

OCEANBASE

# 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 路由策略与使用运维](#)

第七章 / OB 备份与恢复

第八章 / OceanBase 监控与故障排查

OCEANBASE

# 目录

## 第六章 / DBProxy 路由策略与使用运维

- 6.1 OBProxy 简介
- 6.2 OBProxy 部署
- 6.3 路由实现
- 6.4 使用和运维
- 6.5 常见问题

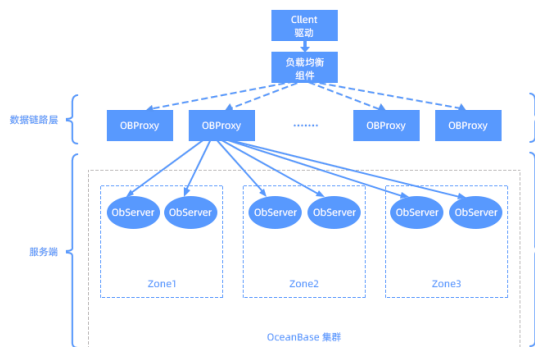
OCEANBASE

## 6.1 OBProxy 简介

OCEANBASE

## 6.1 背景

- OceanBase在部署完成后，应用可以采用OB提供的客户端、MySQL客户端或者其他语言的客户端来访问OceanBase，OceanBase以服务的形式提供给应用访问。OceanBase 集群一般包含多个 Zone，每个 Zone 中又包含一台或多台 OBDServer，集群中的每个 OBDServer 都可以接收用户的连接并处理请求。
- 实现一个OBProxy来接受来自应用请求，并转发给OBDServer，然后OBDServer将数据返回给OBProxy，OBProxy将数据转发给应用客户端



OCEANBASE

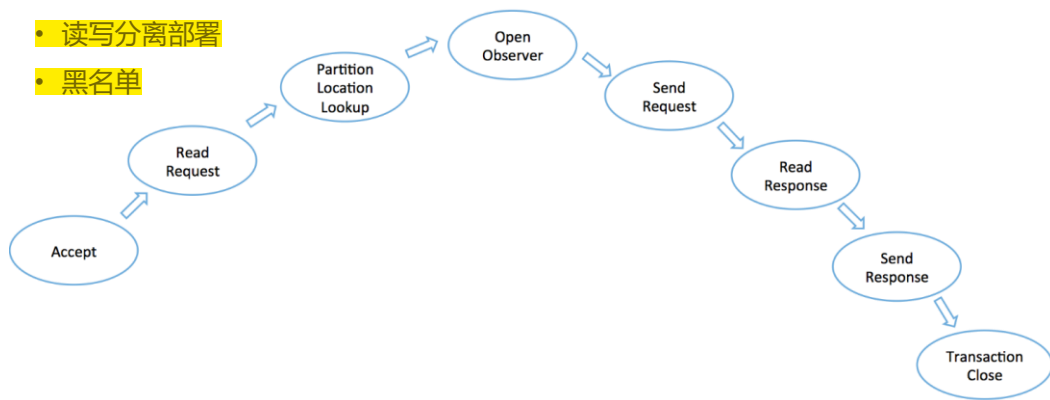
OBProxy作为OceanBase的高性能且易于运维的反向代理服务器，具有防连

接闪断、OBDServer宕机或升级不影响客户端正常请求、兼容所有

MySQL客户端、支持热升级和多集群功能

## 6.1 OBProxy核心功能（路由）

- 简单SqlParser
- LDC路由
- 读写分离部署
- 黑名单



OCEANBASE

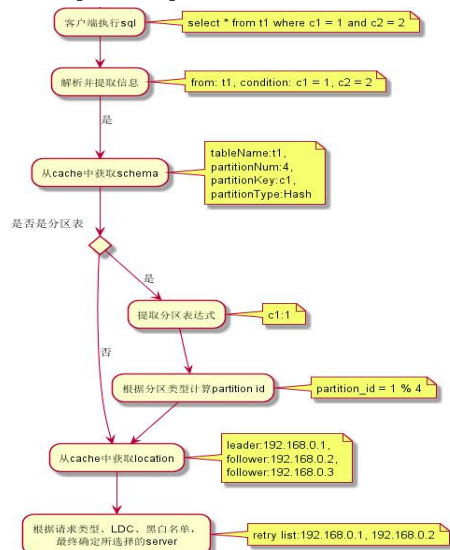
SQLParser: 轻量的sql解析, 判断出客户端的sql请求所涉及的表的主副本在哪台机器上, 将请求路由至主副本所在的机器上

LDC路由: 主要对于读写分离的场景, 根据observer和obproxy配置的region (区域) 和LDC (逻辑机房), 将请求发送给本地的副本 (后面proxy路由策略会详细说明)

读写分离部署: 对于读写分离的场景, OBProxy会把请求优先发送到本地的只读副本

黑名单: OBProxy在路由过程中, 如果发现OBServer不可用, 则把该server加入到黑名单

## 6.1 OBProxy核心功能（路由）



总体流程如上图所示，大致分为以下几个步骤：

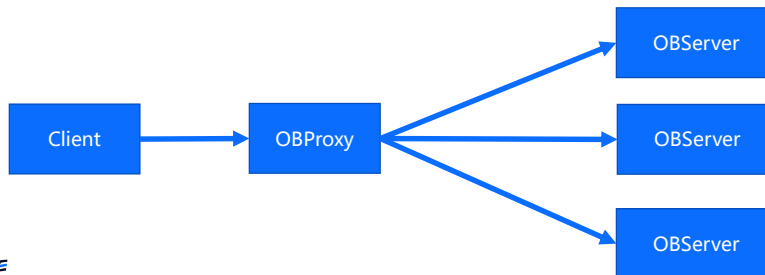
1. **sql parser**：根据用户sql解析出抽象语法树
2. **获取table schema**：根据抽象语法树提取出的表名从table cache里面找到对应的schema
3. **提取分区表达式**：根据table schema和抽象语法树提取分区表达式

4. 计算partition id: 计算分区表达式和分区类型（hash、range或者其他分区类型），计算partition id
5. 计算location: 根据partition id从partition cache里面找到对应的location, 并且根据请求类型（读主或是读备）对location进行优先级排序



## 6.1 OBProxy核心功能（连接管理）

- 在observer宕机/升级/重启时，客户端与OBProxy的连接不会断开，OBProxy可以迅速切换到正常的server上，对应用透明
- OBProxy支持用户通过同一个OBProxy访问多个OceanBase集群
- Server session对于每个client session独占
- 同一个client session对应server session状态保持相同(session变量同步)

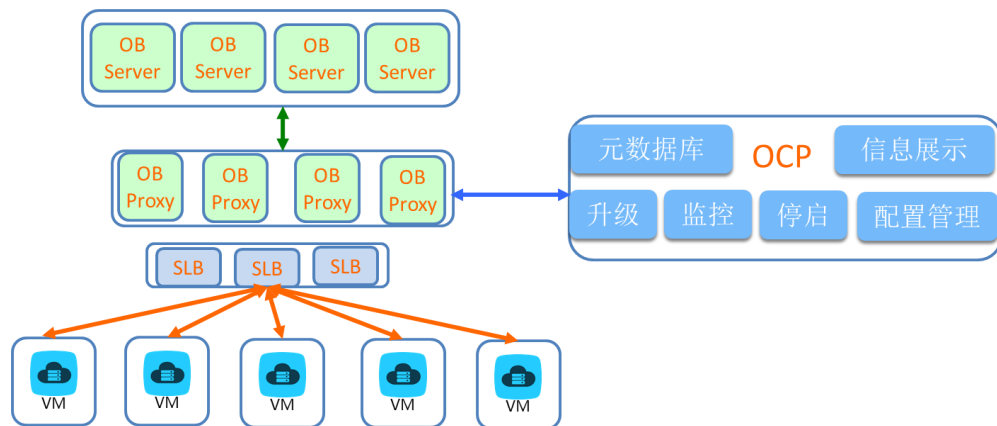


OBProxy与OB集群（OBServer）保持长连接，客户端一般通过连接池的方式连接到OBProxy

## 6.2 OBProxy 部署

OCEANBASE

## 6.2 部署方式 – 集中部署

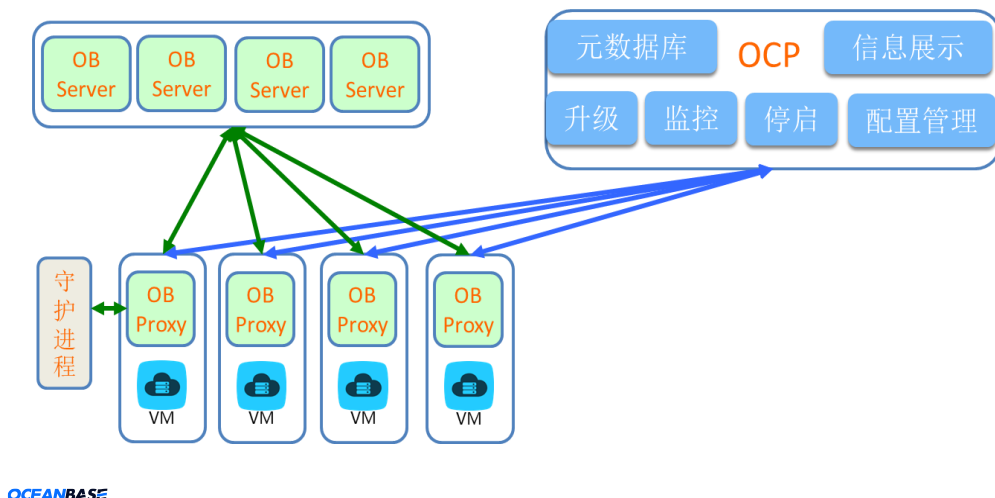


OCEANBASE

Obproxy集中部署的方式主要用于外部上云，方便外部用户使用Mysql协议访问OceanBase。外部用户使用的Mysql客户端种类繁多，Obproxy需要解决可能遇到的各种Mysql兼容问题。Obproxy与Nginx这类Web服务程序类似，多个Obproxy进程之间不需要通讯，单个Obproxy无状态，即使宕机重启也不会导致数据一致性问题，所以Obproxy实

现平滑热升级和宕机快速重启便可以基本满足HA的需求。OceanBase外部上云的整个链路如上图所示，用户的连接请求通过SLB的四层负载均衡，均匀分布到各个Obproxy上；Obproxy通过解析用户的SQL请求，做七层负载均衡，尽量将用户SQL请求转发到数据所在的OceanBase服务器上。同时OCP负责集中部署的Obproxy集群的监控，升级和配置管理等运维功能。

## 6.2 部署方式 – 客户端部署



将Obproxy作为客户端部署在应用的虚拟机上，这种方式用于阿里/蚂蚁内部上云，用户访问本地的Obproxy相比访问集中部署的Obproxy可以减少一次网络开销，能节省0.2ms~1ms的响应延时。Obproxy转发request/response时内部总耗时大约30us左右，但Obproxy能将用户请求路由到数据所在的OceanBase服务器上，从而有利于降低请求的响应时

间。

Obproxy部署在客户端时仍然通过OCP来管理，但由于应用的虚拟机特别多，应用的PE来运维Obproxy有一定的难度，所以Obproxy通过OCP做了一些批量运维工具，Obproxy周期性向OCP汇报状态和统计信息，OCP实时监控各个Obproxy的运行状态，通过OCP可以对Obproxy进行批量热升级和配置更新。同时在应用的虚拟机上部署了一个守护进程，如果发现Obproxy宕机，会立即重启Obproxy，尽量减少由于Obproxy宕机导致的服务不可用时间。

## 6.2 两种启动模式

- 测试模式 主要用于现阶段开发调试，无需依赖 ConfigServer
  - ConfigServer是OCP平台提供的OB集群物理入口管理服务，是一个web api的服务
  - 测试模式通过指定集群的RSLIST (ip列表) 来启动
- 生产模式
  - ObProxy可以通过指定 config server 提供的 config\_url 来启动，config server服务可以协助获取该集群的配置信息。同一个config server可以保存多个OB集群的RSLIST信息，使obproxy能为多个OB集群同时提供服务。
  - 在连接ObProxy时，其用户名类似 root@sys#cluster，其中 root 为用户名，sys 为租户名，cluster 为集群名

OCEANBASE

ConfigServer是OCP平台提供的OB集群物理入口管理服务，用来获取集群的连接信息。OB和客户端都是通过config\_url来跟ConfigServer交互。OB集群会将集群位置信息写入到ConfigServer中，同时，客户端指定OB集群的config\_url访问ConfigServer也可以获取该集群的物理信息，从而进行集群的访问

Config server url是一个web service api，作为ocp一部分会自动启动，ocp部署的

时候一般也会做高可用，api不会down掉

由于OceanBase支持多租户，每个租户对应一个MySQL实例。因此访问OceanBase的用户名需要指定租户名。又由于proxy支持OceanBase的多集群部署，因此通过proxy访问OceanBase服务时，还需要指定集群名，格式有四种：  
username@tenantname#clustername, 如root@trade#xxbank  
clustername:tenantname:username, 如xxbank:trade:root  
clustername-tenantname-username, 如xxbank-trade-root  
clustername.tenantname.username, 如xxbank.trade.root



## 6.2 OBProxy运行环境

- OS: Linux Redhat 7u x86-64 及以上
- OS内核: 3.10 及以上版本
- CPU: 2 核及以上
- 内存: 1G 及以上
- 磁盘空间: 对磁盘大小没有特别需求, 推荐 10G 及以上, 主要用于存放 OBProxy 的应用日志

OCEANBASE

最低配置配置要求比OBServer要低, 使用资源也较小, 资源不足的时候, 可以跟OBServer部署在一起

## 6.3 路由策略

OCEANBASE

## 6.3 和路由相关的一些基础概念

这些基础概念和OBproxy路由密切相关，根据不同的配置，Obproxy 进行综合的路由排序：

### 1. LDC配置：

1. 本地：同城同机房（IDC相同）
2. 同城：同城不同机房（IDC不同，Region 相同）
3. 异地：不同的地域（Region不同）

### 2. Observer 状态：常态 vs 正在合并

### 3. 租户的Zone 类型：读写型 vs 只读型

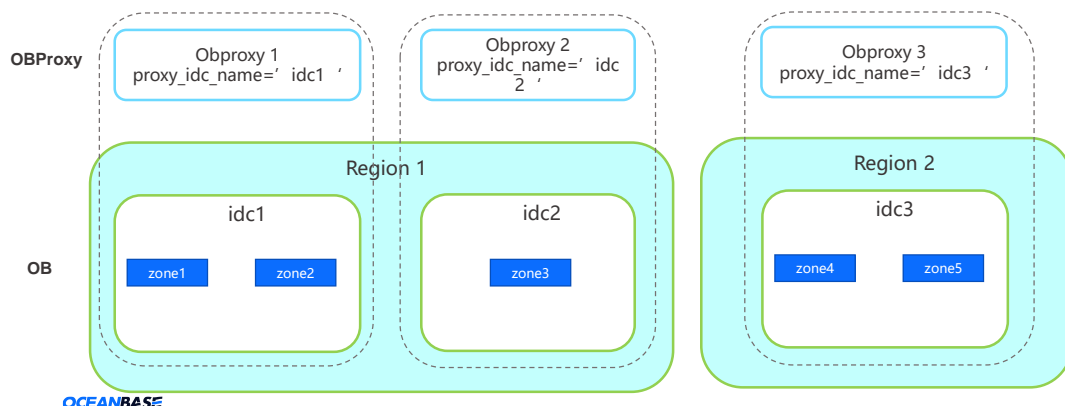
### 4. 路由精准度：优先精准度高的

OBproxy 中有目标 partition 的路由信息（PS）

OBproxy 中没有Partition的路由信息，只有租户的路由信息（TS）

## 6.3 LDC

- IDC：互联网数据中心，可以简单看成一个物理机房
- LDC：logical data center，是对 IDC 的一种逻辑划分
- Region：地域信息，通常代表一个城市，包含一个或多个IDC，每个IDC部署一个或多个zone。一个OB集群可以包含若干个Region，每个Region包含若干个IDC，每个IDC部署若干个zone



IDC: internet data center

- 互联网数据中心，简单可以看成一个物理机房
- IDC记录OB集群的机房信息

LDC: logical data center

- 是对IDC (Internal Data Center, 互联网数据中心) 的一种逻辑划分
- OB在多地多中心部署时, 会以Region和

Zone的单位对所有observer进行划分,这也是一种LDC部署,这种情况下OB的客户端路由和OB内部的RPC路由被称为LDC路由

## 6.4 LDC配置：集群的配置

设置sql:

```
alter system modify zone "z1" set
region = "SHANGHAI";
```

```
alter system modify zone "z1" set ldc =
"zue";
```

检查observer LDC设置内容是否生效:

```
select * from __all_zone;
```

OCEANBASE

zone	name	value	info
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 cluster	0 obs_jlswhee.3js
2017-12-07 18:13:43.508996	2017-12-07 18:36:18.793889	1 config.version	131261417076462
2017-12-07 18:13:43.508996	2017-12-07 18:19:41.824562	1 freeze.time	1312613181615870
2017-12-07 18:13:43.508996	2017-12-07 18:19:41.816383	1 freeze.version	2
2017-12-07 18:13:43.508996	2017-12-07 18:19:41.820905	1 global_broadcast.version	2
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 is_merge_error	0
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 last_merged.version	1
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 last_merge.status	1312614170759808
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 merge.status	1
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 privileges.version	0
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 proposal.freeze.version	2
2017-12-07 18:13:43.508996	2017-12-07 18:13:51.286383	1 time_zone_info.version	1312612831202710
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 ttp.freeze.version	2
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 version.start.time	0
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 all_merged.version	2
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 broadcast.version	2
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 ldc	0 / zue
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 is_merge_error	0
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 is_merge_error	0
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 last_merged.time	1312613252670400
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 last_merged.version	2
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 merge.start.time	1312613301815870
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 merge.status	0 / zue
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 region	0 / SHANGHAI
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 status	2 / ACTIVE
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 suspend_merging	0
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 zone.type	0 / ReadWrite
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 all_merged.version	1
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 broadcast.version	2
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 ldc	0 / zue
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 is_merge_error	0
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 is_merge_error	0
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 last_merged.time	1312612823504340
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 last_merged.version	1
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 merge.start.time	1312612823504340
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 merge.status	0 / zue
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 region	0 / SHANGHAI
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 status	2 / ACTIVE
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 suspend_merging	0
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 zone.type	0 / ReadWrite
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 all_merged.version	1
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 broadcast.version	2
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 ldc	0 / zue
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 is_merge_error	0
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 is_merge_error	0
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 last_merged.time	1312612823504340
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 last_merged.version	1
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 merge.start.time	1312612823504340
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 merge.status	0 / zue
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 region	0 / SHANGHAI
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 status	2 / ACTIVE
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 suspend_merging	0
2017-12-07 18:13:43.508996	2017-12-07 18:13:43.508996	1 zone.type	1 / ReadOnly

33 rows in set (0.08 sec)

## 6.4 LDC配置：Obproxy的配置

proxy的LDC的支持全局级别 和 session级别

- 全局级别, 配置项proxy\_idc\_name用来控制全局级别的当前IDC机房信息, 默认为空。配置项的设置可以通过启动参数/登陆修改/ocp配置项更新进行, 在proxy的启动脚本中使用-i 机房名启动传入, 或者proxy运行后通过alter proxyconfig set proxy\_idc\_name= '机房名' 设置
- session级别, 设置用户变量set @proxy\_idc\_name= 'xx' 控制session级别的当前机房信息, 默认不指定, 用户可以通过进行设置

## 6.4 LDC配置：Obproxy 的配置

ObProxy配置机房名方式：

### 1.启动时通过启动参数(推荐):

```
/opt/taobao/install/obproxy/bin/obproxy -n cif -o proxy_idc_name=gtj
```

### 2.修改proxy配置项:

```
mysql> alter proxyconfig set proxy_idc_name = 'gtj';  
Query OK, 0 rows affected (0.01 sec)
```

OCEANBASE



## 6.4 LDC配置：检查LDC的匹配情况

通过执行内部命令show proxyinfo ldc;可以检查proxy内部识别的LDC部署情况

```
mysql> show proxyinfo ldc;
```

global_ldc_name	cluster_name	match_type	regions_name	same_ldc	same_region	other_region
zue	ob2.jianhua.sjh	MATCHED_BY_NONE		[[0]"z1", [1]"z1", [2]"z2", [3]"z2", [4]"z3", [5]"z3"]		
zue	ob1.jianhua.sjh	MATCHED_BY_IDC	[[0]"SHANGHAI"]	[[0]"z1", [1]"z1", [2]"z2", [3]"z2"]		[[0]"z3", [1]"z3"]
zue	MetaDataBase	MATCHED_BY_IDC	[[0]"SHANGHAI"]	[[0]"z1", [1]"z1", [2]"z2", [3]"z2"]		[[0]"z3", [1]"z3"]

3 rows in set (0.00 sec)

## 6.3 Obproxy 主要路由策略

### 写请求:

- 写请求路由到ReadWrite zone的主副本

### 读请求:

- 强一致性读

- 弱一致性读

- 主备均衡路由策略 (默认)
- 备优先读策略
- 读写分离策略

OCEANBASE

读写默认都走主副本

## 6.3 强一致性读路由策略

- 默认情况，需要读取partition主的数据
- 即SQL必须转发到涉及partition的leader server上执行, 以此保证获取到实时最新的数据
- 强一致性读适用于对读写一致性要求高的场景

OCEANBASE

读写默认都走主副本

## 6.3 弱一致性读：配置弱一致性读

### ➤ 系统变量

- 用户设置当前租户全局系统session变量set global ob\_read\_consistency= 'weak' , 对当前租户所有会话都生效（当前会话不生效）
  - session级别, set ob\_read\_consistency= 'weak' , 只对当前正在连接的session (会话) 生效
- sql hint, 用户在select中加/\*read consistency(weak)\*/的Hint, 仅本语句生效

## 6.3 弱一致性读：主备均衡路由策略（默认）

- 可以读主也可以读备，流量按照副本均匀分配
- 弱一致性读时，或者建立连接时，或者强一致性读找不到partition时，或者强一致性读partition主不可用时，按照：本地常态 -> 同城常态 -> 本地合并 -> 同城合并 -> 异地常态

1. 选取本机房不在合并的副本；
2. 选取同地域机房不在合并的副本；
3. 选取本机房在合并的副本；
4. 选取同地域机房在合并的副本；
5. 随机选取非本地域机房不在合并的副本；
6. 随机选取非本地域机房在合并的副本；

OCEANBASE

## 6.3 弱一致性读：备优先读策略

- OBProxy 支持备优先读路由策略，通过用户级别系统变量 `proxy_route_policy` 控制备优先读路由。备优先读仅在弱一致性读时生效，且优先读 follower 而非主备均衡选择
- 使用 root 用户登录集群的 sys 租户后，运行下述语句对系统变量 `proxy_route_policy` 进行设置：  
`SET @proxy_route_policy='[policy]';`

取值为 follower first 时  
路由逻辑是优先发备（即使集群在合并状态）

```
同机房不合并的备 --> 同城不同机房不合并的备 -->
同机房在合并的备 --> 同城不同机房合并的备 -->
同机房不合并的主 --> 同城不同机房不合并的主 -->
不同城不合并的备 --> 不同城合并的备 -->
不同城不合并的主 --> 不同城合并的主
```

取值为 unmerge follower first 时  
路由逻辑是优先发不在集群合并状态的备机。（Follower 节点）

```
同机房不合并的备 --> 同城不同机房不合并的备 -->
同机房不合并的主 --> 同城不同机房不合并的主 -->
同机房在合并的备 --> 同城不同机房合并的备 -->
不同城不合并的备 --> 不同城不合并的主 -->
不同城合并的备 --> 不同城合并的主
```

注：当取其他值时，退化至普通弱一致性读主备均衡的路由逻辑

OCEANBASE

## 6.3 弱一致性读：读写分离策略

- 读写分离部署，要客户端将写请求路由到 ReadWrite Zone主副本，将弱一致性读请求路由到ReadOnly Zone
- 使用 root 用户登录到集群的业务租户，运行下述语句设置 Global 级别系统变量 ob\_route\_policy 为对应取值即可，默认取值为 readonly\_zone\_first:

```
set @@global.ob_route_policy=readonly_zone_first;
```

值	路由策略	说明
readonly_zone_first	1.本地常态读库PS -> 2.同城常态读库PS -> 3.本地合并读库PS -> 4.同城合并读库PS -> ....	默认值, 读库优先访问 优先级: zone类型 > 合并状态 > idc状态
only_readonly_zone	1.本地常态读库PS -> 2.同城常态读库PS -> 3.本地合并读库PS -> 4.同城合并读库PS -> 5.本地常态读库TS -> ....	只访问读库 优先级: 合并状态 > idc状态
unmerge_zone_first	1.本地常态读库PS -> 2.同城常态读库PS -> 3.本地常态写库PS -> 4.同城常态写库PS -> .....	优先访问不在合并的zone 优先级: 合并状态 > zone类型 > idc状态

OCEANBASE

## 6.3 OBProxy执行流程

收到用户请求后，OBProxy的执行流程：

1. parse sql, 通过简易parse, 解析出sql涉及的table name、database name
2. fetch route entry, 根据用户的tenant name、database name、table name、partition id 向observer拉取该partition的路由表
3. sort route entry, 根据各种相关属性对路由表中ip进行排序
  - read\_consistency: 强一致性读or弱一致性读
  - 目标server状态: 正在合并or常态
  - 路由精准度: PS or TS
  - IDC 匹配: 本地、同城、异地
  - Zone 类型
  - 读写分离的ob\_route\_policy取值
4. filter by congestion, 从路由表中依次尝试目标ip, 通过黑名单进行过滤
5. forward request, 将用户请求转发给目标server

OCEANBASE



## 6.3 OBProxy使用限制

proxy parser在根据SQL选择server时，有以下几点特殊的逻辑：

- proxy parser只解析Begin/START TRANSACTION/SET/和其他DML，如果遇到其他单词开头的语句，proxy的parser会直接跳过，认为该语句不包含表名
- proxy parser会按照第一条包含实体表名的statement进行路由，如果整个statement都不包含表名，则将请求发送至上一条SQL所发送的server

observer会根据执行计划的类型，来告诉proxy是否将请求路由至正确的server，如果路由失败，proxy会更新location，当前的反馈机制如下：

- server返回第一条DML的命中情况

OCEANBASE

## 6.3 示例 - 比较推荐的用法

以下几种情况（select可以等价替换成update/delete/replace/insert，下同），proxy能够将请求发送至正确的server，并且server能够按照proxy的命中情况进行反馈：

```
begin; select * from t1; commit;  
set @@autocommit = 1; insert into t1 values(); set @@autocommit = 0;  
select * from t1; insert into t2 values;  
set @@ob_trx_timeout = 10000000; begin; select * from t1; commit;
```

以下几种情况，proxy会将请求发送至上一个SQL所使用的server

```
create table t1 (id int primary key); create table t2 (id int primary key);
```

OCEANBASE

## 6.3 示例 – 不建议使用的语句

以下几种情况（第一个DML是非实体表），proxy能够将请求发送至正确的server，但是server反馈的信息可能不准，不建议使用：

```
select '1'; select * from t1;  
select '1' from dual; select * from t1;
```

以下几种情况，proxy可能能够将请求路由至正确的server，但是server端反馈信息可能不准，不建议使用：

```
create table t1 (id int primary key); insert into t1 values();
```

以下几种情况，proxy会强制将请求路由至上一次使用的server，server端反馈信息可能不准，不建议使用：

```
show warnings; select * from t1;  
show count(\*) errors; select * from t1;
```

OCEANBASE

## 6.4 DBProxy 的使用与运维

OCEANBASE

## 6.4 生产环境运行OBProxy - 守护进程

- ObProxy无状态，即使宕机重启也不会导致数据一致性，所以ObProxy在部署时都带有一个守护进程，周期性检查Obproxy的健康程度，一旦发现宕机就立即重启ObProxy
- 使用方法：使用 `obproxyd.sh -c COMMAND -e ENVIRONMENT [-n APP_NAME] [-i IDC_NAME]` 启动OBProxy

e.g.

1. `./bin/obproxyd.sh -c start -e alipay -n obpay -i zue`
2. `./bin/obproxyd.sh -c start -e offline -n obpay`

e.g.

1. `obproxyd.sh -c start -e alipay`  
在蚂蚁金服机房启动obproxy
2. `obproxyd.sh -c start -e alipay -n obtrade -i zue`  
在蚂蚁金服zue机房启动obproxy，并制定该obproxy的业务名为obtrade
3. `obproxyd.sh -c start -e inc -n obpay -i su18`  
在集团su18机房启动obproxy，并制定该obproxy的业务名为obpay

OCEANBASE

线下集群：

修改obproxyd.sh的变量：

`OBPROXY_CONFIG_SERVER_URL='http://api.test.ocp.oceanbase.alibaba.net/services?Action=GetObProxyConfig&User_ID=alibaba-inc&UID=ocpmaster'`

启动方法：`./bin/obproxyd.sh -c start -e alipay -n workshop`

Kill掉OBProxy进程，自动恢复

需要注意 测试集群proxy是通过config server url 启动，而不是指定rslist

## 6.4 OBProxy配置项

- 系统租户，通过OBProxy连接OceanBase集群
- 涉及配置项的内部命令有两种，如下：
  - `show proxyconfig`，展示proxy内部各配置项属性以及config server的配置信息
  - `alter proxyconfig set key=value`，更新指定config配置项值
- 注：更新命令只对除config server配置信息之外的其他配置项有效，config server配置信息只能通过config server来更新
- 有些配置需要重启Proxy才生效（参考 need\_reboot 这列的值）

## 6.4 OBProxy配置项

配置项可以分为3种类型况来说明:

- 第一种是proxy写到本地etc文件夹中配置文件的配置项, 这些配置项可供用户根据使用场景进行配置和更新
- 第二种是proxy内部自己使用, 对一般用户不可见的配置项, 不会注册到内部表中
- 第三种是proxy从config server中获取到的配置信息(包括版本号、meta table信息、cluster信息、bin url和更新时间), 这些信息只用来展示config server的配置, 不会注册到内部表或者dump到本地配置文件中, 并且它们全部以字符串“json\_config”开头, 查询时可以使用like进行过滤

OCEANBASE

上述的前两种配置信息会在本地生产一个配置文件, 默认保存在**proxy运行目录的etc目录下**, 文件名为**obproxy\_config.bin**, 第三种配置信息仅保存在proxy内存中。

如果用户使用[alter proxyconfig](#)内部命令,则会更新本地配置文件

**obproxy\_config.bin**

支持like模糊匹配(支持'%'和'\_'的匹配)。

## 6.4 常用OBProxy配置项

- xflush\_log\_level
- syslog\_level
- observer\_query\_timeout\_delta
- log\_cleanup\_interval
- log\_dir\_size\_threshold

OCEANBASE

实际OBProxy有上百个配置项，但实际情况中，我们很少调整这些配置项，这里提到的就是日常运维使用过程中，可能涉及到修改的参数（更多的参数可以参考官网上的OBProxy运维手册）

**xflush\_log\_level**：监控用的xflush的日志级别

**syslog\_level**：OBProxy自己的应用日志的日志级别，两者不关联，

**xflush日志主要是给监控统计信息，**

**syslog是OBProxy自身运行信息**

**observer\_query\_timeout\_delta**：关系到网络断开连接，到认为Observer不可用的delta时间

可以举例：网络不可用的时候，OB多久自动切主

**log\_cleanup\_interval**：清理OBProxy自身应用日志的间隔时间



## 6.5 DBProxy 常见问题

OCEANBASE

## 6.5 启动失败

- 机器是否存在 hostname：输入hostname -i, 确认host ip是否存在
- 目录是否存在，权限是否正确：确保当前目录下有读、写、执行的权限
- 端口是否被占用：使用obproxyd.sh启动OBProxy, 使用的端口为2883
- 启动环境是否指定正确：如果通过obproxyd.sh启动， 需要使用-e参数指定OBProxy运行环境

## 6.5 OBProxy的启动

ObProxy只要能启动成功、建立连接，就解决90%的问题

```
obproxy [OPTIONS]
-h,--help                print this help
-p,--listen_port          LPORT      obproxy listen port
-o,--optstr               OPTSTR     extra options string
-n,--appname              APPNAME    application name
-r,--rs_list              RS_LIST    root server list(format ip:sql_prot)
-c,--cluster_name         CLUSTER_NAME root server cluster name
-d,--dump_config_sql      DSQL       dump config sql to file
-e,--execute_config_sql   ESQL       exectue config sql(create tables, insert initial data)
-N,--nodaemon             don't run in daemon
-V,--version              VERSION    current obproxy version
-R,--releaseid            RELEASEID  current obproxy kernel release id
-t,--regression_test      TEST_NAME  regression test
```

OCEANBASE

## 6.5 无法建立连接

OBProxy启动成功后, 当使用客户端进行连接OBProxy的时候出错

类型	ERROR
IP PORT 错误	ERROR 2003 (HY000): Can't connect to MySQL server on '127.1' (111)
权限错误	ERROR 1045 (42000): Access denied for user 'XXXXXXXX'
租户名错误	ERROR 5160 (HY000): invalid tenant name specified in connection string
认证错误	ERROR 2013 (HY000): Lost connection to MySQL server at 'reading authorization packet', system error: 0

OCEANBASE

下面逐一进行解释

## 6.5 无法建立连接 - IP PORT错误

ERROR 2003 (HY000): Can't connect to MySQL server on '127.1' (111)

检查所需建立连接的obproxy, obproxy是否存在

## 6.5 无法建立连接 -权限错误 / 租户名错误 / 认证错误

权限错误: ERROR 1045 (42000): Access denied for user 'XXXXXXXXX'

租户名错误: ERROR 5160 (HY000): invalid tenant name specified in connection string

认证错误: ERROR 2013 (HY000): Lost connection to MySQL server at 'reading authorization packet' ,  
system error: 0

- 用户名/租户名/集群名/密码 是否正确

如果不确认是否正确, 可以直接连接observer确认该信息是否正确, 注意连接observer的时候不能

带集群名

- 本机mysql版本是否过低

MySQL 5.7.8之前版本, 用户名长度超过16字节会被截断。5.7.8版本之后版本用户长度超过32字节会被截

这里的用户名包含完整的username@tenantname#clustername

- 本地json配置集群是否和远程json文件一致

该配置文件主要用于确认你需要连接的OB集群是否存在

OCEANBASE

### A. 用户名/租户名/集群名/密码 是否正确

如果不确认是否正确, 可以直接连接observer确认该信息是否正确, 注意连接observer的时候不能带集群名

### B. 本机mysql版本是否过低

MySQL 5.7.8之前版本, 用户名长度超过16字节会被截断。5.7.8版本之后版本用户长度超过32字节会被截断。

这里的用户名包含完整的

username@tenantname#clustername

C. 本地json配置集群是否和远程json文件（configure url提供的）一致，可以通过检查本地json文件，和curl

configure url来进行比较

该配置文件主要用于确认你需要连接的OB集群(ObRegion)是否存在

搜索关键字： ObRegion

## 6.5 慢查询

- proxy慢查询? - 分析proxy日志
- server show trace? - 分析server日志
- 远程执行? 为什么?
- 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';`

OCEANBASE

关键日志:

update\_cmd\_stats -> Slow query

Slow transaction -> Slow transaction

两种日志时间记录命名方式一致

如果打印Slow query日志, 必打印Slow transaction, 无论是否存在一个业务层面显示开启的事务

使用conn\_id可以精确的定位一个连接的上的SQL执行情况



关于远程调用：

trans\_type=0 单partition DML

trans\_type=1 AC=1 的select语句

trans\_type=2 跨partition分布式事务  
(单机，或者跨机)

## 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
```

这里要强调OBProxy日志捞取并分析慢sql, 与OB集群的日志区别之处

- OBProxy会统计应用端的请求、请求发送给OBProxy, OBProxy发送给OB集群等请求的耗时
- OB集群日志统计的慢sql, 主要从数据内核的事务层面, 来分析执行效率

感谢学习

OCEANBASE