_name], [i.i_price], [o.o_c_id]), filter(nil)
      conds(nil), nl_params_([o.o_id]), batch_join=true
  2 - output([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), filter(nil),
      access([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), partitions(p0),
      is_index_back=false,
      range_key([o.o_w_id], [o.o_d_id], [o.o_id]), range(3,MIN,MIN ; 3,MAX,MAX),
      range_cond([o.o_w_id = 3])
  3 - output([i.i_name], [i.i_price]), filter(nil),
      access([i.i_name], [i.i_price]), partitions(p0),
      is_index_back=false,
      range_key([i.i_id]), range(MIN ; MAX),
      range_cond([? = i.i_id])

1 row in set (0.008 sec)
```

从上面可以看出 `TABLE SCAN` 访问表 `bmsql_oorder` 的时候，访问的是分区 `partitions(p0)`。0 号分区的主副本在 `zone3` 的节点 `172.20.249.51` 上，跟表 `bmsql_item` 在同一个节点，所以这是个本地 JOIN。

如果调整表 `bmsql_item` 的 PRIMARY_ZONE，换一个节点，再看看这个 SQL。

```
MySQL [tpccdb]> alter table bmsql_item primary_zone='zone1';
Query OK, 0 rows affected (0.043 sec)
```

查看主副本实际位置。

```
+-------------+---------------+-----------------+-------------+----------+--------------+-------+----
-----------+----------+-------------+-----------+
| tenant_name | database_name | tablegroup_name | table_name   | part_num | partition_id | ZONE  | svr
_ip        | svr_port | data_size_mb | row_count |
+-------------+---------------+-----------------+-------------+----------+--------------+-------+----
-----------+----------+-------------+-----------+
| obmysql     | tpccdb        | NULL            | bmsql_item   |        1 |            0 | zone1 | 172
.20.249.52 |     2882 |            7 |    100000 |
| obmysql     | tpccdb        | tpcc_group      | bmsql_oorder |        3 |            0 | zone3 | 172
.20.249.51 |     2882 |            1 |    139802 |
| obmysql     | tpccdb        | tpcc_group      | bmsql_oorder |        3 |            1 | zone2 | 172
.20.249.49 |     2882 |            2 |    203152 |
| obmysql     | tpccdb        | tpcc_group      | bmsql_oorder |        3 |            2 | zone1 | 172
.20.249.52 |     2882 |            2 |    150858 |
+-------------+---------------+-----------------+-------------+----------+--------------+-------+----
-----------+----------+-------------+-----------+
4 rows in set (0.039 sec)
```

进 sys 租户查看 `bmsql_item` 主副本在 `zone1` 的 节点 `172.20.249.52` 上了。 再进业务租户查看 SQL 执行计划。

```
MySQL [tpccdb]> explain extended_noaddr select o.o_w_id , o.o_d_id ,o.o_id , i.i_name ,i.i_price ,o.o_
c_id  from bmsql_oorder o , bmsql_item i  where o.o_id = i.i_id  and o.o_w_id  = 3 limit 10 \G
```

输出:

```
*************************** 1. row ***************************
Query Plan: ============================================
|ID|OPERATOR         |NAME|EST. ROWS|COST|
--------------------------------------------
|0 |LIMIT            |    |10       |407 |
|1 | NESTED-LOOP JOIN|    |10       |406 |
|2 |  TABLE SCAN     |o   |10       |39  |
|3 |  TABLE GET      |i   |1        |36  |
============================================

Outputs & filters:
-------------------------------------
  0 - output([o.o_w_id], [o.o_d_id], [o.o_id], [i.i_name], [i.i_price], [o.o_c_id]), filter(nil)
  1 - output([o.o_w_id], [o.o_d_id], [o.o_id], [i.i_name], [i.i_price], [o.o_c_id]), filter(nil)
      conds(nil), nl_params_([o.o_id]), batch_join=true
  2 - output([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), filter(nil),
      access([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), partitions(p0),
      is_index_back=false,
      range_key([o.o_w_id], [o.o_d_id], [o.o_id]), range(3,MIN,MIN ; 3,MAX,MAX),
      range_cond([o.o_w_id = 3])
  3 - output([i.i_name], [i.i_price]), filter(nil),
      access([i.i_name], [i.i_price]), partitions(p0),
      is_index_back=false,
      range_key([i.i_id]), range(MIN ; MAX),
      range_cond([? = i.i_id])

1 row in set (0.004 sec)
```

跟前面执行计划比，没有发生变化。这个不符合理论。

其原因就是当分区主备副本发生切换后，OBServer 和 OBProxy 内部的分区 LOCATION CACHE 并不是立即更新，而是在 SQL 第一次执行的时候才触发更新。EXPLAIN 查看执行计划的时候，并不会执行 SQL，所以看到的不是实际的执行计划，而是理论的。

我们可以简单访问一下表 bmsql_item 触发 OceanBase 内部 LOCATION CACHE 更新。然后再查看上面 SQL 的执行计划。

```
MySQL [tpccdb]> explain extended_noaddr select o.o_w_id , o.o_d_id ,o.o_id , i.i_name ,i.i_price ,o.o_
c_id  from bmsql_oorder o , bmsql_item i  where o.o_id = i.i_id  and o.o_w_id  = 3 limit 10 \G
```

输出:

```
*************************** 1. row ***************************
Query Plan: ================================================
|ID|OPERATOR             |NAME    |EST. ROWS|COST|
------------------------------------------------
|0 |LIMIT                |        |10       |409 |
|1 | NESTED-LOOP JOIN    |        |10       |408 |
|2 |  TABLE SCAN         |o       |10       |39  |
```

```
|3 |  PX COORDINATOR     |        |1        |37  |
|4 |   EXCHANGE OUT DISTR|:EX10000|1        |36  |
|5 |    TABLE GET        |i       |1        |36  |
=============================================

Outputs & filters:
-------------------------------------
  0 - output([o.o_w_id], [o.o_d_id], [o.o_id], [i.i_name], [i.i_price], [o.o_c_id]), filter(nil)
  1 - output([o.o_w_id], [o.o_d_id], [o.o_id], [i.i_name], [i.i_price], [o.o_c_id]), filter(nil)
      conds(nil), nl_params_([o.o_id]), batch_join=false
  2 - output([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), filter(nil),
      access([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), partitions(p0),
      is_index_back=false,
      range_key([o.o_w_id], [o.o_d_id], [o.o_id]), range(3,MIN,MIN ; 3,MAX,MAX),
      range_cond([o.o_w_id = 3])
  3 - output([i.i_name], [i.i_price]), filter(nil)
  4 - output([i.i_name], [i.i_price]), filter(nil), is_single, dop=1
  5 - output([i.i_name], [i.i_price]), filter(nil),
      access([i.i_name], [i.i_price]), partitions(p0),
      is_index_back=false,
      range_key([i.i_id]), range(MIN ; MAX),
      range_cond([? = i.i_id])

1 row in set (0.004 sec)
```

从上面执行计划可以看出，对表 `bmsql_item` 的访问变成远程访问了 `EXCHANGE OUT DISTR`。并且由于 `bmsql_oorder` 的条件过滤后会有很多记录，所以这里远程访问 `bmsql_item` 的数据还用了并行（`PX COORDINATOR`）。

如上更新的是表 `bmsql_item` 而不是 `bmsql_oorder` 也是有原因的，因为这个连接 SQL 被 OBProxy 路由到哪个 OBServer 节点是由 `from` 语句后的表决定的，即 `bmsql_oorder` 的 `o.o_w_Id=3` 对应的分区的主副本。这个演示是先保证 OBProxy 路由的节点是固定的，方便对比。

下面我们调整这个 SQL 里的表连接顺序再试一次。先回滚 `bmsql_item` 的 PRIMARY_ZONE 变更。

```
MySQL [tpccdb]> explain extended_noaddr select o.o_w_id , o.o_d_id ,o.o_id , i.i_name ,i.i_price ,o.o_
c_id  from bmsql_item i , bmsql_oorder o  where o.o_id = i.i_id  and o.o_w_id  = 3 limit 10\G
```

输出:

```
*************************** 1. row ***************************
Query Plan: ==========================================
|ID|OPERATOR         |NAME|EST. ROWS|COST|
------------------------------------------
|0 |LIMIT            |    |10       |407 |
|1 | NESTED-LOOP JOIN|    |10       |406 |
|2 |  TABLE SCAN     |o   |10       |39  |
|3 |  TABLE GET      |i   |1        |36  |
==========================================

Outputs & filters:
-------------------------------------
  0 - output([o.o_w_id], [o.o_d_id], [o.o_id], [i.i_name], [i.i_price], [o.o_c_id]), filter(nil)
  1 - output([o.o_w_id], [o.o_d_id], [o.o_id], [i.i_name], [i.i_price], [o.o_c_id]), filter(nil)
```

```
      conds(nil), nl_params_([o.o_id]), batch_join=true
  2 - output([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), filter(nil),
      access([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), partitions(p0),
      is_index_back=false,
      range_key([o.o_w_id], [o.o_d_id], [o.o_id]), range(3,MIN,MIN ; 3,MAX,MAX),
      range_cond([o.o_w_id = 3])
  3 - output([i.i_name], [i.i_price]), filter(nil),
      access([i.i_name], [i.i_price]), partitions(p0),
      is_index_back=false,
      range_key([i.i_id]), range(MIN ; MAX),
      range_cond([? = i.i_id])

1 row in set (0.003 sec)
```

调整表 `bmsql_item` 的 PRIMARY_ZONE 到 `zone1` 后，查看上面 SQL 的 EXPLAIN 结果，跟前面一样，没有发生变化。

那么再次使用前面的技巧，对 `bmsql_item` 访问一次。

```
MySQL [tpccdb]> select count(*) from bmsql_item;
+----------+
| count(*) |
+----------+
|   100000 |
+----------+
1 row in set (0.083 sec)

MySQL [tpccdb]> explain extended_noaddr select o.o_w_id , o.o_d_id ,o.o_id , i.i_name ,i.i_pr
*************************** 1. row ***************************
Query Plan: ==================================================
|ID|OPERATOR             |NAME    |EST. ROWS|COST|
--------------------------------------------------
|0 |LIMIT                |        |10       |409 |
|1 | NESTED-LOOP JOIN    |        |10       |408 |
|2 |  TABLE SCAN         |o       |10       |39  |
|3 |  PX COORDINATOR     |        |1        |37  |
|4 |   EXCHANGE OUT DISTR|:EX10000|1        |36  |
|5 |    TABLE GET        |i       |1        |36  |
==================================================

Outputs & filters:
-------------------------------------
  0 - output([o.o_w_id], [o.o_d_id], [o.o_id], [i.i_name], [i.i_price], [o.o_c_id]), filter(nil)
  1 - output([o.o_w_id], [o.o_d_id], [o.o_id], [i.i_name], [i.i_price], [o.o_c_id]), filter(nil)
      conds(nil), nl_params_([o.o_id]), batch_join=false
  2 - output([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), filter(nil),
      access([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), partitions(p0),
      is_index_back=false,
      range_key([o.o_w_id], [o.o_d_id], [o.o_id]), range(3,MIN,MIN ; 3,MAX,MAX),
      range_cond([o.o_w_id = 3])
  3 - output([i.i_name], [i.i_price]), filter(nil)
  4 - output([i.i_name], [i.i_price]), filter(nil), is_single, dop=1
  5 - output([i.i_name], [i.i_price]), filter(nil),
      access([i.i_name], [i.i_price]), partitions(p0),
      is_index_back=false,
      range_key([i.i_id]), range(MIN ; MAX),
```

```
      range_cond([? = i.i_id])

1 row in set (0.003 sec)
```

从上面结果可见，执行计划再次发生变化，不过跟前面那个 SQL 的变化是一样的，对表 `bmsql_item` 的访问依然是远程访问。

看到这里您可能就会奇怪，既然 SQL 被路由到表 `bmsql_item` 的主副本所在的节点了，为什么对表 `bmsql_item` 的访问还是远程访问?

这个问题就出在 EXPLAIN 被路由到的节点上。OBProxy 对 EXPLAIN 语句的路由第一次是随机的，后面每次都跟随前面的 EXPLAIN 语句的路由。虽然上面改变了表 `bmsql_item` 的位置，但是 EXPLAIN 语句依然发往同一个节点（即 `bmsql_oorder` 所在的节点），在那个节点上生产的执行计划就永远是上面这种。

以上 SQL 被路由到哪个节点信息，可以通过 SQL审计视图查询。

```
SELECT /*+ read_consistency(weak) ob_querytimeout(100000000) */ substr(usec_to_time(request_time),1,19
) request_time_, s.svr_ip, s.plan_id, s.client_Ip, s.sid,s.tenant_id, s.tenant_name, s.user_name, s.db
_name, s.query_sql, s.affected_rows, s.return_rows, s.ret_code, s.event, s.elapsed_time, s.queue_time
, s.execute_time, round(s.request_memory_used/1024/1024/1024,2) req_mem_mb, plan_type, is_executor_rpc
, is_inner_sql, TRANSACTION_HASH ,trace_id
FROM gv$sql_audit s
WHERE 1=1  and s.tenant_id = 1002
 and user_name='u_tpcc' and query_sql like '%bmsql_item%'
 and request_time >= time_to_usec(date_sub(CURRENT_TIMESTAMP, interval 30 minute ))
ORDER BY request_time DESC
LIMIT 100;
```

如下可以换一个 OBProxy，或者把之前的 OBProxy KILL 掉重新拉起。再看看上面 SQL 的执行计划，就变成我们期望的那种执行计划了。

```
MySQL [tpccdb]> explain extended_noaddr select o.o_w_id , o.o_d_id ,o.o_id , i.i_name ,i.i_price ,o.o_
c_id  from bmsql_item i , bmsql_oorder o  where o.o_id = i.i_id  and o.o_w_id  = 3 limit 10\G
```

输出:

```
*************************** 1. row ***************************
Query Plan: ====================================================
|ID|OPERATOR             |NAME    |EST. ROWS|COST|
-----------------------------------------------------
|0 |LIMIT                |        |10       |409 |
|1 | NESTED-LOOP JOIN    |        |10       |408 |
|2 |  PX COORDINATOR     |        |10       |40  |
|3 |   EXCHANGE OUT DISTR|:EX10000|10       |39  |
|4 |    TABLE SCAN       |o       |10       |39  |
|5 |  TABLE GET          |i       |1        |36  |
=====================================================

Outputs & filters:
-------------------------------------
  0 - output([o.o_w_id], [o.o_d_id], [o.o_id], [i.i_name], [i.i_price], [o.o_c_id]), filter(nil)
  1 - output([o.o_w_id], [o.o_d_id], [o.o_id], [i.i_name], [i.i_price], [o.o_c_id]), filter(nil)
```

```
      conds(nil), nl_params_([o.o_id]), batch_join=true
  2 - output([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), filter(nil)
  3 - output([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), filter(nil), is_single, dop=1
  4 - output([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), filter(nil),
      access([o.o_w_id], [o.o_id], [o.o_d_id], [o.o_c_id]), partitions(p0),
      is_index_back=false,
      range_key([o.o_w_id], [o.o_d_id], [o.o_id]), range(3,MIN,MIN ; 3,MAX,MAX),
      range_cond([o.o_w_id = 3])
  5 - output([i.i_name], [i.i_price]), filter(nil),
      access([i.i_name], [i.i_price]), partitions(p0),
      is_index_back=false,
      range_key([i.i_id]), range(MIN ; MAX),
      range_cond([? = i.i_id])

1 row in set (0.030 sec)
```

结论:

- EXPLAIN 看执行计划还是很方便的，执行计划的关键是表连接算法和顺序、索引等等。这些都能通过 EXPLAIN 看出问题。
- EXPLAIN 的执行计划可能跟期望的执行计划有些差别，往往是 LOCATION CACHE 和 SQL 路由导致的，请牢记这两个影响因素。

  在生产环境中，表被频繁访问的时候，即使分区主副本切换了，OBServer 也会很快更新路由。虽然是被动更新，代价就是主备副本切换后业务第一次访问的 SQL 会变慢（早期 OceanBase 版本是报错，后期版本改为 OceanBase 内部重试，所以会略微变慢。重试后，OBServer LOCATION CACHE 就更新了）。

如何查看 SQL 实际执行计划

上面内容介绍说 EXPLAIN 执行计划跟期望的执行计划有差别，这个不好说，但是我们可以查看 SQL 实际执行计划来验证是不是有差异。查看 SQL 的实际执行计划要求 SQL 执行过。

运行下面两个 SQL，查看 SQL 审计视图，获取执行节点 和 PLAN_ID 信息。

```
select o.o_w_id , o.o_d_id ,o.o_id , i.i_name ,i.i_price ,o.o_c_id  from bmsql_oorder o , bmsql_

select o.o_w_id , o.o_d_id ,o.o_id , i.i_name ,i.i_price ,o.o_c_id  from bmsql_item i , bmsql_oo

SELECT /*+ read_consistency(weak) ob_querytimeout(100000000) */ substr(usec_to_time(request_time
FROM oceanbase.gv$sql_audit s
WHERE 1=1  and s.tenant_id = 1002
 and user_name='u_tpcc' and query_sql like 'select o.o_w_id%'
 and request_time >= time_to_usec(date_sub(CURRENT_TIMESTAMP, interval 5 minute ))
ORDER BY request_time DESC
LIMIT 10 \G

# 输出:

*************************** 1. row ***************************
request_time_: 2021-10-05 11:24:50
       svr_ip: 172.20.249.52
```

```
    client_Ip: 172.20.249.51
          sid: 3221668666
    tenant_id: 1002
  tenant_name: obmysql
    user_name: u_tpcc
      db_name: tpccdb
    query_sql: select o.o_w_id , o.o_d_id ,o.o_id , i.i_name ,i.i_price ,o.o_c_id  from bmsql_it
      plan_id: 3305
    plan_type: 3
affected_rows: 0
  return_rows: 10
     ret_code: 0
        event: default condition wait
 elapsed_time: 20058
   queue_time: 73
 execute_time: 19726
*************************** 2. row ***************************
request_time_: 2021-10-05 11:24:46
       svr_ip: 172.20.249.51
    client_Ip: 172.20.249.51
          sid: 3222238517
    tenant_id: 1002
  tenant_name: obmysql
    user_name: u_tpcc
      db_name: tpccdb
    query_sql: select o.o_w_id , o.o_d_id ,o.o_id , i.i_name ,i.i_price ,o.o_c_id  from bmsql_oo
      plan_id: 273
    plan_type: 3
affected_rows: 0
  return_rows: 10
     ret_code: 0
        event: system internal wait
 elapsed_time: 141562
   queue_time: 48
 execute_time: 139714
2 rows in set (0.119 sec)
```

其中 `tenant_id`、`svr_ip`、`svr_port` 和 `plan_id` 列信息很重要。查看视图 `gv$plan_cache_plan_explain` 需要这些字段信息。

```
MySQL [tpccdb]> SELECT ip, plan_depth, plan_line_id,operator,name,rows,cost,property from oce
+---------------+------------+--------------+---------------------------+------+------+------+--
| ip            | plan_depth | plan_line_id | operator                  | name | rows | cost | p
+---------------+------------+--------------+---------------------------+------+------+------+--
| 172.20.249.52 |          0 |            0 | PHY_LIMIT                 | NULL |   10 |  408 | N
| 172.20.249.52 |          1 |            1 |  PHY_NESTED_LOOP_JOIN     | NULL |   10 |  407 | N
| 172.20.249.52 |          2 |            2 |   PHY_PX_FIFO_COORD       | NULL |   10 |   39 | N
| 172.20.249.52 |          3 |            3 |    PHY_PX_REDUCE_TRANSMIT | NULL |   10 |   38 | N
| 172.20.249.52 |          4 |            4 |     PHY_TABLE_SCAN        | o    |   10 |   38 | t
| 172.20.249.52 |          2 |            5 |   PHY_TABLE_SCAN          | i    |    1 |   35 | t
+---------------+------------+--------------+---------------------------+------+------+------+--
6 rows in set (0.030 sec)

MySQL [tpccdb]> SELECT ip, plan_depth, plan_line_id,operator,name,rows,cost,property from oce
+---------------+------------+--------------+---------------------------+------+------+------+--
| ip            | plan_depth | plan_line_id | operator                  | name | rows | cost | p
```

```
+---------------+-----------+-------------+--------------------------+------+------+------+--
| 172.20.249.51 |         0 |           0 | PHY_LIMIT                | NULL |   10 |  408 | N
| 172.20.249.51 |         1 |           1 |  PHY_NESTED_LOOP_JOIN    | NULL |   10 |  407 | N
| 172.20.249.51 |         2 |           2 |   PHY_TABLE_SCAN         | o    |   10 |   38 | t
| 172.20.249.51 |         2 |           3 |   PHY_PX_FIFO_COORD      | NULL |    1 |   36 | N
| 172.20.249.51 |         3 |           4 |    PHY_PX_REDUCE_TRANSMIT | NULL |    1 |   35 | N
| 172.20.249.51 |         4 |           5 |     PHY_TABLE_SCAN       | i    |    1 |   35 | t
+---------------+-----------+-------------+--------------------------+------+------+------+--
6 rows in set (0.065 sec)
```

如果是在网页上，且以上输出结果格式化正确，对比 2 个 SQL 的实际执行计划可以看出分别是对那个表进行远程访问。

除了通过 SQL 审计视图定位具体的 SQL 及其执行计划外，还可以通过查看缓存的执行计划汇总视图 `gv$plan_cache_plan_stat`。

```
SELECT s.tenant_id, svr_ip,plan_Id,sql_id,TYPE, query_sql, first_load_time, avg_exe_usec, slow_c
FROM oceanbase.`gv$plan_cache_plan_stat` s
WHERE s.tenant_id = 1002   -- 改成具体的 tenant_id
ORDER BY avg_exe_usec desc limit 10
;

+-----------+---------------+---------+----------------------------------+------+---------------
| tenant_id | svr_ip        | plan_Id | sql_id                           | TYPE | query_sql
+-----------+---------------+---------+----------------------------------+------+---------------
|      1002 | 172.20.249.49 |     251 | FC3FED8CCB2946DE54F1C5BA3656023C |    1 | SELECT d_tax,
|      1002 | 172.20.249.49 |     350 | 54B5A5861DAF78F52D9ADFBEE83D35B5 |    2 | INSERT INTO bm
|      1002 | 172.20.249.49 |     322 | B447DE16B3F42D2409B2A2BE50328E63 |    2 | UPDATE bmsql_w
|      1002 | 172.20.249.52 |    2596 | F4585305C4CB9B091C750826A7DEDD13 |    1 | UPDATE bmsql_d
|      1002 | 172.20.249.52 |    2602 | FC3FED8CCB2946DE54F1C5BA3656023C |    1 | SELECT d_tax,
|      1002 | 172.20.249.51 |     272 | 9D276020142C5B8259B85C3E8966C579 |    3 | select o.o_w_i
from bmsql_oorder o , bmsql_item i
where o.o_id = i.i_id
and o.o_w_id  = 3 limit 10
|      1002 | 172.20.249.49 |     309 | B447DE16B3F42D2409B2A2BE50328E63 |    2 | UPDATE bmsql_w
|      1002 | 172.20.249.51 |     273 | 274FB280E08E876819555632D6951C5A |    3 | select o.o_w_i
|      1002 | 172.20.249.52 |    3367 | CDA629CFD7B13E58328149264FA50055 |    3 | SELECT /*+ rea
|      1002 | 172.20.249.52 |    3362 | 55E820294D4EEFB396B1BAD3CE27CD47 |    3 | SELECT /*+ rea
FROM oceanbase.gv$sql_audit s
WHERE 1=1  and s.tenant_id = 1002
 and user_name='u_tpcc' and query_sql like 'select o.o_w_id%'
 and request_time >= time_to_usec(date_sub(CURRENT_TIMESTAMP, interval 60 minute ))
ORDER BY request_time DESC
LIMIT 100 | 2021-10-05 11:24:04.440565 |       111119 |          1 |          2 |           1500
+-----------+---------------+---------+----------------------------------+------+---------------
10 rows in set (0.014 sec)
```

从这个视图里可以看到全局的 SQL 执行汇总。适合找 TOP N 慢 SQL。根据里面的节点信息、SQLID 和 PLANID 信息，既可以到 SQL 审计视图里定位具体的 SQL 信息，也可以查看实际运行的执行计划信息。

执行计划可以清空，命令如下：

```
ALTER SYSTEM flush plan cache GLOBAL;
```

仅用于测试环境研究，生产环境的 SQL 执行计划缓存通常不可随便清空。清空执行计划会导致所有 SQL 要重新进行一次硬解析。

如何使用 Outline 改变执行计划

OceanBase 在 SQL 执行和性能诊断方面的逻辑参考了 Oracle 的设计思路。OceanBase 也支持 SQL Outline 功能，能够修改在线运行的 SQL 执行计划，同时也支持 SQL 限流功能。

Outline 的用法也是通过 SQL Hint 固定 SQL 的执行计划，可以调整表连接算法、使用的索引等等。

创建 Outline 的语法如下:

```
CREATE [OR REPLACE] OUTLINE outline_name ON stmt [ TO target_stmt ];

# 或

CREATE [OR REPLACE] OUTLINE outline_name ON 'SQLID' using 'HINTS';
```

1. 其中 stmt 为一个带有 HINT 的 DML 语句。限流或固定计划，通过 stmt 中的 HINT 来区分。
2. 如果期望对含有 HINT 的语句进行限流和固定计划，则需要 TO target_stmt 来指明相应的 SQL。

   `create outline outline_name on stmt1 to stmt2;` 对 `stmt2` 创建 outline，让 `stmt2` 使用 `stmt1` 中的 `hint`。
3. 指定 `OR REPLACE` 后，即可对已经存在执行计划或限流规则进行替换（注：限流规则和执行计划间可以彼此替换）。
4. 在使用 `target_stmt` 时，严格要求 `stmt` 与 `target_stmt` 在去掉 `hint` 后完全匹配（实现中为去掉 `hint` 的 `signature` 相同）。若是在创建限流时使用 `target_stmt`，则同时要求 `fix_param` 完全匹配。

通过语句文本或者 SQLID 都可以创建 Outline。不过由于文本中空格和换行如果有差异就会导致匹配不上。所以，不建议通过文本创建 Outline（除非文本非常简单）。这里通过 SQLID 来创建 Outline，只要 SQL 文本不变，SQLID 就不会变。

Outline 相关视图

- 存储 outline 的 schema 信息的系统表

  ```
  oceanbase.__all_outline
  oceanbase.__all_outline_history
  ```

- 固定计划相关虚拟表和视图

  展示的均是当前租户的信息:

  ```
  oceanbase.__tenant_virtual_outline  # 用于 outline 迁移使用的虚拟表，同时显示固定计划的信息。
  ```

```
oceanbase.gv$outline # 在 __tenant_virutal_outline 基础上创建的视图。

information_schema.dba_outlines  # 在 __all_table 上创建的视图。
```

- 限流相关虚拟表和试图

  下表展示的均是当前租户的信息:

```
oceanbase.__tenant_virtual_concurrent_limit_sql  # 展示限流信息
oceanbase.gv$concurrent_limit_sql   # 在 __tenant_virtual_concurrent_limit_sql 上创建的视图。
```

Outline 调优执行计划示例

下面示例还是针对前面测试 SQL，尝试调整一下执行计划里表连接顺序。

- 执行原 SQL，通过视图 `gv$sql_audit` 或 `gv$plan_cache_plan_stat` 找到该 SQL 的 SQLID。
- 根据 SQLID 创建 Outline，指定 HINTS。

```
create outline otl_test_1 on "9D276020142C5B8259B85C3E8966C579" using hint /*+ leading(i) */ ;

MySQL [tpccdb]> select * from oceanbase.__all_outline \G
*************************** 1. row ***************************
   gmt_create: 2021-10-05 12:18:26.032586
 gmt_modified: 2021-10-05 12:18:26.032586
    tenant_id: 0
   outline_id: 1001
  database_id: 1052
schema_version: 1633407506030264
         name: otl_test_1
    signature:
outline_content: /*+leading(i) */
     sql_text:
        owner: root
         used: 0
      version: 3-b20901e8c84d3ea774beeaca963c67d7802e4b4e
   compatible: 1
      enabled: 1
       format: 0
outline_params: ⊠⊠⊠⊠
outline_target:
       sql_id: 9D276020142C5B8259B85C3E8966C579
     owner_id: NULL
1 row in set (0.006 sec)
```

- 再次执行 SQL

```
select o.o_w_id , o.o_d_id ,o.o_id , i.i_name ,i.i_price ,o.o_c_id
from bmsql_oorder o , bmsql_item i
where o.o_id = i.i_id
and o.o_w_id  = 3 limit 10
```

```
;
```

- 查看实际执行计划

```
SELECT s.tenant_id, svr_ip,plan_Id,sql_id,TYPE, query_sql, first_load_time, avg_exe_usec, slow_c
ount,executions, slowest_exe_usec,s.outline_id
FROM oceanbase.`gv$plan_cache_plan_stat` s
WHERE s.tenant_id = 1002   and sql_id='9D276020142C5B8259B85C3E8966C579'
ORDER BY avg_exe_usec desc limit 10
;
```

输出:

```
+-----------+---------------+---------+----------------------------------+------+---------------
| tenant_id | svr_ip        | plan_Id | sql_id                           | TYPE | query_sql
+-----------+---------------+---------+----------------------------------+------+---------------
|      1002 | 172.20.249.51 |     282 | 9D276020142C5B8259B85C3E8966C579 |    3 | select o.o_w_i
from bmsql_oorder o , bmsql_item i
where o.o_id = i.i_id
and o.o_w_id  = 3 limit 10 | 2021-10-05 12:19:15.443750 |       141419 |          1 |          1
|      1002 | 172.20.249.52 |    3488 | 9D276020142C5B8259B85C3E8966C579 |    3 | select o.o_w_i
from bmsql_oorder o , bmsql_item i
where o.o_id = i.i_id
and o.o_w_id  = 3 limit 10 | 2021-10-05 12:18:36.746965 |        88434 |          0 |          1
+-----------+---------------+---------+----------------------------------+------+---------------
2 rows in set (0.092 sec)

SELECT ip, plan_depth, plan_line_id,operator,name,rows,cost,property
from oceanbase.`gv$plan_cache_plan_explain`
where tenant_id=1002 AND ip = '172.20.249.51' AND port=2882 AND plan_id=282  ;

+---------------+------------+--------------+---------------------------+------+-------+--------
| ip            | plan_depth | plan_line_id | operator                  | name | rows  | cost
+---------------+------------+--------------+---------------------------+------+-------+--------
| 172.20.249.51 |          0 |            0 | PHY_LIMIT                 | NULL |    10 | 126991
| 172.20.249.51 |          1 |            1 |  PHY_MERGE_JOIN           | NULL |    10 | 126990
| 172.20.249.51 |          2 |            2 |   PHY_PX_FIFO_COORD       | NULL |  1453 |    856
| 172.20.249.51 |          3 |            3 |    PHY_PX_REDUCE_TRANSMIT | NULL |  1453 |    546
| 172.20.249.51 |          4 |            4 |     PHY_TABLE_SCAN        | i    |  1453 |    546
| 172.20.249.51 |          2 |            5 |   PHY_SORT                | NULL |   689 | 125832
| 172.20.249.51 |          3 |            6 |    PHY_TABLE_SCAN         | o    | 47407 |  11907
+---------------+------------+--------------+---------------------------+------+-------+--------
7 rows in set (0.026 sec)

SELECT ip, plan_depth, plan_line_id,operator,name,rows,cost,property
from oceanbase.`gv$plan_cache_plan_explain`
where tenant_id=1002 AND ip = '172.20.249.52' AND port=2882 AND plan_id=3488  ;

+---------------+------------+--------------+---------------------------+------+-------+--------
| ip            | plan_depth | plan_line_id | operator                  | name | rows  | cost
+---------------+------------+--------------+---------------------------+------+-------+--------
| 172.20.249.52 |          0 |            0 | PHY_LIMIT                 | NULL |    10 | 135657
| 172.20.249.52 |          1 |            1 |  PHY_MERGE_JOIN           | NULL |    10 | 135656
| 172.20.249.52 |          2 |            2 |   PHY_TABLE_SCAN          | i    |  1453 |    546
| 172.20.249.52 |          2 |            3 |   PHY_PX_FIFO_COORD       | NULL | 47407 | 134808
```

```
| 172.20.249.52 |          3 |            4 |    PHY_PX_REDUCE_TRANSMIT | NULL | 47407 | 125832
| 172.20.249.52 |          4 |            5 |     PHY_SORT              | NULL | 47407 | 125832
| 172.20.249.52 |          5 |            6 |      PHY_TABLE_SCAN       | o    | 47407 |  11907
+---------------+------------+--------------+---------------------------+------+-------+--------
7 rows in set (0.008 sec)
```

从上面结果看出，这个 SQL 执行了两次，两次被路由到的节点不同，导致远程访问的表会不一样。但是两个 SQL 都是用同一个 Outline 执行成功的。

实际执行计划里，表 `bmsql_item` 确实是驱动表，表连接算法也从 `NESTED-LOOP JOIN` 变成了 `MERGE JOIN`。 当然，这个只是示例，这个 HINTS 并不一定会让 SQL 性能更好。

- 删除 Outline

```
drop outline otl_test_1 ;
```

删除 Outline 后，该 SQL ID 的执行计划重新生成，又回归到原始的执行计划了。

如何更新统计信息

OceanBase 当前版本（3.1）还没有手动更新表统计信息能力，表的统计信息更新是在合并（`MAJOR FREEZE`）的时候更新的。

默认情况下，OceanBase 收集统计信息会扫描所有数据块，所以统计信息里的分区大小和行数非常准确。如果表非常大，为了节省这个时间，也可以通过参数 `merge_stat_sampling_ratio` 设置一个采样比例（默认是 100）。

在 sys 租户运行:

```
alter system set merge_stat_sampling_ratio = 50 ;
```

查看统计信息

```
SELECT  t.tenant_id, a.tenant_name, t.table_name, d.database_name,  tg.tablegroup_name , t.part_
FROM oceanbase.__all_tenant AS a
 JOIN oceanbase.__all_virtual_database AS d ON ( a.tenant_id = d.tenant_id  )
 JOIN oceanbase.__all_virtual_table AS t  ON (t.tenant_id = d.tenant_id AND t.database_id = d.da
 JOIN oceanbase.__all_virtual_meta_table t2 ON (t.tenant_id = t2.tenant_id AND (t.table_id=t2.ta
 LEFT JOIN oceanbase.__all_virtual_tablegroup AS tg ON (t.tenant_id = tg.tenant_id and  t.tableg
WHERE a.tenant_id IN (1002 ) AND  t.table_type IN (3)
 AND d.database_name = 'tpccdb'
ORDER BY t.tenant_id, tg.tablegroup_name, d.database_name, t.table_name, t2.partition_id
;
```

输出:

```
+-----------+-------------+------------------+---------------+-----------------+----------+-----
| tenant_id | tenant_name | table_name       | database_name | tablegroup_name | part_num | part
+-----------+-------------+------------------+---------------+-----------------+----------+-----
```

```
|      1002 | obmysql     | bmsql_config     | tpccdb        | NULL            |         1 |
|      1002 | obmysql     | bmsql_item       | tpccdb        | NULL            |         1 |
|      1002 | obmysql     | bmsql_customer   | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_customer   | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_customer   | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_district   | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_district   | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_district   | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_history    | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_history    | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_history    | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_new_order  | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_new_order  | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_new_order  | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_oorder     | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_oorder     | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_oorder     | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_order_line | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_order_line | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_order_line | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_stock      | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_stock      | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_stock      | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_warehouse  | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_warehouse  | tpccdb        | tpcc_group      |         3 |
|      1002 | obmysql     | bmsql_warehouse  | tpccdb        | tpcc_group      |         3 |
+-----------+-------------+------------------+---------------+-----------------+-----------+-----
26 rows in set (0.014 sec)
```

抽查一个表数据，可以看出行数统计还是很精准的。

```
MySQL [tpccdb]> select 'p0', count(*) from bmsql_oorder partition (p0)
 union select 'p1', count(*) from bmsql_oorder partition (p1)
 union select 'p2', count(*) from bmsql_oorder partition (p2)
;

+------+----------+
| p0   | count(*) |
+------+----------+
| p0   |   139802 |
| p1   |   203152 |
| p2   |   150876 |
+------+----------+
3 rows in set (0.623 sec)
```

第 8 章：OceanBase 生态工具介绍

本章主要简单介绍 OceanBase 相关的工具、产品等用法。 更多工具欢迎大家补充。

本章目录

8.1 主机监控产品

传统监控产品

OceanBase 本质上是一个单进程软件，进程名是：`observer`。进程启动时，默认会占据主机的 CPU、内存和磁盘空间（指数据盘，启动参数里可以定义）大部分资源。其中 CPU 的占用是声明式的，并不会实际占有。内存的占用是预分配的，默认会占用主机 80% 的可用内存（由参数 `memory_limit_percentage` 指定）。

通常 OceanBase 进程所在主机不适合再运行其他数据库或者应用软件等。磁盘的占用也是预分配的，默认会占用 90%（由参数 `datafile_disk_percentage` 指定）。进程正常启动后会监听两个默认端口，分别是 `2881` 和 `2882`。

跟其他传统数据库一样，可以使用已有的监控平台监控 OceanBase 主机，监控项如下：

- 主机可用性，可以 `ping`。
- 主机 `load`。
- 主机 `cpu`。
- 主机 `mem`。注意仅需监控可用内存。可用内存低于 1G 时，进程 observer 有 OOM 风险。
- 主机磁盘，包括 IO 利用率、IO 延时、IO 吞吐量、分数据盘和日志盘等。
- 主机网络监听端口，包括 `ssh` 的端口（默认 `22`，可修改）、observer 的连接端口（默认 `2881`，可修改）、observer 的 RPC 端口（默认 `2882`，可修改）。
- 主机网卡流量。包括进程 observer 监听的那个网卡。当网卡流量接近能力上限（通常是 10000Mb），SQL 性能变慢，load 变高。

Tsar 工具

Tsar 已在 github 开源，项目地址：https://github.com/alibaba/tsar。

Tsar 是淘宝的一个用来收集服务器系统和应用信息的采集报告工具，可以收集服务器的系统信息（CPU，mem 等），以及应用数据（nginx、swift 等）。收集到的数据会存储在服务器的磁盘上，可以随时查询历史信息，也可以将数据发送到 nagios 报警。

Tsar 带来的性能影响很小，存储空间也很小。可以独立运行，作为现有监控手段的一个补充。

Tsar 增加模块方便，只需要按照 Tsar 的要求编写数据的采集函数和展现函数，就可以把自定义的模块加入到 Tsar 中。

8.2 数据迁移产品

DataX 数据交换平台

DataX 是阿里巴巴开源的一款实用产品，项目地址：https://github.com/alibaba/datax。

DataX 是阿里云 DataWorks 数据集成的开源版本，是在阿里巴巴集团内被广泛使用的离线数据同步工具/平台。

DataX 实现了包括 OceanBase、MySQL、Oracle、SqlServer、Postgre、HDFS、Hive、ADS、HBase、TableStore(OTS)、MaxCompute(ODPS)、Hologres、DRDS 等各种异构数据源之间高效的数据同步功能。

Canal 数据增量工具

Canal 也是阿里巴巴开源的一款数据同步产品，项目地址：https://github.com/alibaba/canal。

Canal 主要用途是基于 MySQL 数据库增量日志解析，提供增量数据订阅和消费。基于日志增量订阅和消费的业务包括:

- 数据库镜像
- 数据库实时备份
- 索引构建和实时维护（拆分异构索引、倒排索引等）
- 业务 cache 刷新
- 带业务逻辑的增量数据处理

当前的 Canal 支持源端 MySQL 版本（包括 5.1.x、5.5.x、5.6.x、5.7.x 和 8.0.x）。

Canal 跟 DataX 结合可以实现将 MySQL 数据实时同步到 OceanBase 的 MySQL 租户中。

8.3 运维工具

DOOBA 性能监控

DOOBA 是 OceanBase 内部的一个运维脚本，使用 Python 语言开发，仅支持 Python 2.7 版本。

项目地址：

- GitHub 项目地址：https://github.com/oceanbase/oceanbase/blob/master/tools/scripts/dooba.py
- Gitee 项目地址：https://gitee.com/oceanbase/oceanbase/blob/master/tools/scripts/dooba.py

使用方法

DOOBA 的原理是使用 MySQL 命令连接到 OceanBase 的 SYS 租户里，实时展示租户 SQL 的 QPS（包括 `select`、`update`、`insert`、`delete`、`commit`）以及相应 SQL 的平均延时（RT）。同时还可以查看各个节点的 SQL 的 QPS 以及 RT 等。

> **说明**
>
> 若要使用 DOOBA，您必须安装 MySQL 客户端。建议安装 5.5/5.6/5.7 版本的 MySQL 客户端，因为早期的 OBProxy （版本低于 2.0.0）可能不完全兼容 MySQL 8.0 的连接协议。

使用命令如下：

```
python dooba.py -h OBPROXY地址 -u root@sys#集群名 -P OBPROXY端口 -p密码

# 如:
python dooba.py -h 172.20.249.54 -u root@sys#obce-3zones -P 2883 -pxxxxxx
```

登录后常用的快捷键如下：

- `c`：选择租户。一般是观察业务租户的性能。
- `1`：查看快捷键。这里没有提到的快捷键尽量别用，不保证功能正确。
- `2`：查看租户性能概览。
- `3`：查看租户各个节点的性能概览。如果节点很多会显示不全（除非屏幕非常大）。
- `tab`：在各个 `widget` 之间跳转。
- `d`：删除 `tab` 焦点所在的 `widget`。
- `R`：恢复当前界面所有的 `widget` 布局（包括被删除的）。

脚本 `dooba` 的源码可以直接编辑查看，工具中各个缩写对应的含义都可在源码中找到解释。

租户性能概览

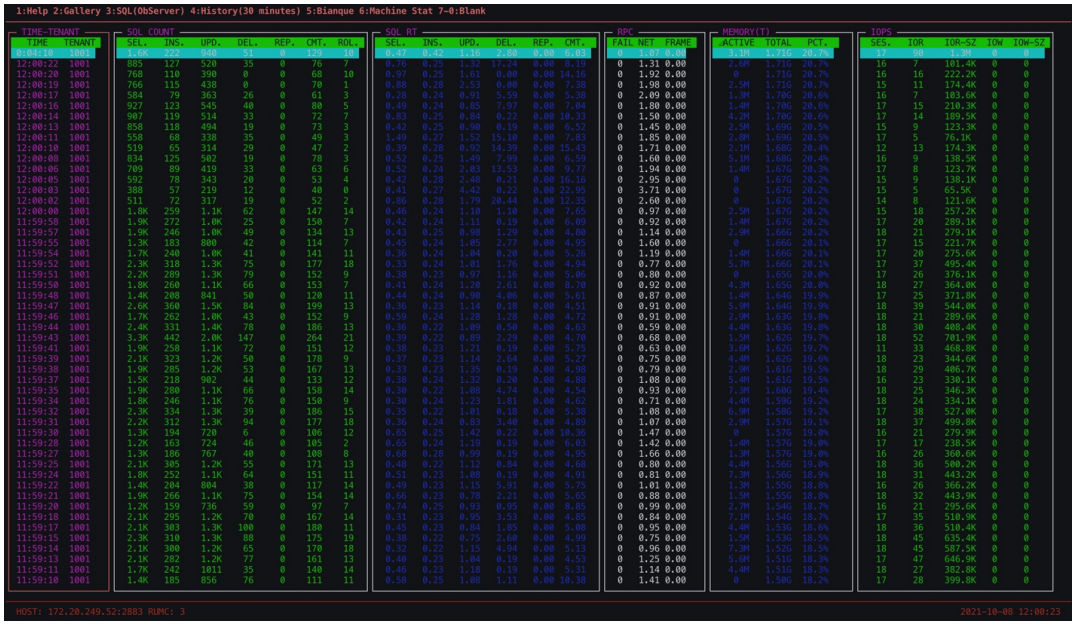
主要查看租户层面 SQL 的 QPS、RT 以及内存、IO 等实时信息。

DOOBA 的性能数据取自 sys 租户的内部视图 `gv$sysstat`。

DOOBA 的监控信息 SQL 都可以直接查看，可以参考它做类似的监控产品。

```
# 取 QPS 和 RT 信息

SELECT t1.con_id tenant_id,t2.zone, t1.svr_ip,t1.svr_port , t1.stat_id, t1.name, t1.value
FROM gv$sysstat t1 JOIN  __all_server t2 ON (t1.svr_ip=t2.svr_ip and t1.svr_port=t2.svr_port) JOIN __a
ll_resource_pool t3 ON (t1.con_id=t3.tenant_id) JOIN __all_unit_config t4 ON (t3.unit_config_id=t4.uni
t_config_id)
WHERE  t1.con_id IN (1001)
and stat_id in (10000,10004,10005,10006,30007,30008,30009,40000,40001,40002,40003,40004,40005,40006,40
007,40008,40009,40010,40011,40012,40013,50000,50001,50002,50003,50004,50005,50006,50007,50008,50009,50
010,50011,60000,60002,60003,60005,130000,130001,130002,130004)
ORDER BY t1.con_id, t2.zone, t1.svr_ip, t1.svr_port;
```

上述 SQL 可以取出所有节点的性能数据，汇总就是整个租户的性能数据。



需要关注以下几点:

- `SQL COUNT`：QPS 信息。包括各类 SQL 的每秒执行次数。

  SQL 类型分别是 `select`、`insert`、`update`、`delete`、`replace`、`commit` 和 `rollback` 。前四类 SQL 总数跟 `commit` 的比值大致可以推出事务的平均语句数。`rollback` 是回滚的信息，如果太高，很有可能是数据库有问题或者业务设计有问题。

- `SQL RT`：SQL 每次平均执行耗时。包括各类 SQL 的每次平均执行耗时，单位是毫秒（ms）。

  SQL 类型分别是 `select`、`insert`、`update`、`delete`、`replace`、`commit`。前四类可以看出 SQL 正常情况下的执行耗时水平。如果有大幅变动，可能是因为数据量突增、SQL 缺乏索引、有锁等待、CPU 瓶颈等引起的。具

体分析还要结合 SQL 审计视图 `gv$sql_audit` 验证。

`commit` 的平均执行耗时也可以反映事务的大小。OceanBase 的 `commit` 会同步事务所有的事务日志（`clog`）到多数派节点（包括自身），这个时间会反映在 `commit` 的耗时里。如果这个值突然增长很大，说明可能有突发的大事务。当节点网络性能下降时或者时间同步误差变大时，都可能导致 `commit` 耗时增大。

- `RPC`：反馈网络信息。
- `Memory`：`Memstore` 内存的实时监控。默认情况下，租户可用内存的一半用于 `Memstore` 的内存，用来存放业务写操作带来的变化的数据。

  OceanBase 的特色之一就是只写数据的变化部分，并且数据会尽可能的维持在内存里不落盘，直到增量内存利用率到达阈值（参数 `freeze_trigger_percentage` ）后触发转储或合并才将数据落盘并释放内存。

  转储是内存中增量数据直接写到磁盘暂存，合并是内存中增量数据跟磁盘中基线数据在内存中合并为最新的数据块（`sstable`）再写回到数据文件中。转储类似传统数据库的 `checkpoint`，对租户性能影响很小并且可以调优，合并对租户性能稍大一些并且也是可以调优的。

  通常建议尽量发起转储，在凌晨低峰期时再发起合并操作。合并可以定时触发，也可以手动触发。当转储次数（参数 `minor_freeze_times` ）用尽时也会自动触发合并。这个监控看的就是增量内存的变化量、累积量和增量内存使用水位。

  这个监控结果再结合视图 `gv$memstore` 基本上可以定位所有的增量内存不足问题。当这个内存水位增长到 `freeze_trigger_percentage` 就会触发转储，当水位继续增长到租户限速阈值（租户参数 `writing_throttling_trigger_percentage` ）会触发租户写入限速，当水位增长到接近 100% 的时候，租户内存耗尽，业务写会大量报错（`Over tenant memory limit1`）。

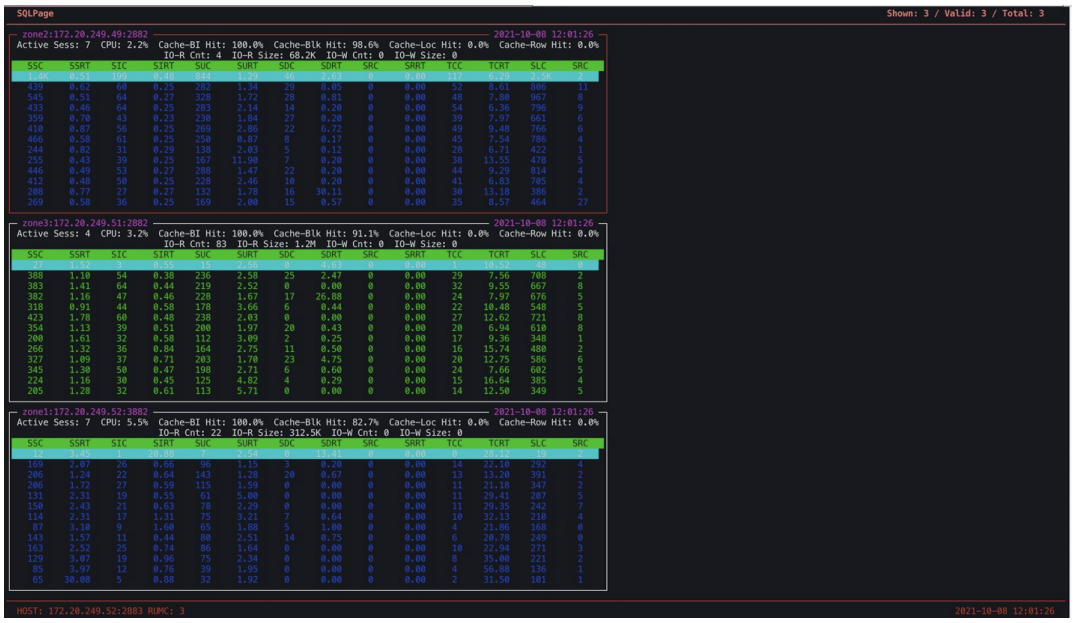
  调优参数 `freeze_trigger_percentage` 和转储内部并发数可以将租户的内存水位控制在一个合理的安全的范围内。

- `IOPS`：磁盘 IO 的实时监控。`SESS` 是会话数（仅供参考，量大的时候可能会不准）。`IOR` 是读 IOPS，`IOR-SZ` 是读 IO 吞吐量。当第一次读入数据或者数据不在内存中时就会有读物理 IO。读物理 IO 如果非常高，会导致 SQL 的平均延时水平很高，说明内存不足，对 IO 依赖性变大。很可能导致 IO 触达瓶颈。`IOW`是写 IOPS，`IOW-SZ` 是写吞吐量。平时很少有写 IO，因为 OceanBase 的写都在内存里。只有转储和合并的时候，能看到短暂密集的写 IO。

租户节点性能概览

查看租户在各个节点 SQL 的 QPS、RT 以及一些缓存命中率等。

```
# 取节点 CPU
SELECT t1.con_id tenant_id,t2.zone, t1.svr_ip,t1.svr_port , round(t1.value/(100*t4.max_cpu), 3) cpu_us
age
FROM gv$sysstat t1 JOIN  __all_server t2 ON (t1.svr_ip=t2.svr_ip and t1.svr_port=t2.svr_port) JOIN __a
ll_resource_pool t3 ON (t1.con_id=t3.tenant_id) JOIN __all_unit_config t4 ON (t3.unit_config_id=t4.uni
t_config_id)
WHERE  t1.con_id IN (1001) and t1.stat_id=140006
ORDER BY t1.con_id, t2.zone, t1.svr_ip, t1.svr_port;
```



需要关注以下几点：

- 活跃会话数：仅供参考，非严格精准。
- CPU 利用率：这个是 OceanBase 内部统计的各个节点的 CPU 利用率，不是 OS 里看到的 CPU 利用率（OceanBase 是单进程软件）。
- 各个缓存的命中率：这里有 `Cache Block` 以及其 `Cache Block Index` 的缓存命中率，主要关注前者。理想情况是 99% 或者 100%。当初次读入大量数据、内存有瓶颈或合并后时，这个利用率会低于 99%，甚至低于 90%。如果长期低于 80%，那说明内存瓶颈问题很严重。其他行缓存命中率通常不是问题。
- 物理 IO 读写：各个节点的物理 IO 信息，有很大参考意义。
- SQL 的 QPS 以及 RT 信息：各个节点的 QPS 可以看出租户请求在各个节点的流量比例关系，相应的 RT 可以看出各个节点的性能状况。其中最后两个 `SLC` 和 `SRC` 表示是节点上的本地 SQL 和远程 SQL 执行次数信息。

  通常本地 SQL 的性能会比远程 SQL 性能好，所以请注意远程 SQL 的比例，如果比例很高，这个节点的 CPU 利用率、SQL 的 RT 均值都会比较高。这个就需要分析原因。可能是租户 PRIMARY_ZONE 不适合打散或者表结构设计不合理，可以考虑使用表分组、调整 PRIMARY_ZONE 或 UNIT_NUM 等等。

ob_admin 工具

`ob_admin` 是 OceanBase 内部开发和运维使用的一个工具，用于处理一些疑难问题。这个工具不是标准产品，有很大的使用风险。仅供对 OceanBase 原理非常熟悉的技术人员使用。

项目地址：

- GitHub 项目地址：https://github.com/oceanbase/oceanbase/tree/master/tools/ob_admin

- Gitee 项目地址：https://gitee.com/oceanbase/oceanbase/tree/master/tools/ob_admin

目前 `ob_admin` 可以用于两个场景：

- 测试数据盘性能，生成 IO 配置文件。
- 多数派故障时将单副本强制激活为主副本。

ob_error 工具

`ob_error` 是随着 OceanBase 一起开源的工具。

项目地址：

- GitHub 项目地址：https://github.com/oceanbase/oceanbase/tree/master/tools/ob_error
- Gitee 项目地址：https://gitee.com/oceanbase/oceanbase/tree/master/tools/ob_error

ob_error 主要用来查看 OceanBase 错误码含义，节省查文档时间。

```
[root@obce00 ~]# ./ob_error 4030

OceanBase:
        OceanBase Error Code: OB_TENANT_OUT_OF_MEM(-4030)
        Message: Over tenant memory limits
        Cause: Internal Error
        Solution: Contact OceanBase Support

[root@obce00 ~]# ./ob_error 4013

OceanBase:
        OceanBase Error Code: OB_ALLOCATE_MEMORY_FAILED(-4013)
        Message: No memory or reach tenant memory limit
        Cause: Internal Error
        Solution: Contact OceanBase Support

Oracle:
        Oracle Error Code: ORA-04013
        Message: number to CACHE must be less than one cycle
        Related OceanBase Error Code:
                OB_ERR_SEQ_CACHE_TOO_LARGE(-4328)
```

Addr2line 工具

Addr2line 工具（标准的 GNU Binutils 中的一部分）是一个可以将指令地址和可执行映像转换成文件名、函数名和源代码行数的工具。一般适用于 debug 版本或带有 symbol 信息的库。

命令使用方法：`addr2line -pCfe $observer $symbol_addr`。其中，`$observer` 是进程 `observer` 的目录，`$symbol_addr` 是进程运行日志中打印的十六进制字符串。

示例如下：

```
[admin@obce01 oceanbase-ce]$ grep ERROR log/observer.log -A1
[2021-10-08 15:35:14.343634] ERROR easy_client_dispatch (easy_client.c:30) [90761][8][YB42AC14F934-000
5CDD26B1C3D3A] [lt=0] [dc=0] easy_io_dispatch is failure: easy not started
 BACKTRACE:0x9124b6e 0x9060f71 0x9057dcf 0x905abd4 0x92ff0d5 0x9395b48 0x9395cf3 0x930a68e 0x4397e89 0
x936beea 0x246f0a5 0x
```

输出结果是函数的调用关系和参数值。将这些反馈给 OceanBase 技术支持人员，可协助解决问题。

```
[admin@obce01 oceanbase-ce]$ addr2line -pCfe bin/observer 0x9124b6e 0x9060f71 0x9057dcf 0x905abd4 0x92
ff0d5 0x9395b48 0x9395cf3 0x930a68e 0x4397e89 0x936beea 0x246f0a5 0x
oceanbase::common::lbt() at ??:?
oceanbase::common::ObLogger::check_error_log(oceanbase::common::ObPLogItem&) at ??:?
oceanbase::common::ObLogger::async_log_message_kv(char const*, int, oceanbase::common::ObLogger::LogLo
cation const&, unsigned long, char const*, long) at ??:?
oceanbase::common::ObLogger::log_message_va(char const*, int, char const*, int, char const*, unsigned
long, char const*, __va_list_tag*) at ??:?
oceanbase::common::easy_log_format_adaptor(int, char const*, int, char const*, unsigned long, char con
st*, ...) at ??:?
easy_client_dispatch at ??:?
easy_client_send at ??:?
oceanbase::rpc::frame::ObReqTransport::send_session(easy_session_t*) const at ??:?
int oceanbase::rpc::frame::ObReqTransport::send<oceanbase::obrpc::ObRpcPacket>(oceanbase::rpc::frame::
ObReqTransport::Request<oceanbase::obrpc::ObRpcPacket> const&, oceanbase::rpc::frame::ObReqTransport::
Result<oceanbase::obrpc::ObRpcPacket>&) const at ??:?
oceanbase::obrpc::ObRpcProxy::send_request(oceanbase::rpc::frame::ObReqTransport::Request<oceanbase::o
brpc::ObRpcPacket> const&, oceanbase::rpc::frame::ObReqTransport::Result<oceanbase::obrpc::ObRpcPacket
>&) const at ??:?
int oceanbase::obrpc::ObRpcProxy::rpc_call<oceanbase::obrpc::ObFetchAliveServerArg, oceanbase::obrpc::
ObFetchAliveServerResult>(oceanbase::obrpc::ObRpcPacketCode, oceanbase::obrpc::ObFetchAliveServerArg c
onst&, oceanbase::obrpc::ObFetchAliveServerResult&, oceanbase::obrpc::Handle*, oceanbase::obrpc::ObRpc
Opts const&) at ??:?
?? ??:0
```

附录：教程文档贡献者

邹阳，从事系统集成工作多年，目前就职于华信永道（北京）科技股份有限公司，主要负责以 Oracle、DB2 为主的数据库，熟悉 AIX 及 Linux 等操作系统，WebSphaer 及 Weblogic 等中间件。已获得 OCOM、MySQL OCP、OGCA 等证书。

刘谦， 目前就职于上海识装信息科技有限公司，主要负责管理 MySQL、Redis、分布式 KV 存储，擅长数据库优化、架构设计等。

严少安，从事数据库管理工作多年，目前就职于某证券公司旗下的金融科技公司，主要负责管理以 MySQL (MariaDB) 为主的数据库。已获得 OBCP、MySQL OCP 等证书。

赵志勇，从事数据库管理工作多年，熟悉 Oracle、MySQL 等数据库，目前就职于某银行，主要负责 Oracle、MySQL 等数据库的日常管理。对 OceanBase 有深入的了解，并已获得 OceanBase 认证证书。

王兆坤，从事数据库管理工作多年，目前就职于四川省农村信用社联合社，负责数据库管理和运维工作。正在深入研究 OceanBase，并已获得 OceanBase 认证证书。

夏克，从事金融行业核心系统设计开发工作多年，目前就职于某交易所子公司，现阶段负责国产数据库调研。已获得 OBCA、PCTA 等证书。

胡军委，从事数据库管理工作多年，目前就职于上海天玑科技股份有限公司杭州分公司，主要负责管理以 Oracle 为主的数据库。已获得 OBCP、OCM、RHCA 等证书。

吉帅，从事数据开发工作多年，目前就职于北京某数据科技公司，主要负责以 MySQL 为主的数据库，擅长大型数据库的设计、性能管理和调优，对 OceanBase 的设计也有深入研究。

罗呈祥，从事 DBA 多年，熟悉 Oracle、MySQL、Postgresql 等相关技术，目前就职于某教育行业 SaaS 服务商，主要负责公司内部关系型数据库的运维管理工作。

许玉冲，从事数据库管理工作多年，目前就职于北京华创方舟科技集团有限公司，主要负责数据库售前、售后、产品选型、Poc 测试、故障处理等工作。已获得 OBCP、OCM、PGCM、HCIE、PCTP、SCDP、MySQL OCP 等证书。

李成许，从事 DBA 工作多年，目前就职于鲁能软件公司研发中心，主要负责 MySQL、Oracle 数据库的运维，擅长 Oracle 和 MySQL 数据库故障诊断处理和性能优化，对 OceanBase 数据库存储等模块也有深入的研究。

葛海波，从事测试工作多年，目前就职于某云平台存储相关部门，主要负责数据库相关功能测试、对外支持及其内部脚本开发，对 Oceanbase 部署及性能测试等有着比较深入的了解。

马顺华，从事运维管理工作多年，目前就职于六棱镜（杭州）科技有限公司，熟悉运维自动化、OceanBase 部署运维、MySQL 运维以及各种云平台技术和产品。并已获得 OceanBase 认证证书。

王硕，从事数据库管理工作多年，目前就职于某知名支付公司，主要负责 MySQL、Redis、MongoDB 的数据库

运维管理工作，对数据库故障诊断、性能调优、备份迁移等方面有丰富的经验。对分布式数据库也有深入研究，并已获得 OceanBase 认证证书。