

# OBCP 认证培训



# 目录

第一章/ OB 分布式架构高级技术

第二章 / OB 存储引擎高级技术

第三章 / OB SQL 引擎高级技术

第四章/ OB SQL调优

第五章 / OB 分布式事务高级技术

第六章/ OBProxy 路由与使用运维

第七章 / OB 备份与恢复

第八章 / OB 运维、 监控与异常处理

# 目录

## 第八章 / OceanBase 运维、监控与异常处理

8.1 用户权限管理

8.2 日志查询

8.2 日常运维操作

8.3 数据库监控

8.4 常见异常处理

8.5 灾难恢复

## 8.1 用户权限管理

## 8.1 黑屏：用户管理

- 数据库用户管理操作包括新建用户、删除用户、修改密码、修改用户名、锁定用户、用户授权和撤销授权等
- Tenant -> user ->
- 用户分为两类：系统租户下的用户，一般租户下的用户。

创建用户时，如果当前会话的租户为系统租户，则新建的用户为系统租户用户，反之一般为一般租户下的用户。

- 创建用户：create user 'my\_user' identified by 'my\_password'
- 删除用户：drop user 'my\_user'
- 给用户相应权限：grant [用户权限] on [资源对象] to username
- 收回权限：revoke [用户权限] on [资源对象] to username
- 查看用户：show grants for username

# 8.1 黑屏： 权限与权限等级管理

## 用户权限

alter
create
Create user
Create view
delete
drop
Grant option
index
insert
Show database
Show view
super
select
update
usage
All Privilege

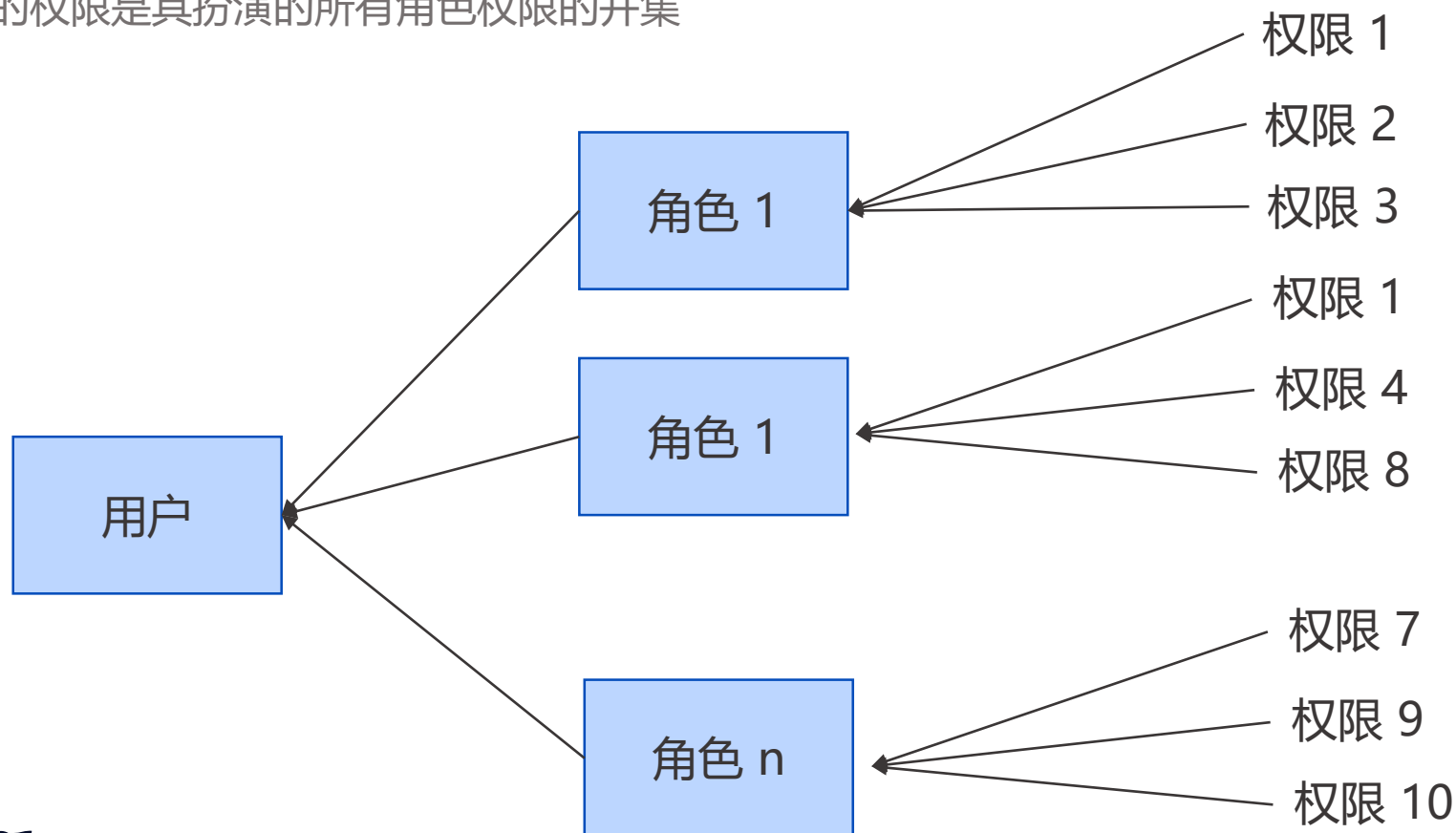
## 权限等级（资源对象）

1. 全局层级：适用于所有的数据库。使用GRANT ALL ON \*.\*授予全局权限。
2. 数据库层级：适用于一个给定数据库中的所有目标。  
使用GRANT ALL ON db\_name.\*授予数据库权限
3. 表层级：表权限适用于一个给定表中的所有列。使用  
GRANT ALL ON db\_name.tbl\_name授予表权限

# 8.1 白屏：OCP 平台的用户管理

OCP 平台的用户，是OCP 平台的使用者、功能系统的操作者（非黑屏用户）。 OCP 用户主要有以下几个关键概念：

- 角色： 角色是一组权限的集合，是权限的载体。
- 用户： 每个用户都必须扮演一个或者多个角色，具备这些角色所包含的权限
- 用户的权限是其扮演的所有角色权限的并集



# 8.1 白屏：OCP 平台增加角色

OCP 管理界面—>安全—>角色管理： 查看系统已有的默认角色，或者创建新的角色并分配权限

集群

租户

主机

系统管理

告警

任务

安全

系统参数

安全

用户管理

角色管理

搜索角色名称

新建角色

角色名称	说明	角色类型	权限	已分配用户	操作
TENANT_VIEWER	租户只读角色，拥有对OCP管理的...	默认角色	CLUSTER:*TENANT:*READ, CLUST...	-	复制
TENANT_MANAGER	租户管理员角色，拥有对OCP管理...	默认角色	CLUSTER:*TENANT:**, CLUSTER:*...	-	复制
BACKUP_MANAGER	集群备份恢复管理角色，拥有对O...	默认角色	CLUSTER:*BACKUP:**, CLUSTER:*...	-	复制
CLUSTER_VIEWER	集群只读角色，拥有对OCP管理的...	默认角色	CLUSTER:*READ, HOST:*READ, TA...	-	复制
CLUSTER_MANAGER	集群管理员角色，拥有对OCP管理...	默认角色	CLUSTER:**, HOST:**, TASK:**, AL...	-	复制
ALARM_MANAGER	OCP告警管理角色，拥有告警和订...	默认角色	ALARM:**, CLUSTER:*READ, HOS...	-	复制
TASK_MANAGER	OCP后台任务管理角色，默认角色...	默认角色	TASK:**	-	复制
HOST_MANAGER	OCP主机管理角色，默认角色，不...	默认角色	HOST:**	-	复制
PROPERTY_MANAGER	OCP系统配置参数管理角色，默认...	默认角色	PROPERTY:**	-	复制
ROLE_MANAGER	OCP角色管理角色，默认角色，不...	默认角色	ROLE:**	-	复制



# 8.1 白屏：OCP 平台增加新用户

OCP 管理界面—>安全—>用户管理：

- 默认 admin 系统管理员用户
- 增加新的用户，并给新用户关联角色

创建好新的用户后，可以使用新的用户登录 OCP

OceanBase

集群

租户

主机

系统管理

告警

任务

安全

系统参数

回到旧版

帮助中心

简体中文

admin

安全

用户管理角色管理

搜索用户名称

创建用户

用户名	角色	邮箱	组织	部门	操作
admin	ADMIN	changeme@changeme.com	-	-	编辑 修改密码 复制 ...

共 1 条

<

1

>

10 条/页

OCEANBASE

## 8.2 日志查询

## 8.2 日志概述

1. Observer程序运行日志：OceanBase在运行过程中会自动生成日志。维护工程师通过查看和分析日志，可以了解OceanBase的启动和运行状态

- /home/admin/oceanbase/log

2. 事务/存储日志：OceanBase在事务执行过程中，会持久化事务/存储日志

- /data/log1/集群名/

# 8.2 事务/存储日志清单

日志名称	日志路径	说明
clog	OBServer服务器的“~/datadir/clog”目录下。	Commit Log，所有Partition共用，日志可能是乱序的，记录事务、PartitionService提供的原始日志内容。此目录下的日志基于Paxos协议在多个副本之间同步。
ilog	OBServer服务器的“~/datadir/ilog”目录下。	Index Log，所有Partition共用，单Partition内部日志有序，记录Partition内部log_id>clog(file_id, offset)的索引信息。每个副本自行记录。
slog	OBServer服务器的“~/datadir/slog”目录下。	记录Storage log,指SSTable操作日志信息。

## 8.2 关于clog

事务的日志包括：redo log, prepare log, commit log, abort log, clear log等

1. redo log记录了事务的具体操作，比如某一行数据的某个字段从A修改为B
2. prepare log记录了事务的prepare状态
3. commit log表示这个事务成功commit，并记录commit信息，比如事务的全局版本号
4. clear log用于通知事务清理事务上下文
5. abort log表示这个事务被回滚

所有的事务日志信息，不用于用户查看和定位系统问题

# 8.2 Observer日志级别 syslog\_level

日志级别	含义
ERROR	严重错误，用于记录系统的故障信息，且必须进行故障排除，否则系统不可用。
WARN	警告，用于记录可能会出现的潜在错误。
INFO(default)	提示，用于记录系统运行的当前状态，该信息为正常信息。
TRACE	更细致化的记录事件消息。
DEBUG	调试信息，用于调试时更详细的了解系统运行状态，包括当前调用的函数名、参数、变量、函数调用返回值等。

## 8.2 Observer日志格式

- 日志记录主要组成：记录时间、日志级别、[模块名]、文件名：行号、线程ID和日志内容

[time] log\_level [module\_name] function\_name (file\_name:file\_no) [thread\_id][Ytrace\_id0-trace\_id1]

[log=last\_log\_print\_time]log\_data

- 例子：

[2015-08-06 15:34:30.006962] INFO [SQL.RESV] ob\_basic\_session\_info.cpp:220 [84753][Y0-0] use database

success.(database\_name=oceanbase)

表示执行SQL指令use database oceanbase成功

## 8.2 Observer日志注意事项

- Observer日志放在 /home/admin/oceanbase/log 目录，包含以下部分：
  - ✓ election.log , election.log.wf: 选举模块日志
  - ✓ rootservice.log , rootservice.log.wf: rootservice模块日志
  - ✓ observer.log , observer.log.wf: 除以上两个模块外其它所有日志
- 日志写满256MB时，会做日志文件切换，原日志文件名加上 .%Y%m%d%H%M%S 格式的时间，如  
observer.log.20161230235020



## 8.2 Observer日志注意事项

1. OceanBase默认不会自动清理日志。另外OceanBase可按日志个数回收日志，通过以下两个

配置项控制：

① `enable_syslog_recycle: true`

② `max_syslog_file_count: 100`

2. 日志限流控制参数 `syslog_io_bandwidth_limit`

3. Warning及以上信息生成单独日志文件控制参数：`enable_syslog_wf`：默认 true

## 8.2 OB MySQL错误码


1. 如果一个错误码的值大于4000，表明它是OB特有的错误码
  - 参考 “OceanBase 1.0 参考指南”
2. 如果在4000以内，表示它是MySQL兼容错误：
  - MySQL Server端错误码范围1000-2000，客户端错误码2000-3000, 1-1000预留
3. 参考：
  - <http://dev.mysql.com/doc/refman/5.1/en/error-messages-server.html>
  - <http://dev.mysql.com/doc/refman/5.1/en/error-messages-client.html>

## 8.2 OB MySQL错误码取值范围说明

错误码范围	说明
[-1 , -4000)	和MySQL兼容的错误码。MySQL Server端错误码范围1000-2000，客户端错误码2000-3000, 1-1000预留。MySQL服务端错误码,请参考 <a href="http://dev.mysql.com/doc/refman/5.1/en/error-messages-server.html">http://dev.mysql.com/doc/refman/5.1/en/error-messages-server.html</a> MySQL客户端错误码，请参考 <a href="http://dev.mysql.com/doc/refman/5.1/en/error-messages-client.html">http://dev.mysql.com/doc/refman/5.1/en/error-messages-client.html</a>
[-4000 , -4500)	通用错误码，含sstable等。
[-4500 , -5000)	RootService错误码。
[-5000 , -6000)	SQL错误码，包含各种schema相关错误。
[-6000, -7000)	事务引擎错误码，包含clog, memtable等。
[-7000 , -7100)	选举模块错误码
[-8000 , -9000)	致命错误，客户端收到8XXX错误，需要关闭SQL连接

## 8.3 日常运维操作

## 8.3 白屏： 集群、Zone、Observer 常用运维操作

 OceanBase

← 集群概览

obcp\_test

ID: 7

● 运行中

总览

拓扑图

租户管理

性能监控

合并管理

参数管理

回到旧版 帮助中心 简体中文 admin

Zone 名	所属 Region	所在机房	机器数量	Root Server	状态	操作
zone3	hangzhou	hangzhou2	2	172.18.6.6:2882	● 运行中	添加 OBCServer 重启 ...
zone2	hangzhou	hangzhou2	2	172.18.6.34:2882(主)	● 运行中	添加 OBCServer 重启 ...
zone1	hangzhou	hangzhou1	2	172.18.6.38:2882	● 运行中	添加 OBCServer 重启 ...

共 3 条 < 1 > 5 条/页

OBCServer 列表

搜索 IP

<input type="checkbox"/>	IP	端口	所在机房	所属 Zone	机型	剩余资源	状态	操作
<input type="checkbox"/>	172.18.6.6	2882	hangzhou2	zone3	ali_server	<div>CPU 20.5/102 核</div> <div>内存 80.0/301.1 GB</div> <div>磁盘 0.00/3.10 TB</div>	● 运行中	重启 停止 ...
<input type="checkbox"/>	172.18.3.14	2882	hangzhou2	zone3	inspur	<div>CPU 3/62 核</div> <div>内存 20.0/201.1 GB</div> <div>磁盘 0.00/1.45 TB</div>	● 运行中	重启 停止 ...

替换 删除

## 8.3 常用运维操作：时钟同步

1. OceanBase从Partition的多个副本中选出主对外提供服务。为避免Paxos的活锁问题，OceanBase采用一种基于时钟的选举算法选主
2. 检查ntp时间是否同步，OceanBase容忍的集群内时钟偏差为100ms
3. `ntpq -p`，输出的offset应小于50ms
4. `clockdiff`

## 8.3 黑屏：集群运维管理

在集群中启动或停止 Zone 的操作通常用于允许或禁止 Zone 内的所有物理服务器对外提供服务的需求场景。

1、查看 Zone 的状态 `Select * from __all_zone ;`

2、启动或停止 Zone

◆ `ALTER SYSTEM {START|STOP|FORCE STOP} ZONE [Zone_Name];`

◆ 示例 1: `ALTER SYSTEM START ZONE Zone1;`

◆ 示例 2: `ALTER SYSTEM STOP ZONE Zone1;`

3、修改 Zone 信息

`ALTER SYSTEM {ALTER|CHANGE|MODIFY} ZONE [Zone_Name] SET [Zone_Option_List];`

`Zone_option_list : region , IDC, Zone_type (READONLY, READWRITE)`

## 8.3 黑屏:Observer 运维管理

### 1、查看 observer 的信息

- ◆ `select * from __all_server ;`
- ◆ `select * from __all_server_event_history;`

### 2、管理 OBServer 状态： 进程启动后，在集群中 OBServer 作为节点单元的管理类似 Zone 的管理。

Start Server 操作：

```
ALTER SYSTEM START SERVER 'ip:port' [, 'ip:port'...] [ZONE='zone']
```

示例：`alter system start server '192.168.100.1:2882'`

Stop Server 操作：

```
ALTER SYSTEM STOP SERVER 'ip:port' [, 'ip:port'...] [ZONE='zone']
```

示例语：`alter system stop server '192.168.100.1:2882' zone='z1'`

stopped 并非等价于进程退出，进程可能仍然在运行，仅仅是集群认为该节点为 stopped 状态。



## 8.3 黑屏:Observer 服务管理（进程）

1、查看 observer 进程： 登录 OceanBase Server 所在的宿主机

```
ps -ef |grep observer
```

2、启动 observer 进程： 登录 OceanBase Server 主机

① os: admin, cd /home/admin/oceanbase/

② ./bin/observer [启动参数] 可以运行

使用 ./bin/observer --help 查看 observer 启动参数的详细信息。

3、停止 observer 进程：

◆ kill -15 `pgrep observer`

◆ kill -9 `pgrep observer`

## 8.3 黑屏：observer服务启动恢复

- 注意：由于增删改数据在内存中，进程启动后
  - 需要与其他副本同步，将clog或者ssd基线数据进行同步（补齐）
  - 需要将上一次合并之后的内存数据恢复出来（clog回放），才能提供服务
- 这一过程可能需要数分钟
- 这一过程结束后，该OBServer才能对外提供服务
- 可以在停止 observer 服务前执行准出（alter system minor freeze;），以加快 observer 服务恢复过程。

## 8.3 黑屏：服务停止（停机运维）

机器需要运维操作时，需要停止OceanBase服务进程

1. 系统租户登陆，确定运维时长，如果大于1小时但小于1天，为了避免服务恢复后的补副本操作，需要设置永久下线时间（`alter system set server_permanent_offline_time = '86400s'`）
2. 将服务从当前 observer 切走，保证停服务的时候，对于业务没有影响（`alter system stop server 'ip地址:2882' ;`）内含切主动作
3. 检查主副本都切走（`select count(*) from __all_virtual_table t, __all_virtual_meta_table m where t.table_id=m.table_id and role=1 and m.svr_ip='ip地址' ;`），返回值应为0
4. 停止进程 `kill -15 <observer pid>`

## 8.3 黑屏：服务恢复（停机运维结束）

机器需要运维操作结束后，需要恢复OceanBase服务进程

1. 机器上电
2. 检查该机器ntp同步状态和服务运行情况
3. admin用户启动observer进程
4. 系统租户登陆，启动server (alter system start server 'ip地址:2882'; )
5. 检查\_all\_server表，查看status为 ' active ' 且 ' start\_service\_time ' 的值>0，则表示observer正常启动并开始提供服务
6. 将永久下线时间改回默认值3600s ( alter system set server\_permanent\_offline\_time = '3600s')

## 8.3 黑屏：故障节点替换

首先要确保集群中有足够的冗余资源（observer），可以代替故障节点进行工作

1. 系统租户登陆，stop server，确保主副本都切走。
2. 为目标zone添加新的server（alter system add server 'ip地址:2882' ZONE 'zone1';）
3. 将故障server下线（alter system delete server 'ip地址:2882' ZONE 'zone1' ;）

OB 会自动将被下线Observer的 Unit 迁移至新添加的 Observer 上。

4. 检查\_\_all\_server表检查server状态，旧 observer 的信息已经消失。

## 8.3 常用运维操作：容量不足

- 内存

1. OB是准内存数据库，任何写操作都需要消耗内存资源，只有合并和转储操作能够释放内存资源，所以当合并和转储速度长时间低于内存消耗速度时，内存最终将被耗尽，服务能力跌零
2. 调大租户内存
3. 转储 / 合并

- 外存

1. 运行日志盘满：可清空较老的日志
2. clog盘满：查询表\_\_all\_virtual\_server\_clog\_stat，清除较老的日志，再合并
3. 数据文件满：扩容，或将较老的数据迁移到历史库，再合并

## 8.4 数据库监控

## 8.4.1 系统监控视图： 系统视图

1. 每当一个observer启动之后，其对应的v\$系列视图便可用于诊断查询
2. gv\$视图的不同之处在于，会从集群所有的observer查询结果并返回
3. 对于等待事件和统计事件相关字段，时间类型单位如无特殊说明是微秒



## 8.4.2 实施监控： zone状态

➤ `select * from __all_zone;`

➤ 关注以下点:

- `is_merge_error` 对应的value是否是0
- `status` 是否全为ACTIVE

## 8.4.2 实时监控：server状态

1. `select zone, svr_ip, status from __all_server;`
2. 查看是否所有server状态都是active
3. inactive状态的server说明对应机器宕机 / 断网，或上面的observer进程退出，此时应登录到该机器检查，并启动上面的observer服务
4. 检查 `start_service_time / last_offline_time`

## 8.4.2 实时监控资源分配率

1. `select zone, svr_ip, cpu_assigned_percent, mem_assigned_percent, disk_assigned_percent from __all_virtual_server_stat;`
2. 如果某个zone中所有server的某项指标(cpu\_assigned\_percent, mem\_assigned\_percent) 都比较高(>90)，后续加租户或扩租户资源可能会因资源不够失败，可考虑集群扩容

## 8.4.2 实时监控：查看机器剩余资源

```
select b.zone, a.svr_ip, a.cpu_total, a.cpu_assigned cpu_ass, a.cpu_assigned_percent  
cpu_ass_percent, round(a.mem_total/1024/1024/1024, 2) as mem_total,  
round(a.mem_assigned/1024/1024/1024, 2) mem_ass, round((a.mem_total-  
a.mem_assigned)/1024/1024/1024, 2) as mem_free, a.mem_assigned_percent mem_ass_percent  
from __all_virtual_server_stat a, __all_server b where a.svr_ip = b.svr_ip order by  
zone, cpu_assigned_percent desc;
```

zone	svr_ip	cpu_total	cpu_ass	cpu_ass_percent	mem_total	mem_ass	mem_free	mem_ass_percent
ZONE_1		30	15.5	51	161.08	43.00	118.08	26
ZONE_2		30	15.5	51	161.08	43.00	118.08	26
ZONE_3		30	15.5	51	161.08	43.00	118.08	26

## 8.4.1 系统监控视图：gv\$memory

展示当前租户在所有ObServer上各个模块的内存使用情况，基于\_\_all\_virtual\_memory\_info创建

1. CONTEXT 对应模块名，即mod\_name
2. COUNT 分配内存的次数，每次为该模块分配内存均加1
3. USED 模块使用的内存大小

```
mysql> select * from gv$memory where USED>0;
```

TENANT_ID	IP	PORT	CONTEXT	COUNT	USED	ALLOC_COUNT	FREE_COUNT
1152	100.81.140.78	2882	OB_ELECTION	5	89160	5	0
1152	100.81.140.78	2882	OB_MEMTABLE_OBJECT	10	99840	170	160
1152	100.81.140.78	2882	OB_LOG_INDEX_MOD	5	40960	5	0
1152	100.81.140.80	2882	OB_ELECTION	5	89160	5	0
1152	100.81.140.80	2882	OB_MEMTABLE_OBJECT	10	99840	170	160
1152	100.81.140.80	2882	OB_LOG_INDEX_MOD	5	40960	5	0
1152	100.81.140.82	2882	OB_SQL_PHY_PLAN	5	327680	762	757
1152	100.81.140.82	2882	OB_SQL_PLAN_CACHE	22	2334	730	708
1152	100.81.140.82	2882	OB_ELECTION	5	89160	5	0
1152	100.81.140.82	2882	OB_MEMTABLE_OBJECT	10	99840	170	160
1152	100.81.140.82	2882	OB_LOG_INDEX_MOD	5	40960	5	0

# 8.4.1 系统监控视图：gv\$memstore

展示当前租户在所有ObServer上 memstore的信息基于\_\_all\_virtual\_tenant\_memstore\_info 创建

- 1. ACTIVE活跃 MemTable 占用的内存
- 2. TOTAL MemStore 整体占用的内(包括 active + frozen memstore内存
- 3. FREEZE\_TRIGGER 触发冻结的内存大小

Oceanbase>desc gv\$memstore;

Field	Type	Null	Key	Default	Extra
TENANT_ID	bigint(20)	NO		NULL	
IP	varchar(32)	NO		NULL	
PORT	bigint(20)	NO		NULL	
ACTIVE	bigint(20)	NO		NULL	
TOTAL	bigint(20)	NO		NULL	
FREEZE_TRIGGER	bigint(20)	NO		NULL	
MEM_LIMIT	bigint(20)	NO		NULL	

mysql> select \* from gv\$memstore;

TENANT_ID	IP	PORT	ACTIVE	TOTAL	FREEZE_TRIGGER	MEM_LIMIT
1152	100.81.140.78	2882	0	40960	5862630340	8375186220
1152	100.81.140.80	2882	0	40960	5862630340	8375186220
1152	100.81.140.82	2882	0	40960	5862630340	8375186220

3 rows in set (0.59 sec)

## 8.4.2 实时监控：磁盘空间

- `select svr_ip, total_size/1024/1024/1024 total_G,  
free_size/1024/1024/1024 free_G, (total_size - free_size)  
/1024/1024/1024 used_G from __all_virtual_disk_stat;`
- ✓ free\_G 一般应 > 800 (根据实际机器配置会有区别)
- ✓ 如果所有server都小于此值, 说明集群存储空间不够, 应考虑集群扩容。
- ✓ 其它情况应检查租户空间

## 8.4.2 实时监控：检查租户分区表情况

- `__all_virtual_meta_table` 记录了副本信息，可按租户，表统计磁盘空间使用
- `select tenant_id, svr_ip, unit_id, table_id, sum(data_size) / 1024 / 1024 / 1024 size_G  
from __all_virtual_meta_table group by 1, 2, 3, 4;`
- 1. 如果租户某unit磁盘空间占用过大(比如>4TB)应考虑增加租户unit
- 2. 如果单表磁盘空间占用过大 (比如>200GB)，应考虑对表进行分区
- 3. 只包含 SSTable磁盘空间，不含memTable内存中数据。



# 8.4.1 系统监控视图： gv\$sql\_audit (sys租户权限)

展示了observer所有执行的sql

Field	Type	Null	Key	Default	Extra
SVR_IP	varchar(32)	NO		NULL	
SVR_PORT	bigint(20)	NO		NULL	
REQUEST_ID	bigint(20)	NO		NULL	
TRACE_ID	varchar(128)	NO		NULL	
CLIENT_IP	varchar(32)	NO		NULL	
CLIENT_PORT	bigint(20)	NO		NULL	
TENANT_ID	bigint(20)	NO		NULL	
TENANT_NAME	varchar(64)	NO		NULL	
USER_ID	bigint(20)	NO		NULL	
USER_NAME	varchar(64)	NO		NULL	
SQL_ID	varchar(32)	NO		NULL	
QUERY_SQL	varchar(32768)	NO		NULL	
AFFECTED_ROWS	bigint(20)	NO		NULL	
RETURN_ROWS	bigint(20)	NO		NULL	
RET_CODE	bigint(20)	NO		NULL	
EVENT	varchar(64)	NO		NULL	
P1TEXT	varchar(64)	NO		NULL	
P1	bigint(20) unsigned	NO		NULL	
P2TEXT	varchar(64)	NO		NULL	
P2	bigint(20) unsigned	NO		NULL	
P3TEXT	varchar(64)	NO		NULL	
P3	bigint(20) unsigned	NO		NULL	
LEVEL	bigint(20)	NO		NULL	
WAIT_CLASS_ID	bigint(20)	NO		NULL	
WAIT_CLASS#	bigint(20)	NO		NULL	
WAIT_CLASS	varchar(64)	NO		NULL	

STATE	varchar(19)	NO		NULL	
WAIT_TIME_MICRO	decimal(38,3)	NO		NULL	
TOTAL_WAIT_TIME_MICRO	decimal(38,3)	NO		NULL	
TOTAL_WAITS	bigint(20)	NO		NULL	
RPC_COUNT	bigint(20)	NO		NULL	
PLAN_TYPE	bigint(20)	NO		NULL	
IS_INNER_SQL	tinyint(4)	NO		NULL	
IS_EXECUTOR_RPC	tinyint(4)	NO		NULL	
IS_HIT_PLAN	tinyint(4)	NO		NULL	
REQUEST_TIME	bigint(20)	NO		NULL	
ELAPSED_TIME	bigint(20)	NO		NULL	
NET_TIME	bigint(20)	NO		NULL	
NET_WAIT_TIME	bigint(20)	NO		NULL	
QUEUE_TIME	bigint(20)	NO		NULL	
DECODE_TIME	bigint(20)	NO		NULL	
GET_PLAN_TIME	bigint(20)	NO		NULL	
EXECUTE_TIME	bigint(20)	NO		NULL	
APPLICATION_WAIT_TIME	bigint(20) unsigned	NO		NULL	
CONCURRENCY_WAIT_TIME	bigint(20) unsigned	NO		NULL	
USER_IO_WAIT_TIME	bigint(20) unsigned	NO		NULL	
SCHEDULE_TIME	bigint(20) unsigned	NO		NULL	
ROW_CACHE_HIT	bigint(20)	NO		NULL	
BLOOM_FILTER_CACHE_HIT	bigint(20)	NO		NULL	
BLOCK_CACHE_HIT	bigint(20)	NO		NULL	
BLOCK_INDEX_CACHE_HIT	bigint(20)	NO		NULL	
DISK_READS	bigint(20)	NO		NULL	
RETRY_CNT	bigint(20)	NO		NULL	
TABLE_SCAN	tinyint(4)	NO		NULL	
CONSISTENCY_LEVEL	bigint(20)	NO		NULL	
MEMSTORE_READ_ROW_COUNT	bigint(20)	NO		NULL	
SSSTORE_READ_ROW_COUNT	bigint(20)	NO		NULL	

## 8.4.1 系统监控视图：gv\$sql\_audit(sys租户权限)

- 检查特定租户下Top 10的sql执行时间

- ```
select sql_id, query_sql, count(*), avg(elapsed_time), avg(execute_time), avg(queue_time), avg(user_io_wait_time)
from gv$sql_audit where tenant_id=1002 group by sql_id having count(*)>1 order by 5 desc limit 10\G;
```

- 检查特定租户下消耗cpu最多的top sql

- ```
select sql_id, avg(execute_time) avg_exec_time, count(*) cnt, avg(execute_time-TOTAL_WAIT_TIME_MICRO)
cpu_time from gv$sql_audit where tenant_id=1002 group by 1 order by avg_exec_time * cnt desc limit 5;
```

## 8.4.1 系统监控视图：gv\$sql

记录所有租户在所有 ObServer 上的  
SQL 相关统计信息，每个 plan 都会在  
表中有一行数据

1. CON\_ID 租户 ID
2. PLAN\_ID 执行计划 ID
3. SQL\_ID SQL 的标识符
4. TYPE 计划的类型，local，remote 或 distribute
5. SQL\_TEXT SQL 语句内容
6. PLAN\_HASH\_VALUE 执行计划的 hash 值
7. FIRST\_LOAD\_TIME 第一次执行的开始时间
8. LAST\_ACTIVE\_TIME 最近一次执行的开始时间
9. AVG\_EXE\_USEC 平均执行耗时
10. SLOWEST\_EXE\_TIME 最慢一次执行的开始时间
11. SLOWEST\_EXE\_USEC 最慢一次执行的耗时
12. SLOW\_COUNT 慢查询次数统计
13. HIT\_COUNT 命中 plan cache 的次数
14. MEM\_USED 内存空间使用大小
15. EXECUTIONS 执行次数
16. DISK\_READS 读盘次数
17. DIRECT\_WRITES 写盘次数
18. BUFFER\_GETS 逻辑读次数
19. APPLICATION\_WAIT\_TIME application 类事件等待时间
20. CONCURRENCY\_WAIT\_TIME concurrency 类事件等待时间
21. USER\_IO\_WAIT\_TIME 所有 IO 类事件等待时间
22. ROWS\_PROCESSED 返回的行数
23. ELAPSED\_TIME 接收至处理完成总消耗时间
24. CPU\_TIME 消耗的 cpu 时间

## 8.4.1 系统监控视图：gv\$plan\_cache\_plan\_stat

1. 普通租户权限登陆
2. 记录所有 ObServer 上当前租户缓存计划的统计信息，可以通过该表找到Top-SQL
3. 每日合并（Merge）会触发计划淘汰，这个表会被更新

```
MySQL [oceanbase]> select * from gv$plan_cache_plan_stat limit 1\G;
***** 1. row *****
      tenant_id: 1
        svr_ip: 10.81.48.147
        svr_port: 2882
        plan_id: 4576
        sql_id: 555FFA5F489B7ACCB66F9C830F03B09
          type: 2
         db_id: 1099511627777
    statement: SELECT data_version,row_count,data_size FROM __all_
    query_sql: SELECT data_version,row_count,data_size FROM __all_
    special_params:
      sys_vars: 45,4194304,2,4,1,0,0,32,2,1,0,1,1,0,10485760,1,1
      plan_hash: 2686512768737522118
    first_load_time: 2017-12-07 02:00:25.552524
    schema_version: 1512532756533952
    merged_version: 8
  last_active_time: 2017-12-07 02:00:32.393120
    avg_exe_usec: 1579
  slowest_exe_time: 1970-01-01 08:00:00.000000
  slowest_exe_usec: 0
        slow_count: 0
        hit_count: 8
        plan_size: 65536
        executions: 9
        disk_reads: 0
        direct_writes: 0
        buffer_gets: 0
  application_wait_time: 0
  concurrency_wait_time: 0
    user_io_wait_time: 0
    rows_processed: 0
    elapsed_time: 14212
    cpu_time: 14212
    large_querys: 0
  delayed_large_querys: 0
    outline_version: 0
    outline_id: -1
    outline_data: /*+ BEGIN_OUTLINE_DATA FULL(@"SEL$1" "oceanbase.____a
    table_scan: 1
1 row in set (0.13 sec)
```

## 8.4.1 系统监控视图：gv\$plan\_cache\_plan\_explain

- 记录当前租户在所有ObServer上缓存的计划的 explain 信息（具体的计划信息）
- 需要提供四元组(tenant\_id, ip, port, plan\_id)

```
mysql> select * from oceanbase.gv$plan_cache_plan_explain where tenant_id = 1061 and ip = "11.232.2.27" and port = 2882 and plan_id = 121005;
```

TENANT_ID	IP	PORT	PLAN_ID	OPERATOR	NAME	ROWS	COST	PROPERTY
1061	11.232.2.27	2882	121005	PHY_ROOT_TRANSMIT	NULL	0	0	NULL
1061	11.232.2.27	2882	121005	PHY_TABLE_SCAN	business_activity_10(business_activity_gmt_ind)	5	83	table_rows:5, access_rows:5, est_method:storage, opt_method:cost_based, avl_name:business_activity_gmt_ind, pruned_name:business_activity_10,idx_oracle_flag

```
2 rows in set (0.01 sec)
```

## 8.4.2 实时监控：集群级事件 - 查看root service操作记录

- `__all_rootservice_event_history`: 记录集群级事件，如major freeze, 合并, server 上下线, 修改primary\_zone引发的切主操作、负载均衡任务执行等, 保留最近7天历史
  - ✓ `select * from __all_rootservice_event_history order by gmt_create desc limit 10;`
- 如发现一台server宕机, 要确认OceanBase探测到此server宕机时间点, 可执以下SQL:
  - ✓ `select * from __all_rootservice_event_history where module = 'server';`

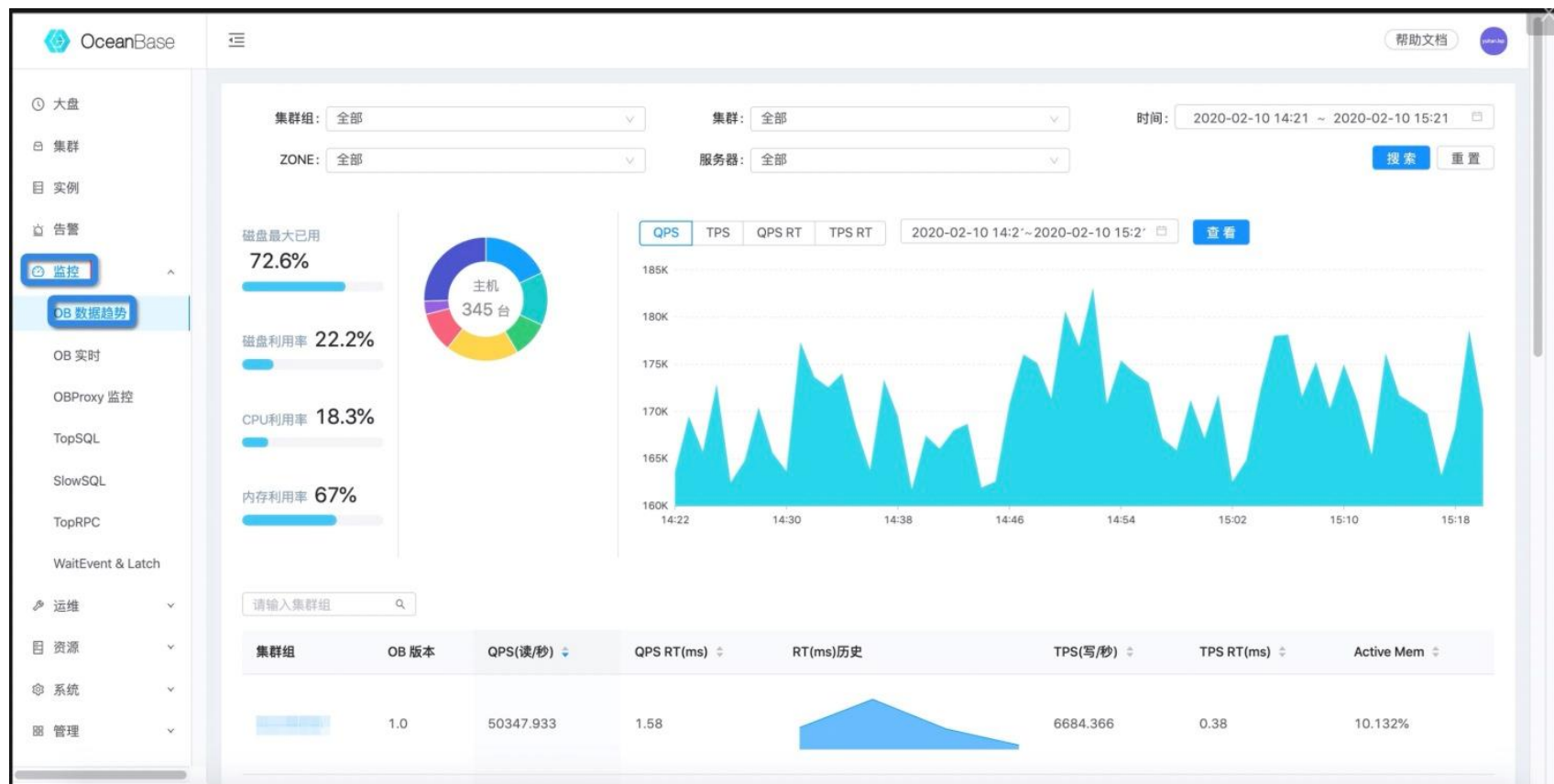
## 8.4.2 实时监控：server级事件

- `__all_server_event_history`:
  - 记录server级事件，如转储，用户发起的系统命令
  - 保留最近7天历史
- 查看转储次数：

```
select * from __all_server_event_history where module like '%minor%'  
order by gmt_create desc limit 10;
```

## 8.4.3 性能监控：常规监测

性能问题应优先通过OCP 管理员入口 ==> 集群入口 ==> 性能 监控 ==> 数据趋势中查看 QPS\_RT, TPS\_RT, 大致定位出问题时间点





## 8.4.3 性能监控：捞取慢SQL

定位方法 1：

- OceanBase中执行时间超过 `trace_log_slow_query_watermark` (系统参数)的sql，在 observer 日志中都会打slow query消息。默认 100ms
- 在 observer 日志中查找慢 SQL 消息：
  - ① `cd /home/admin/oceanbase/log`
  - ② `fgrep '[slow query]' observer.log |sed -e 's/|/\n/g' | more`  
<--查看日志中所有的 slow query
  - ① `grep "<trace_id>" observer.log |sed -e 's/|/\n/g' | more`  
<---根据trace\_id 查询某个 slow query

## 8.4.3 性能监控：捞取慢SQL

关注点：

1. [optimize\_end] u=439 方框号内的是指SQL执行经过的每一个内部模块
2. trace\_id与gv\$sql\_audit里的trace\_id字段对应
3. stmt是指执行的SQL
4. u代表每一步消耗的时间，单位是微秒
5. total\_timeu是指整个过程消耗的总时间，这里是31ms。
6. [例子](#)

```
[2020-09-05 08:33:36.954843] TRACE [TRACE]:0 [120535][1114] [YB420BA64E88-0005ACAF1E96D32] [lt=16] [dc=0] [slow query]((TRACE=begin_ts=1599266016923637
2020-09-05 00:33:36.923637↓
[process_begin] u=0 in_queue_time:21, receive_ts:1599266016923615,
enqueue_ts:1599266016923619↓
[start_sql] u=1 addr:{ip:"11.166.78.136", port:27521}↓
[query_begin] u=1 trace_id:YB420BA64E88-0005ACAF1E96D32↓
[before_processor_run] u=3 ↓
[session] u=4 sid:3221663963, tenant_id:1↓
[session] u=9 sid:3221663963, tenant_id:1↓
[parse_begin] u=25 stmt:"select a.tenant_id, count(*) from all_virtual_table as
a left join all_virtual_ori_schema_version as b on a.tenant_id = b.tenant_id and
a.table_id = b.table_id where a.tenant_id != 1 and b.table_id is null group by
a.tenant_id", stmt_len:230↓
[cache_get_plan_begin] u=8 ↓
[cache_get_plan_end] u=33 ↓
[llc_bitmap, llc_bitmap_size FROM __all_column_statistic WHERE TENANT_ID=1 AND ↓
[end_stmt] u=3 ↓
[process_end] u=21130 (total_timeu=31205 DROPPED_EVENTS=488) ↓
↓
↓
```

## 8.4.3 性能监控：捞取慢SQL

定位方法 2:

1. OceanBase提供两张虚拟表 v\$sql\_audit , gv\$sql\_audit 记录最近一段时间sql执行历史
2. v\$sql\_audit 存储本机的sql执行历史, gv\$sql\_audit 存储整个集群的sql执行历史
3. 查询 v\$sql\_audit 表, 如查询某租户执行时间大于1s (1000000微秒)的SQL:

```
select * from v$sql_audit where tenant_id = <tenant id> and elapsed_time >
1000000 limit 10
```

4. 查询SQL执行时间按秒分布的直方图

```
select round(elapsed_time/1000000), count(*) from v$sql_audit where tenant_id =
<tenant_id> group by 1;
```

## 8.4.3 性能监控：捞取慢SQL

定位方法 3：

1. proxy慢查询？ - 分析proxy日志
2. server show trace？ - 分析server日志
3. 远程执行？为什么？
4. 机器负载如何？

- 配置项默认为5s  
slow\_transaction\_time\_threshold=5s
- 修改配置项  
e.g. 将慢查询阈值设置为 100ms  
alter proxyconfig set  
slow\_transaction\_time\_threshold= '100ms';

# Y. 5 慢查询举例

[2016-05-24 22:39:00.824392] WARN [PROXY.SM] update\_cmd\_stats (ob\_mysql\_sm.cpp:4357) [28044][Y0-7F70BE3853F0] [14]  
Slow query:

```
client_ip=127.0.0.1:17403 // 执行SQL client IP
server_ip=100.81.152.109:45785 // SQL被路由到的observer
conn_id=2147549270
cs_id=1
sm_id=8
cmd_size_stats=
  client_request_bytes:26 // 客户端请求 SQL大小
  server_request_bytes:26 // 路由到observer SQL大小
  server_response_bytes:1998889110 // observer转发给obproxy数据大小
  client_response_bytes:1998889110 // obproxy转发给client数据大小
cmd_time_stats=
  client_transaction_idle_time_us=0 // 在事务中该条SQL与上一条SQL执行结束之间的间隔时间, 即客户端事务中SQL间隔时间
  client_request_read_time_us=0 // obproxy读取客户端SQL耗时
  server_request_write_time_us=0 // obproxy将客户端SQL转发给observer耗时
  client_request_analyze_time_us=15 // obproxy 解析本条SQL消耗时间
  cluster_resource_create_time_us=0 // 如果执行该条SQL, 需要收集OB cluster相关信息(一般发生在第一次连接到该集群的时候), 建立集群相关信息耗时
  pl_lookup_time_us=3158 // 查询partition location耗时
  prepare_send_request_to_server_time_us=3196 // 从obproxy接受到客户端请求, 到转发到observer执行前总计时间, 正常应该是前面所有时间之和
  server_process_request_time_us=955 // observer处理请求的时间, 对于流式请求就是从observer处理数据, 到第一次转发数据的时间
  server_response_read_time_us=26067801 // observer转发数据到obproxy耗时, 对于流式请求observer是一边处理请求, 一边转发数据(包含网络上的等待时间)
  client_response_write_time_us=716825 // obproxy将数据写到客户端耗时
request_total_time_us=26788969 // 处理该请求总时间
sql=select * from sbtest1 // 请求SQL
```

## 8.4.3 修复慢SQL

1. 创建索引：当慢SQL因无合适索引可用时导致时，可创建索引
2. outline绑定：
  1. 如慢SQL由OceanBase优化器选择了不够优的执行计划导致，可通过outline绑定执行计划(具体hint语法和outline绑定语法请参考OceanBase官网文档)。
3. 实例：将 `select * from t1 where v1 = 3` 这条SELECT绑定走主键索引
  1. 以业务租户身份登录到OceanBase （注意 mysql 命令中要加上 -c 参数)后执行以下命令：
  2. `create outline bind_to_primary_key on select/*+ index(t1 primary)*/ * f rom t1 where v1 = 3;`

## 8.5 常见异常处理

## 8.5.1 集群问题排查解决：集群故障

- 通过OCP或者内部表（\_\_all\_server）查看是否有节点处于inactive状态
- 应急操作：
  - 重启OBServer进程
  - 如果是硬件因素导致，先修复硬件故障（包括网络）



## 8.5.1 集群问题排查解决：机器硬件故障

- 建议 “alter system stop zone” 将observer 上的 leader 服务切走，减少对业务的影响
- 修改primary zone，如果stop zone不能解决，可以尝试用这个办法
- 如果故障集群正在合并，可以快速暂停掉，确定影响后再决策是否打开

## 8.5.1 集群问题排查解决：合并较慢

合并和转储较慢：

- 有些压缩算法会比其它压缩算法慢，如zstd要比lz4慢。
- 轮转合并被打开
- 合并线程数太少

## 8.5.1 集群问题排查解决：合并异常

- ① 确认是否有merge\_error, 卡合并的  
partition
- ② 确认卡合并的server
- ③ 检查IO是否到瓶颈
- ④ 检查是否有硬件问题
- ⑤ 检查是否磁盘已满(clog/程序运行日志)
- ⑥ 检查内核是否有问题
- ⑦ 检查observer的日志
- ⑧ 检查参数配置异常（手工档）



## 8.5.1 集群问题排查解决：内部表 / 系统参数不可用

- 检查NTP服务
- 检查时钟是否同步 (比如 `ntpq -p`, `clockdiff`)
- 时钟同步的安全范围在100ms以内，超过100ms则会给集群带来各种问题：  
选主失败、partition无主等

## 8.5.1 Unit/副本迁移失败

1. 检查动态负载均衡是否开启: `show parameters like 'enable_rebalance';`
  - 如果需要副本自动迁移（比如delete server），该配置项必须为true。
2. 检查resource\_soft\_limit参数的值是否小于100:
  - `show parameters like 'resource_soft_limit';`
  - 如果值 $\geq 100$ ，默认不会往新添加的机器上做Unit迁移和均衡。

## 8.5.1 集群问题排查解决：负载均衡 - 迁移复制

内部表:

- `__all_virtual_rebalance_task_stat`: 迁移复制任务
- `__all_virtual_partition_migration_status`: observer 端正在执行的对象迁移任务

配置项:

`enable_rereplication` 复制开关

`enable_rebalance` 迁移开关

`resource_soft_limit` Unit自动均衡至空闲资源的开关

`data_copy_concurrency` 任务并发

`server_data_copy_in_concurrency` 单机拷入并发

`server_data_copy_out_concurrency` 单机拷出并发

`sys_bkgd_net_percentage` 网络带宽限制

`sys_bkgd_io_low_percentage` io限制

`sys_bkgd_io_high_percentage` io限制

## 8.5.2 业务问题排查解决：远程/异地执行

- gv\$sql\_audit中的plan\_type字段：1 – 本地执行；2 – 远端执行；3 – 分布式执行
- 原因：
  1. obproxy是轻量级的sql parser；过于复杂的WHERE条件，可能会导致proxy路由不准。
  2. JDBC在执行sql前可能会发一些 set auto\_commit/select @tx\_read\_only 之类的请求，obproxy会随机选择事务的协调者，可能造成异地执行。
- 在OCP 数据趋势中选中server后看SQL\_COUNT中SQL\_REMOTE\_COUNT指标

## 8.6 灾难恢复



## 8.6 灾难恢复

灾难恢复是指当数据库中的数据在被有意或无意破坏后复原数据库所需要执行的活动。

- **回收站：** 用来存储被删掉的对象。被放入回收站的对象数据都被保存，仍然占据着物理空间，除非手动进行清除（PURGE RECYCLEBIN）。
- **闪回查询：** 允许用户可以获取某个历史版本的数据。

# 8.6 回收站

➤ 回收站支持的对象

如下表所示可以进入回收站的对象有索引、表和库。被删除的租户是无法进入回收站的.

模式	索引 (Index)	表 (Table)	数据库 (Database)	租户 (Tenant)
MySQL	✓	✓	✓	✗
Oracle	✗	✓	✗	✗

➤ 查看回收站

```
show recyclebin;
```

➤ 开关回收站

租户创建之后，默认是开启回收站的，此时对数据库对象进行 Truncate / Drop 操作后，对象会进入到回收站

- 租户级别的开启关闭语句：set global `recyclebin` = on /off;
- Session 级别的开启关闭语句：set @@recyclebin = on/off

## 8.6 回收站

### ➤ 恢复回收站数据

使用 FLASHBACK 命令可恢复回收站中的数据库和表对象，只有租户的管理员用户才可以使用该命令：

- 恢复对象数据库

```
FLASHBACK DATABASE <object_name> TO BEFORE DROP [RENAME TO database_name];
```

- 恢复对象表

```
FLASHBACK TABLE <object_name> TO BEFORE DROP [RENAME to table_name];
```

### ➤ 回收站清理

回收站中的数据可以通过 PURGE 命令清理；当一个对象的上层对象被 PURGE，那么当前回收站中关联的下一层对象也会被 PURGE

- 指定库物理删除： PURGE DATABASE <object\_name>;
- 指定表物理删除： PURGE TABLE <object\_name>;
- 指定索引表物理删除： PURGE INDEX <object\_name>;
- 清空整个回收站： PURGE RECYCLEBIN;

## 8.6 闪回查询

闪回查询（Flashback Query）是 Oracle 中记录级别的闪回功能。该功能允许用户获取某个历史版本的数据。OB同时支持 MySQL 和 Oracle 两种模式下的查询，具体细节可以参考官网文档。

闪回查询支持 SCN（time\_to\_usec()）和 TIMESTAMP 两种维度的查询（usec\_to\_time()）。

示例 1：通过 TIMESTAMP 指定的历史时间并闪回查询一张单表在该历史时间中的状态的数据：

```
select * from tbl1 as of timestamp TO_TIMESTAMP('2020-08-13 16:20:00','yyyy-mm-dd hh24:mi:ss');
```

示例 2：通过 TIMESTAMP 指定的历史时间并闪回查询多表在该历史时间中的状态的数据：

```
select * from tbl1 as of timestamp TO_TIMESTAMP('2020-08-13 16:20:00','yyyy-mm-dd hh24:mi:ss'),tbl2 as  
of timestamp TO_TIMESTAMP('2020-08-13 16:20:00','yyyy-mm-dd hh24:mi:ss');
```

示例 3：通过 SCN 指定历史时间并闪回查询单表在该历史时间点的状态的数据：

```
select * from tbl1 as of scn 1582807800000000;
```

## 8.6 restore point

### 创建 restore point（租户级）

```
obclient> CREATE RESTORE POINT restore_point;
```

### 查询 restore point

```
obclient> SELECT * FROM V$RESTORE_POINT;
```

#### •查询历史数据:

##### •MySQL 模式

```
obclient> SELECT * FROM table_name AS OF SNAPSHOT 10000;
```

##### •Oracle 模式

```
obclient> SELECT * FROM table_name AS OF SCN 10000;
```

### 删除 restore point

保留的 restore point 对应的数据会占用相应的存储资源，在分析业务结束后需要手动执行删除 restore point 的操作。

```
obclient> DROP RESTORE POINT restore_point;
```

#### 当前 restore point 功能的使用限制如下:

- 不支持物理备份。
- 不支持主备库。
- 创建 restore point 后，如果对创建 restore point 前就存在的表执行 DDL 语句，系统会报 -4179 的错误。
- 由于 restore point 功能依赖 GTS 维护全局的一致性快照，故在使用 restore point 时，需要开启 GTS。
  - 开启 GTS 的 SQL 命令如下。
  - obclient> set GLOBAL ob\_timestamp\_service='GTS';

感谢学习